Enhanced Transport-Layer Mechanisms for MEC-Assisted Cellular Networks

Mamoutou Diarra

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Mécanismes de couche transport efficaces pour réseaux cellulaires avec nœuds de traitement en bordure du réseau (MEC)

Mamoutou DIARRA

Inria

Présentée en vue de l'obtention du grade de docteur en Informatique **d'**Université Côte d'Azur

Dirigée par : Thierry TURLETTI, Directeur de recherche, Inria
Co-dirigée par : Walid DABBOUS, Directeur de recherche, Inria
Co-encadrée par : Brice TETU, Directeur R&D, Ekinops
Soutenue le : 28 Novembre 2022

Devant le jury, composé de :
Guillaume URVOY-KELLER, Professeur, I3S
Véronique VÈQUE, Professeure, Université Paris-Saclay
Hossam AFIFI, Professeur, Télécom SudParis
Navid NIKAEIN, Professeur, Eurecom

Inria

ekinops

MÉCANISMES DE COUCHE TRANSPORT EFFICACES POUR RÉSEAUX CELLULAIRES AVEC NŒUDS DE TRAITEMENT EN BORDURE DU RÉSEAU (MEC)

Enhanced Transport-Layer Mechanisms for MEC-Assisted Cellular Networks

Mamoutou DIARRA

Jury :

Président du jury
Guillaume URVOY-KELLER, Professeur, I3S

Rapporteurs
Véronique VÈQUE, Professeure, Université Paris-Saclay
Hossam AFIFI, Professeur, Télécom SudParis

Examinateurs
Navid NIKAEIN, Professeur, Eurecom

Directeur de thèse
Thierry TURLETTI, Directeur de recherche, Inria

Co-directeur de thèse
Walid DABBOUS, Directeur de recherche, Inria

Co-encadrant de thèse
Brice TETU, Directeur R&D, Ekinops

Université Côte d'Azur

Mamoutou DIARRA

Mécanismes de couche transport efficaces pour réseaux cellulaires avec nœuds de traitement en bordure du réseau (MEC)

xvii+111 p.

Ce document a été préparé avec LaTeX2e et la classe `these-ISSS` version v. 2.10.

Impression : output.tex – 1/12/2022 – 20:41

Révision pour la classe : these-ISSS.cls,v 2.10 2020/06/24 14:16:37 mpelleau

Dedicated to my dear parents,
Ousmane Diarra and Nana Santara,
and to my son, Cheick Oumar Diarra.

Résumé

La résolution des problèmes de la couche transport dans les réseaux cellulaires est toujours un sujet d'actualité, malgré le fait que de nombreuses solutions ont été proposées au cours des dernières décennies. Une des raisons qui pourrait expliquer ce phénomène, est la difficulté d'adoption des solutions proposées dans le monde réel car elles nécessitent soit des changements drastiques dans la pile protocolaire des réseaux cellulaires (au niveau de l'UE et/ou au niveau de la station de base), soit un effort de normalisation pour que les en-têtes TCP puissent inclure des informations radio. Une autre raison est le fait que la plupart de ces solutions ne sont efficaces que dans quelques scénarios et ne parviennent pas à resoudre les problèmes de couche transport considérés d'une manière générale ou à grande échelle.

Inspiré par ces limitations et aussi par la tendance grandissante vers le traitement à la bordure des réseaux mobiles ou Multi-access Edge Computing (MEC), nous nous sommes donné comme objectif dans cette thèse de démontrer qu'un très grand nombre de problèmes de couche transport dans un réseau cellulaire avec nœuds de traitement en bordure du réseau (scénario MEC), peuvent être resolus sans nécessiter de modifications dans l'appareil de l'utilisateur final ou dans la pile protocolaire 4G/5G. À cette fin, nous proposons de nouvelles solutions d'optimisation de la couche de transport qui exploitent le service de collecte d'informations radio du MEC (nommé RNIS) et d'autres moyens fournis par l'environnement MEC pour améliorer certains mécanismes clés de la couche transport tels que, le mécanisme de demarrage lent ou slow start, le mécanisme d'évitement de congestion ou régime permanent, le mécanisme de détection des pertes, ou encore le mécanisme de contrôle de flux. Nous montrons l'efficacité de cette approche en proposant d'abord SIGMA, un algorithme de contrôle de congestion (CCA) orienté lien-montant (uplink) qui surpasse les performances de BBR et des CCAs existants en termes de débit et de délai ; puis nous proposons et implémentons MELD, un algorithme de discrimination de pertes capable d'améliorer jusqu'à 80% le débit des CCAs de type loss-based (i.e., les CCAs qui utilisent les pertes comme signaux de congestion) en cas de pertes aléatoires ; et enfin nous proposons RAPID, un proxy d'amélioration de performance (PEP) qui tient compte de l'environnement radio et qui permet de réduire les augmentations de délais de bout-en-bout de manière transparente par un facteur de 10 à 50 sans introduire une diminution de débit, et ceci, quel que soit le nombre ou le comportement des flux TCP partageant la file d'attente dédiée à l'utilisateur. Enfin, nous avons implementé les solutions MELD et RAPID sur Linux et avons validé leurs performances dans un réseau 4G basé sur OpenAirInterface et FlexRAN.

Mots-clés : Réseaux cellulaires, Multi-access Edge Computing, Algorithmes de Contrôle de Congestion, Problèmes de couche transport, Optimisation TCP, Service de collecte d'informations radio

Abstract

Addressing transport-layer issues in cellular networks is still a hot research topic, despite the fact that a wide range of solutions have been proposed over the last decades. One reason that could explain this phenomenon is the difficulties in the real-world adoption of the proposed solutions as they require either drastic changes in the cellular network protocol stack (in the UE and/or in the base station) or a standardization effort so that TCP headers can include radio information. Another reason is the fact that most of these solutions are effective only in a few scenarios and fail to mitigate the transport-layer issues considered on a global or large scale.

Inspired by these limitations and the growing trend towards Mobile and Multi-Access Edge Computing (MEC), we set out in this thesis to demonstrate that several transport-layer issues in a MEC-enabled 4G/5G network can be mitigated without requiring any modifications in the end-user's device or in the 4G/5G stack. To this end, we propose novel transport-layer optimization solutions that leverage the MEC Radio Network Information Service (RNIS) and other MEC capabilities to improve some key traditional transport-layer mechanisms such as slow start, steady state behavior, loss detection and flow control mechanisms. We show the efficiency of this approach by first proposing SIGMA, an uplink-oriented Congestion Control Algorithms (CCA) that outperforms BBR and existing CCAs in terms of throughput and delay; then we propose and implement MELD, a loss discrimination algorithm that can improve the throughput of loss-based CCAs by up to 80% in case of wireless/random losses; and lastly we propose RAPID, a RAN-aware Performance Enhancing Proxy (PEP) capable of transparently reducing the increase in delay by a factor of 10 to 50 without lowering the throughput, regardless of the number or the behaviors of the TCP flows sharing the same per-user buffer. Finally, we implement MELD and RAPID in Linux and validate their performance in a real-world 4G network based on OpenAirInterface and FlexRAN.

Keywords: Cellular networks, Multi-access Edge Computing, Congestion Control Algorithms, Transport-layer issues, TCP optimization, Radio Network Information Service

Acknowledgments

First and foremost, I would like to thank Ekinops and the french government for funding this thesis. All of this would not have been possible without my encounter with Dr. Amine Ismail who believed in me and took me in his team as a PhD candidate. I am immensely grateful to him, for being a great mentor and for co-supervising my research.

I would also like to thank my advisors at Inria, Dr. Thierry Turletti and Dr. Walid Dabbous for their support and guidance during this thesis. It has been an honor to be their PhD student. I cannot thank them enough for everything they taught me. Thanks to them, I learned what it means to be a researcher. Even during difficult times, they were present, supportive and understanding.

I also want to thank the fellow PhD students and friends I met at Inria during my PhD : Houssam Elbouanani, Othmane Belmoukadam, Bernard Tamba Sandouno, Mariella Jreidy, Raza Ul Mustafa and Hari Kuttivelil. Thank you guys for all the insightful and sometimes philosophical discussions we had during our coffee breaks.

My time at Ekinops was enjoyable thanks to my dear colleagues : Mireille Remy, Philippe Coquelet, Thierry Masson, Luc Ottavj, Imed Lassoued, Riccardo Ravaioli, Franck Messaoudi, Brice Tetu, David Marie, Denis Salgon, Jacques Webert, Romain Lhuissier, Mayeul Mathias, Jean-Michel Pelletier, Jean-Pierre Stierlin, Jean-Francois Roux, Francois Coutant, Anais Hachmanian, Quentin Jacquemart, Alain Enout and Alain TranThanh. They encouraged me during my PhD and were supportive during difficult times.

Of course, I am immensely grateful to my family for everything they have done for me. All of this work would not have been possible without their support. I could never thank my father and my mother enough. They believed in me and sacrificed everything to make sure I study in the best possible conditions. A special thanks to my big brother, Moussa Diarra, who inspired me to follow my passion for telecommunications.

I couldn't finish this section without giving a special thanks to my wife, Djarafa Tambadou. She is the person who best knows the ups and downs I went through during these years. She listened to every absurd idea that crossed my mind during these three years. I really commend her for her patience and her cheerful spirit. Her support and understanding has been crucial for my personal and professional development. Thank you for everything.

Contents

List of Abbreviations

3GPP	Third Generation Partnership Project
4G	Fourth Generation of cellular networks
5G	Fifth Generation of cellular networks
ACK	Acknowledgement
ADC	Analog-to-Digital-Conversion
AF	Application Function
AIMD	Additive Increase Multiplicative Decrease
AM	Acknowledged Mode
AMC	Adaptive Modulation and Coding
AMF	Access and Mobility Management Function
APP	Application
APN	Acess Point Name
AQM	Active Queue Management
ARQ	Automatic Retransmission Request
AUSF	Authentication Server Function
BBR	Bottleneck Bandwidth and RTT
BBU	Baseband Unit
BDP	Bandwidth Delay Product
BS	Base Station
BW	Bandwidth
CA	Congestion Avoidance
CC	Congestion Control
CCA	Congestion Control Algorithm
CDF	Cumulative Distribution Function
CDN	Content Delivery Network
CN	Core Network
CoDel	Controlled Delay
CQI	Channel Quality Indicator
CQIC	CQI Congestion Control
CU	Centralized Unit
CWND	Congestion Window
DAC	Digital-to-Analog-Conversion
DCI	Downlink Control Information
DL	Downlink
DN	Data Network
DRB	Data Radio Bearer
DTX	Discontinuous Transmission
DU	Distributed Unit
DUPACK	Duplicated ACK
E2E	End-to-End

ECN	Explicit Congestion Notification
EMM	EPS Mobility Management
eNB	Evolved NodeB
EPC	Evolved Packet Core
EPS	Evolved Packet System
ESM	EPS Session Management
ETSI	European Telecommunication Stabdard Institute
EUTRAN	Evolved UMTS Terrestrial Radio Access Network
FDD	Frequency Division Duplex
FQDN	Fully Qualified Domain Name
gNB	Next Generation NodeB
GPRS	General Packet Radio Service
GSM	Global system for Mobile
GTP	GPRS Tunneling Protocol
HARQ	Hybrid Automatic Repeat ReQuest
HSS	Home Subscriber Server
IETF	Internet Engineering Task Force
IMSI	International Mobile Subscriber Identity
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISI	Inter Symbol Interference
ISP	Internet Service Provider
IW	Initial Window
L1	Layer 1 (Physical layer)
L2	Layer 2 (Link layer)
L4	Layer 4 (Transport layer)
LADN	Local Area Data Network
LCM	Life Cycle Management
LD	Loss Detection
LR	Loss Recovery
LSS	Limited Slow Start
LTE	Long-Term Evolution
MAC	Medium Access Control
Mbps	Mega-Bits per second
MCS	Modulation and Coding Scheme
MEC	Multi-access Edge Computing
MELD	MEC-based Edge Loss Discrimination
MEO	MEC Orchestrator
MEP	MEC Platform
MEPM	MEC Platform Manager
MIMD	Multiplicative Increase Multiplicative Decrease
MIMO	Multiple Input Multiple Output
MME	Mobile Management Entity
MSISDN	Mobile Subscriber ISDN Number
NACK	Negative-ACK
NAS	Non-Access Stratum

NEF	Network Exposure Function
NF	Network Function
NR	New Radio
NRF	Network Repository Function
NS3	Network Simulator 3
NSSF	Network Slice Selection Function
OAI	OpenAirInterface
OFDM	Orthogonal Frequency Division Multiplexing
OFDMA	Orthogonal Frequency Division Multiple Access
PCF	Policy Control Function
PDCP	Packet Data Convergence Protocol
PDN	Packet Data Network
PDU	Protocol Data Unit
PEP	Performance Enhancement Proxy
PF	Proportional Fair
PGW	Packet Data Network Gateway
PHY	Physical layer
Pkt	Packet
PLMN	Public Land Mobile Network
PRB	Physical Resource Block
QCI	QoS Class Identifier
QFI	QoS flow Indicator
QoE	Quality of Experience
QoS	Quality of Service
QUIC	Quick UDP Internet Connection
RACK	Recent Acknowledgment
RAN	Radio Access Network
RAPID	RAN-aware Proxy-based flow control for High Throughput and Low Delay
RB	Resource Block
RBG	Resource Block Group
RFC	Request For Comment
RLC	Radio Link Control
RNIS	Radio Network Information Service
RRC	Radio Resource Control
RRH	Remote Radio Head
RRM	Radio Resource Management
RRU	Remote Radio Unit
RTO	Retransmission Timeout
RTT	Round Trip Time
SACK	Selective ACK
SCS	Sub-Carrier Spacing
SDAP	Service Data Adaptation Protocol
SDF	Service Data Flow
SDN	Software Defined Network
SD-RAN	Software Defined RAN
SDU	Service Data Unit

SFQ	Stochastic Fairness Queuing
SGW	Serving Gateway
SIGMA	Simple Increase in Goodput based on MEC Awareness
SINR	Signal to Interference plus Noise Ratio
SMF	Session Management Function
TBS	Transport Block Size
TCP	Transmission Control Protocol
TDD	Time Division Duplex
TFT	Traffic Flow Template
TM	Transparent Mode
TR	Technical Report
TS	Technical Specification
TTI	Transmission Time Interval
UCI	Uplink Control Information
UDM	Unified Data Managemen
UDP	User Datagram Protocol
UE	User Equipment
UL	Uplink
UM	Un-Acknowledgement Mode
UMTS	Universal Mobile Telecommunications System
UPF	User Plane Function
WS	Window Scaling

List of Contributions

Conferences :

- Mamoutou Diarra, Walid Dabbous, Amine Ismail and Thierry Turletti, **“Cross-layer Loss Discrimination Algorithms for MEC in 4G networks”**, 22nd IEEE International Conference on High Performance Switching and Routing (HPSR), June 2021, Paris, France

- Mamoutou Diarra, Walid Dabbous, Amine Ismail, and Thierry Turletti, **“RAN-aware Proxy-based Flow Control for High Throughput and Low Delay eMBB”**, In Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM ’21), November 2021, Alicante, Spain

Journals :

- Mamoutou Diarra, Walid Dabbous, Brice Tetu, Thierry Turletti, **"RAPID : A RAN-aware performance enhancing proxy for high throughput low delay flows in MEC-enabled cellular networks"**, Computer Networks, September 2022

In preparation :

- **"SIGMA : A Simplistic Uplink oriented Congestion Control Algorithm based on MEC Awareness "**

CHAPTER 1

Introduction

At its inception, the Transmission Control Protocol (TCP) was based on the principle that the underlying network should be considered as a black box, meaning that reliable end-to-end delivery should be ensured without needing any explicit information from intermediate devices or lower layers. This principle, embodied by early TCP variants also known as congestion control algorithms (CCAs) has been true until the introduction of Explicit Congestion Notification (ECN) in the early 90s, which advocates the use of explicit network layer signals (triggered by intermediate routers) in order to help TCP avoid unnecessary packet losses [43] [95]. This additional mechanism was mostly driven by the evolution in application requirements at that time [95]. More specifically, instead of dealing only with sheer download flows which have little or no sensitivity to delay or loss of individual packets, the Internet had also to cope with the increasing use of delay-sensitive and interactive services such as telnet or web-browsing whose requirements where not taken into account in the design of early TCP built-in mechanisms and CCAs. Naturally, following the same pattern, with the evolution of the Internet and end-users flows characteristics, other built-in mechanisms (e.g., Fast-Recovery, Selective-Acknowledgments/SACKs, Recent-Acknowledgment/RACK, etc.) as well as more refined and adapted CCAs (e.g., NewReno, Vegas, Cubic, etc.) have been added to TCP. Although most of these new techniques prove effective over wired networks, it is worth noting that they were designed with little or no considerations for cellular networks. This owes to the use of Ethernet access technologies (i.e., cables or WiFi) by the vast majority of end-user in the early days of the Internet. Even though this trend has gradually shifted in the favor of cellular access technologies due to the huge advances in cellular technologies over the last two decades [112] [12, 28], TCP has remained with its core built-in mechanisms which are still at the time of this writing, the causes of serious performance issues in today's next-generation cellular networks (i.e., fourth and fifth generations, 4G and 5G).

Because of the characteristics of the radio environment, it is clear that cellular access technologies will always be inferior to traditional wired technologies (i.e., twisted pair, fiber) in terms of speed and latency [62]. Nevertheless, throughout the different generations of cellular networks, spectral efficiency and latency have been tremendously improved among other thanks to the use of advanced multiple access techniques; effective link/physical layer mechanisms; and various design choices such as the use of deep buffers at the base stations [62] to cope with the fast-varying nature the radio link; or the introduction of the so called bearers to separate users traffic and ensure Quality of Service (QoS). For example, Orthogonal Frequency Division Multiple Access (OFDMA) backed with Adaptive Modulation and Coding (AMC) and retransmissions at lower layers [6] enables a 20 MHz 4G radio access network (RAN) with no Carrier Aggregation (CA) to deliver in theory up to 100 Mbps Downlink data rate [57] with a two-way user-plane latency of approximately 8 ms [6]. More importantly, 5G New Radio (5G NR) which is the Radio Access Tech-

nology (RAT) used by the recently introduced 5G networks is expected to provide even largely superior performances. Compared to 4G RAT, NR can operate in a much higher frequency band with 20 times larger bandwidth (up to 400 MHz) [4]. Such large bandwidths, along with flexible sub-carrier spacing (or Numerology) and CA (maximum of 16 carriers [2]) enable 5G NR to allow in theory up to 20 Gbps peak data rates and user plane latencies of up to 1-millisecond [72].

However, despite being very impressive, the physical layer data rates and latencies displayed by 4G and 5G technologies are meaningless if they cannot be fully exploited by transport and application layer protocols. This is unfortunately the case since some key mechanisms and design choices in 4G/5G stacks don't match with TCP's basic assumptions and built-in mechanisms.

For instance, loss-based CCAs (used by the majority of today's TCP flows [81]) are commonly known for continuously increasing their sending rate as long as no packet loss occurs. Therefore, they end up creating long queues at the base stations due to the presence of deep buffers, thus penalizing any concurrent delay-sensitive or short flow. This phenomenon, first coined by Gettys in late 2010 [47] as bufferbloat, is so prevalent in cellular networks * that the Third Generation Partnership Project (3GPP) has defined an additional sublayer (Service Data Adaptation Layer : SDAP) as well as new QoS flow indicators (QFIs) so that different flows sharing the same PDU session can be mapped to different Data Radio Bearers (i.e., different Buffers at Radio Link Control Layer) based on predefined Service Data Flow templates (SDF templates) [8, 9]. Although these improvements reduce bufferbloat effects in some scenarios, they are not sufficient given that a single User Equipment (UE) can have several flows in parallel, which can be associated to up to 64 different QFIs but only a maximum of 8 Data Radio Bearers (DRB) is allowed per user [62]. Therefore, following the pigeon hole principle, some flows will inevitably end up sharing the same DRBs and suffering from bufferbloat.

Another example is the TCP's Retransmission Timeouts (RTO) and Throughput collapse caused by retransmissions at lower layers. In 4G and 5G stacks, in addition to the forward error correction performed by channel coding, some retransmission techniques are used in the Radio Link Control (RLC) and Medium Access Control (MAC) layers, namely, Automatic Repeat Request (ARQ) and Hybrid Automatic Repeat Request (HARQ). HARQ uses a stop-and-wait technique in which the corrupted block is retransmitted upon the reception of a Negative Acknowledgment (NACK). ARQ, on the other hand, relies on a window-based selective repeat technique in order to retransmit missing blocks resulting from residual HARQ errors (e.g., misinterpreted NACKs or maximum number of HARQ retransmission attempts). Depending on the RLC operation mode (i.e., AM, UM, TM), these two retransmission techniques affect TCP throughput in different ways. In case of RLC Acknowledged Mode (AM) in which both ARQ and HARQ are used, a large number of retransmissions, due to a persistent bad radio condition, creates packet accumulation at the RLC buffer which eventually leads to an RTO, causing a severe reduction in TCP throughput (also termed undesired slow start in [59]). In RLC Unacknowledged Mode (UM), after a fixed number of unsuccessful HARQ attempts, the received data (i.e., both ACKed and corrupted blocks) are delivered to the upper layer [90]. In this case, the missing blocks naturally trigger duplicate Acknowledgments at the TCP layer, which eventually lead to unnecessary throughput throttling and/or delay increase [90]

Mitigating TCP performance issues in cellular networks has been a hot topic for a while. Several approaches have been proposed in the literature over the years. From traditional delay

∗. Haiqing Jiang et. Al observed up to 10 seconds end-to-end delays in some commercial networks in the US [67]

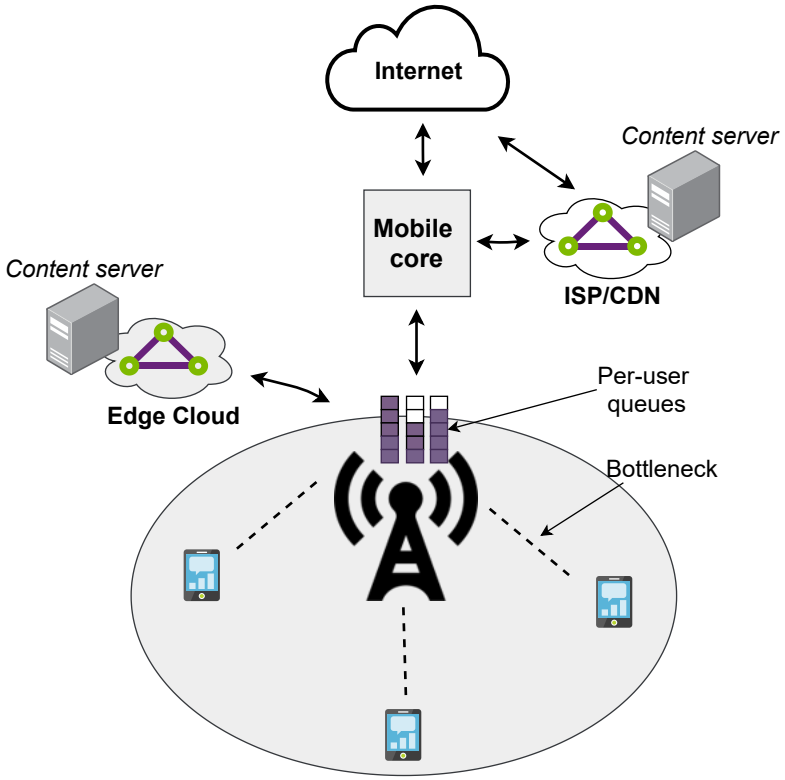


Figure 1.1 – Bottleneck location in today's cellular networks

and model-based end-to-end CCAs (e.g., Vegas, BBR etc.) to more cellular-access-oriented approaches such as Performance Enhancing Proxies (PEPs) and cross-layer CCAs. The latter approach, which requires modifications at both the UE and the server in order to leverage radio information at TCP layer is being increasingly proposed in recent papers [54], owing to the simplicity of accessing and decoding radio information at the UE. Although most of these solutions improve the utilization of the radio link, they fail to solve the bufferbloat issue in case of multiple parallel flows or when a delay-sensitive flow shares the same buffer with a greedy flow that uses a loss-based CCA. Moreover, the adoption of such solutions in real-world commercial networks is quite difficult, since they involve kernel modifications in the end-user's device.

However, thanks to the growing trend of bringing content close to the end-user and the future adoption of the Mobile/Multi-Access Edge Computing (MEC) [40] framework recently defined and standardized by the European Telecommunications Standards Institute (ETSI), new opportunities and approaches for enhancing transport layer protocols can be exploited. In fact, over the past few years, Internet content and resources have increasingly been located close to the end users by means such as Content Delivery Networks (CDN) or edge computing. In such a context and as today's cellular networks rely on well-provisioned backhaul, the radio link between the user and the base station, as illustrated in Figure 1.1, most often becomes the bottleneck [54]. As mentioned earlier, this radio bottleneck is characterized by fast varying data rates and large buffers, which are not in line with the philosophies of traditional CCAs. With that being said, today, with the services provided by ETSI MEC, the applications deployed at the edge can easily receive accurate real-time information about the radio bottleneck, and this without involving the end-user's device

(unlike traditional cross-layer CCA approach). Put another way, ETSI MEC can explicitly provide the transport layer with accurate information about the actual capacity of the bottleneck, which up to now, has always been estimated by CCAs through continuous probing cycles. Therefore, it is not an overstatement to say that ETSI MEC brings a new range of possibilities/opportunities for enhancing traditional transport layer mechanisms. The challenge that remains is how to re-think traditional transport layer approaches and mechanisms so that they can benefit from the new opportunities brought by MEC and edge computing.

1.1 Objectives and Contributions

The main objective of this thesis is to propose novel transport-layer mechanisms that exploit the services and architectural changes brought by MEC in order to mitigate some well-known transport-layer issues that specifically occur in cellular networks. More specifically, we have considered the following issues : the cellular uplink utilization issue and the on-device bufferbloat issue, which occur under conservative and aggressive uplink TCP traffic, respectively ; the throughput degradation issue that affects loss-based CCAs in case of wireless random losses ; the bufferbloat and CCA fairness issues on the downlink which occur when a greedy TCP flow shares the same RLC buffer with a delay-sensitive flow or with a flow using a delay-based or a more conservative CCA. We have thoroughly investigated these issues in this thesis, and have proposed novel and efficient solutions that address them by considering specific transport-layer metrics and by exploiting specific cellular RAN information via the ETSI MEC's Radio Network Information Service (RNIS). Overall, our contributions throughout this thesis can be summarized as follows :

- We identified and extensively studied the limitations of traditional CCAs, existing cross-layer CCAs and most popular cellular-network-oriented transport-layer optimization solutions over today's 4G/5G stack.

- We analyzed the edge computing paradigm and the ETSI MEC framework from a transport-layer perspective, which allowed us to identify some new opportunities that could help enhance traditional transport-layer mechanisms. We leveraged one of these new opportunities, more precisely, the knowledge of the bottleneck location, in order to propose a new cross-layer and uplink-oriented CCA named **"SIGMA"**. We implemented SIGMA in ns-3 and showed that it offers a better tradeoff between goodput and delay than NewReno, Cubic and BBR.

- We highlighted to what extent some specific information transmitted over the RAN can be used as relevant indicators of congestion in the radio access network. This allowed us to propose a new loss discrimination scheme named **"MELD"**, that allows loss-based CCAs such as NewReno and Cubic to discriminate packet losses based on up-to-date RAN information collected via the RNI service. We implemented MELD as a plugin to picoquic [113] (a well-known IETF QUIC implementation) and evaluated its performances in a open-source LTE environment. These works led to the publication of a paper in the 2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR), entitled **"Cross-layer Loss Discrimination Algorithms for MEC in**

4G networks" [36].

- We proposed a RAN-aware performance enhancing proxy, named **"RAPID"** that intercepts TCP connections and distributes proportionally the available RAN bandwidth among the active flows. In order to do that, we relied on the MEC RNI service and on packets arrival rates to estimate the aggregated RAN bandwidth and to categorize the concurrent TCP flows in the UE. This approach does not involve the end user unlike existing cross-layer CCAs that introduce computational overhead and additional power consumption at the UE as they require client-side modifications in order to introduce radio information into the TCP header. We implemented RAPID in ns-3 and showcased its efficiency in mitigating the bufferbloat issue throughout various scenarios. The obtained results were published in the Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '21), under the title **"RAN-aware Proxy-based Flow Control for High Throughput and Low Delay eMBB"** [35].

- We extended [35] and implemented our proposed RAPID mechanism in Linux based on well-known open-source projects. This allowed us to evaluate our solution in a real-world 4G network, to study and clarify its behavior in the presence of certain non-TCP traffic such as QUIC, and to further explore the demand-aware fairness concept, which is one of the key concepts embodied in its design. These works led to the publication of an extended paper in the Computer Networks journal (COMNET), entitled **"RAPID : a RAN-aware Performance Enhancing Proxy for High Throughput Low Delay Flows in MEC-enabled cellular networks"** [34].

1.2 Thesis outline

The rest of this thesis is organised as follows :

In Chapter 2, we revisit the traditional transport layer mechanisms and congestion control approaches that are currently used on the Internet. This is followed by a brief background on the 4G/5G stack, which is necessary in order to understand how user's data traffic is handled in today's cellular networks. After that, we extensively describe the root causes of the major transport layer issues in 4G/5G networks and present some existing mitigation solutions.

In Chapter 3 we explore the edge computing paradigm from a transport-layer perspective, present the ETSI MEC framework, and identify some new opportunities that can be exploited in order to enhance existing transport-layer mechanisms. Next, we introduce SIGMA, our new uplink-oriented CCA that leverages the architectural changes brought by MEC and the availability of RAN information on the user's device in order to improve the performance of uplink TCP traffic. We present the design of SIGMA as well as its implementation details and compare it in terms of performance against NewReno, Cubic and BBR. Lastly, we discuss SIGMA's potential limitations and propose some improvements for the future.

In Chapter 4, we focus on the loss differentiation issue that hinders loss-based CCAs performances and discuss the limitations of existing solutions in the context of cellular networks. Next,

we present MELD, a new Loss Discrimination Algorithms that is based on MEC and adapted to 4G/5G physical and link-layer mechanisms. We explain the key ideas behind MELD and describe how we implement it as a plugin to picoquic. After that, we evaluate the efficiency and overhead of MELD on NewReno and Cubic in a 4G OpenAirInterface deployment. Finally, we end the chapter by a brief summary in which we highlight the benefits of using MELD in real-world scenarios and then discuss MELD's limitations in controlling the greedy behavior of loss-based CCAs as well as the issues that may arise from such a behavior.

In Chapter 5, we highlight the difficulties of addressing the downlink bufferbloat issues in today's cellular networks given the ever-growing number of CCAs that exist on the Internet and that could compete for downlink bandwidth. Next, we discuss the limitations of existing bufferbloat mitigation solutions and then propose a transparent and CCA-agnostic bufferbloat mitigation solution, named RAPID. We describe the design and implementation details of RAPID and then evaluate its performance in a simulated 5G environment in a real-world 4G environment based on openAirInterface. Lastly, we discuss the observed limitations and implemented workarounds.

In Chapter 6, we summarize the conclusions of this thesis and present some potential future works.

CHAPTER 2

Understanding transport layer issues with 4G/5G access networks

In this chapter we first review the transport layer in the TCP/IP stack and recall how congestion control is currently done in the wired world. A particular attention is given to TCP, and we explore its most popular Congestion Control Algorithms (CCAs). After that, we present the 4G and 5G stacks, highlight the importance of the techniques they use in the Radio Access and Core Network as well as their consequences for TCP at the transport layer. This is necessary in order to understand why mitigation solutions are needed. This chapter ends presenting current solutions for improving link utilization and mitigating bufferbloat in cellular networks.

2.1 Transport layer and congestion control on wired Internet

The main goal of the transport layer in the TCP/IP stack is to provide the upper application layer with an end-to-end communication channel. More specifically, it enables applications on different hosts to communicate in an end-to-end manner and remain completely agnostic to the lower layers or the network infrastructure in between. Two major transport layer protocols dominate today's Internet traffic, the User Datagram Protocol (UDP) and the Transmission Control Protocol (TCP). While the former offers an unreliable and unordered connectionless data delivery channel, the latter provides a reliable and in-order connection-oriented data delivery channel. With no surprise, a large number of applications and services on the Internet rely on TCP because of its reliability (e.g., HTTP, FTP, SMTP, SSH etc.). According to recent statistics, TCP is still by far the most widely used transport layer protocol globally. To fully understand this popularity, it is important to break down the protocol and go through its basic foundations and building blocks.

In the early days of TCP, it was possible to determine TCP behavior simply based on the underlying network conditions, because the protocol was strongly tied to a specific congestion control algorithm (e.g., Tahoe, Reno or NewReno [41]) with specific built-in mechanisms for start-up (i.e., Slow Start), flow control, loss detection and recovery. Today, because of the increasing

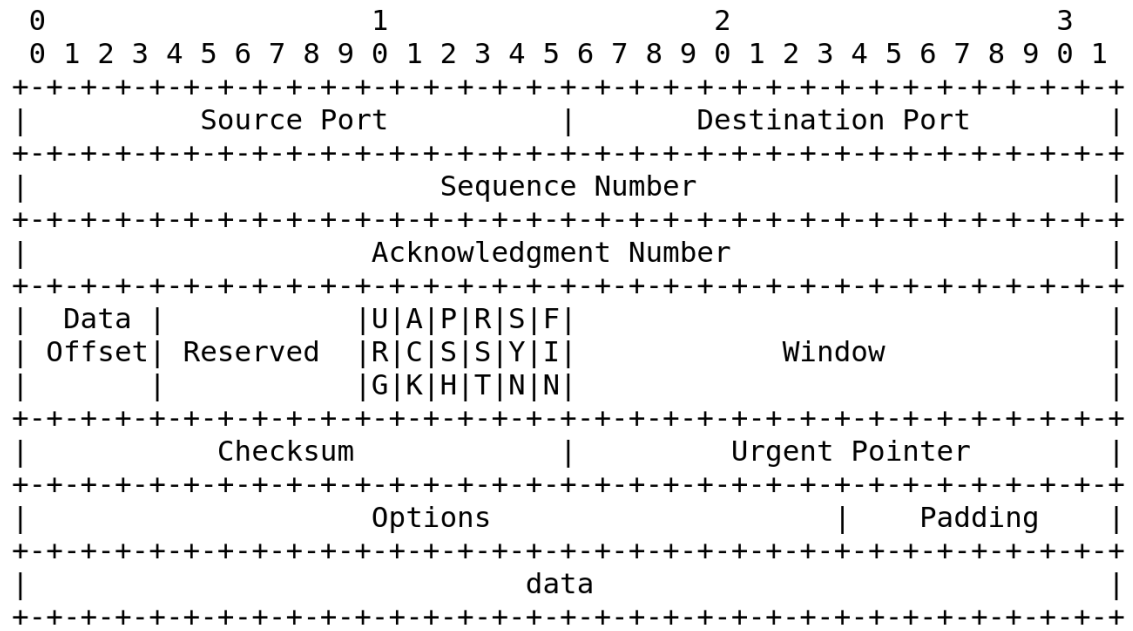


Figure 2.1 – TCP Header from RFC 793

number of CCAs, the large number of authorized TCP options and the plug-and-play nature of new TCP mechanisms, it is necessary to make a distinction between TCP's built-in mechanisms and its CCAs. Basically, a typical TCP implementation relies at least on four built-in mechanisms or algorithms :

The Slow Start (SS) algorithm : It defines how the data rate of the sender should evolve just after the establishment of the connection. In the standard SS mechanism, after the negotiation of the Maximum Segment Size (MSS) during the 3-way handshake, the sender starts with an Initial Congestion Window (i.e., initial number of segments to be sent at once during one Round Trip Time) of 4 or 10 MSS and increases the value by 1 at each acknowledgment (ACK) until the ssthresh (initialized to a relatively high value) is reached or a loss event occurs. At this point, the Congestion Window (cwnd) is halved and TCP proceeds with the Congestion Avoidance (CA) phase which is specific to the CCA in use. Apart from the standard SS mechanism, another algorithm known as Hybrid Slow Start (HyStart [53]) is currently used by some CCAs. Hystart addresses the overshooting problem of the exponential growth in the standard Slow Start. Basically, it determines a safe exit point where the Slow Start phase can be exited based on the increase in delay and the ACK train length (i.e., the sum of inter-arrival times of all the closely spaced ACKs within an Round Trip Time) [53]. However, because of Round Trip Time (RTT) fluctuations and transient queue buildup, Hystart often exits Slow Start prematurely. To alleviate the impact of this behavior, HyStart++ [16] which adds an additional Limited Slow Start phase (LSS) has been proposed. The additional LSS phase takes place just after the algorithm exits SS based only on delay increase. During this phase, LSS grows the cwnd faster than Reno's additive increase, but much slower than traditional slow start until the first packet loss occurs, at which point TCP enters the CA phase [16].

The Flow Control mechanism : The main goal of Flow Control is to prevent the sender from overwhelming the receiver. This is done on the receiver side through the Window field of the TCP

header (see Figure 2.1). Technically speaking, the receiver indicates in the 16-bits window field of each acknowledgment the amount of bytes it can receive (based on its available buffer space), and on the server side, the send window (i.e., maximum number of bytes the sender is allowed to send at once) is set to the minimum of the congestion window and the indicated receive window. With this mechanism, the maximum reachable number of bytes in flight (i.e., Bytes that are not yet ACKed) is controlled by the receiver. By default, this number is limited to 2^{16} bytes, but with the Window Scaling option (WS), a scaling factor of up to 14 can be used, which brings the maximum value to 1GB (i.e., 2^{16+14} bytes) [20].

The loss Detection and Recovery (LD/LR) mechanism : This allows TCP to detect packet losses and retransmit the missing data. By default, a segment is declared lost if no ACK for this segment is received until the expiration of its Retransmission Timer or RTO (initially set to 1s and computed later based on RFC 6298), at which point, cwnd is set to 1 MSS and TCP returns to Slow Start. To speed up the process and avoid waiting for an RTO, the initial version of TCP (TCP Tahoe) relies also on the Fast-retransmit algorithm[37], which infers that segment $N + 1$ is lost after receiving three duplicate ACKs for segment N. Upon loss detection, the missing segment is retransmitted, then cwnd is set to 1 MSS and TCP goes back to Slow Start. The issue with Fast-retransmit is that it completely drains the pipe each time a single loss occurs. As a countermeasure to this behavior, the Fast-Recovery algorithm (first used in Reno) has been proposed. In the Fast-Recovery algorithm, the cwnd is not anymore set to 1 MSS upon three duplicate ACKs, but halved (i.e., $cwnd = cwnd/2$) and TCP does not go back to Slow Start. However, because of the cumulative nature of TCP acknowledgments, it is worth noting that both Fast-retransmsit and Fast-Recovery can provoke as many cwnd reductions as the number of segments that are lost in the same window. For instance, let's consider a window of N segments $(1, 2, 3, ..., N)$ and let's consider segments 2 and 3 are lost. In this case, after receiving three duplicate ACKs for segment 1, the Fast-Recovery algorithm halves cwnd and retransmit segment 2. At this point, the sender expects to receive an ACK that acknowledges all the outstanding data (i.e., up to segment N), but instead receives an ACK for segment 2 and after three duplicate ACKs, it halves cwnd again and retransmits segment 3. To avoid these unnecessary window reductions, Fast-Recovery has been improved (in NewReno [52]) so that it can take partial acknowledgments into account. A partial acknowledgment is an ACK that comes after a retransmission and does not acknowledge all the outstanding data (i.e., it acknowledges data up to just before the second lost segment). Upon receipt of such an ACK, the enhanced Fast-Recovery algorithm directly retransmits the next segment without halving the congestion window. However, the problem with Fast-Recovery, is that it requires one RTT to detect each individual packet loss, and is limited to only one retransmission per RTT. In fact, because of the cumulative nature of TCP acknowledgments, the sender cannot detect multiple lost packets at once. This issue has been alleviated thanks to the addition of the Selective Acknowledgment option (SACK) [44]. As its name suggests, SACK allows the receiver to selectively acknowledge blocks of correctly received segments (as opposed to the traditional cumulative acknowledgment). This allows the sender to know which segments have been acknowledged and which are still in flight. Therefore, it becomes possible to detect or retransmit multiple lost packets per RTT. Technically speaking, the SACK option allows the receiver to acknowledge a maximum of four blocks in a single ACK [44]. With this information, the sender can easily detect gaps in the sequence spaces between the acknowledged blocks and retransmit all the missing segments at once. In addition to the algorithms and techniques mentioned above, it is also important to mention the RACK [31] (Recent Acknowledgment) loss detection algorithm, which is more efficient on networks with high packet reordering rate (where TCP default 3-duplicate-ACKs loss detec-

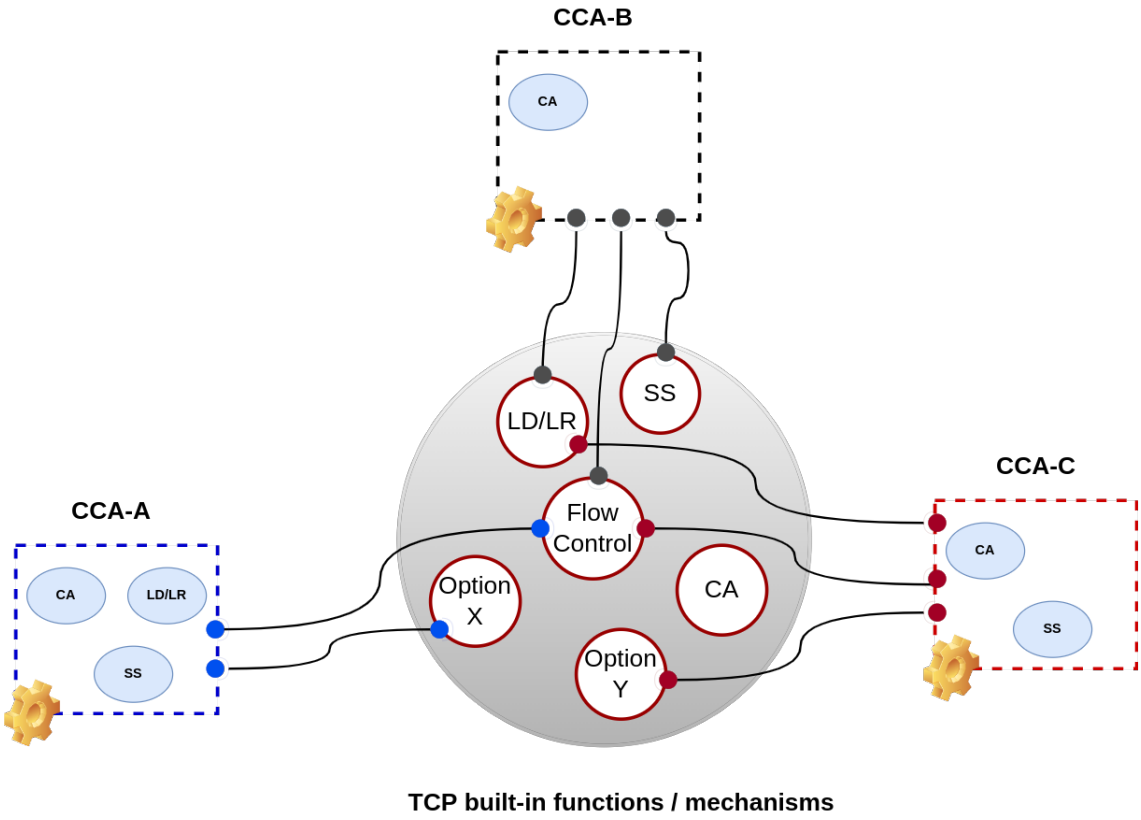


Figure 2.2 – Abstracted view of the relationship between CCAs and TCP functions

tion might declare losses too early). RACK uses a time-based loss detection mechanism; instead of counting duplicate ACKs, it uses the most recently delivered packet's transmission time to determine if some packets sent before that time should be considered lost or reordered. A packet is marked as lost if this packet has been sent a certain window of time before the most recently delivered/acknowledged packet [31]. This window of time is called the reordering window and can be customized based on the degree of reordering in the network. Its value is set by default to RTT/4, which proves to be an empirically efficient value according to authors in [31].

The Congestion Avoidance (CA) algorithm : It dictates the steady state behavior, i.e., the behavior of the flow after the slow start phase. Most congestion control algorithms differ by the CA mechanism they use. For instance, the only difference between Reno and Cubic lies in their respective CA algorithms. While Reno relies on Additive Increase Multiplicative Decrease (AIMD), which increases the congestion window value by 1 MSS every RTT and divides it by 2 in case of packet loss, Cubic replaces this linear AIMD growth function by a cubic function in order to improve TCP scalability over fast and long delay links and speed up cwnd growth after loss events.

The presented mechanisms are used as input by the CCA (on the sender side), which regulates the sending rate of TCP segments from the moment the connection is established to the moment it is closed. It is important to note that, aside from the way the Flow Control mechanism is done (which is receiver-based), a CCA can use its own built-in techniques to perform the other required mechanisms (i.e., SS, Loss Detection/Recovery and CA). Figure 2.2 shows an abstracted view of

the relationship between the CCAs and the built-in mechanisms. Indeed, over the years, several dozens of CCAs have been proposed in the literature, but they can all be classified into three main categories : loss-based, delay-based, and rate-based [82]. Therefore, a thorough description of these three categories is necessary in order to understand how the TCP servers on the Internet perform Congestion Control (CC) :

Historically, the first proposed CCAs were all loss-based (e.g. Tahoe, Reno etc.). Algorithms in this category consider packet loss as a congestion indicator, so they use reactive congestion adaptation methods each time a packet loss is detected. The most deployed CCA in this category is Cubic (configured by default in Linux since kernel-2.6.18 [65] and in Windows10 and Windows Server 2019 since 2019 [82]). Unlike Reno and NewReno which use a linear growth function after loss events, Cubic relies on a cubic function that is characterized by the aggressive growth of the congestion window after loss events. More specifically, Cubic slows down its growth rate as the cwnd size approaches the point of the last congestion event (Wmax) and after reaching this point, it probes for more bandwidth by slowly increasing the cwnd. It eventually speeds up its growth rate again as it moves away from Wmax [77].

The delay-based CCAs, unlike loss-based algorithms, rely on delay in order to detect and adapt to congestion. The most known algorithm in this category is Vegas [82]. Vegas [22] uses a proactive approach to congestion detection, i.e., it detects congestion before packet losses occur. This is done by controlling the growth of the congestion window based on the RTTs of the packets. Technically speaking, Vegas first measures the expected data rate by dividing the cwnd by the base RTT (i.e., minimum of all measured RTTs). Then, in the same way, it computes the current data rate using the current RTT. After that, it computes the difference between expected and current data rate (i.e., $Diff = Expected - Current$). Based on the obtained value, Vegas decides whether it should increase or decrease the cwnd, since the computed $Diff$ value clearly shows whether there is extra data in the network or not. To make its decision, Vegas defines two thresholds α and β, with $\alpha < \beta$ and compares the computed $Diff$ value to them. As a result, during the next RTT : it linearly increases the cwnd if $Diff < \alpha$ (i.e., too little extra data) ; decreases the cwnd if $Diff > \beta$ (i.e., too much extra data) ; and leaves the cwnd unchanged if $\alpha < Diff < \beta$ (i.e., authorized amount of extra data). In a general sense, Vegas assumes the farther away the current throughput gets from the expected throughput (or the more the current RTT is greater than the base RTT), the worse is the degree of congestion in the network [22]. However, as highlighted in [82], despite its proactive approach to congestion and its ability to avoid increased delay, Vegas faces a serious fairness issue when competing with any loss-based CCA, especially when the bottleneck buffer is deep. Basically, Vegas will always try to minimize its buffer occupancy or extra data, while a loss-based CCA will not stop increasing its sending rate until the bottleneck buffer is filled (e.g., NewReno, Cubic). And in such a situation, the flow with the highest number of packets in the bottleneck buffer has the greatest share of the bandwidth. Because of this, Vegas, and by extension, all pure delay-based CCAs suffer in terms of throughput when competing with aggressive or greedy CCAs. A recently proposed CCA, named Nimbus [50] aims to bypass this shortcoming of delay-based CCAs by using an hybrid scheme that switches between loss-based and delay-based behavior depending on whether the competing cross traffic is elastic or inelastic. To determine the elasticity of the competing cross traffic, Nimbus introduces frequent spikes in its own sending rate and then checks if these spikes affect the rate of the cross traffic, at which case it switches to a loss-based (NewReno or Cubic) behavior ; otherwise it uses a delay-based behavior (Vegas or Copa). However, as reported by the authors, this technique yields inaccurate results over

wireless links. In fact, the time-varying nature of these networks causes the elasticity detector to incorrectly classify all cross-traffic as elastic.

The rate-based CCAs, unlike delay-based and loss-based algorithms, adapt their sending rate based on the network delivery rate. These algorithms have the advantages of both loss-based CCAs (in terms of link utilization) and delay-based CCAs (in terms of delay reduction). The most deployed algorithm in this category is BBR [26, 24, 105], published by Google in 2016. BBR is referred by its authors as a Model-based congestion control algorithm, in the sense that its behavior is based on an explicit network path model. This model includes explicit estimates of two parameters (updated upon receipt of ACKs) : the estimated bottleneck bandwidth or $BBR.Bw$ (using recent delivery rate samples) and the estimated two-way round-trip delay or $BBR.min\_rtt$. With these two estimates, the main goal of BBR is to pace its packets at or near $BBR.Bw$ and then make sure the pipe remains full but not congested by maintaining a Bandwidth Delay Product (BDP) worth of data in flight (i.e., $BBR.Bw \times BBR.min\_rtt$), which result in maximum throughput with minimum delay. The most recent IETF (Internet Engineering Task Force) draft about BBR [27] defines in total four control phases for BBR operation : Startup, Drain, ProbeBW (i.e., Probe Bandwidth) and ProbeRTT. Firstly, in the Startup phase, which is similar to the traditional slow start, the sending rate is doubled at each RTT until the measured bandwidth does not increase further (which means the bottleneck pipe is full). Then in the Drain phase, the sending rate is reduced in order to drain the queue created during the startup phase. After that comes the ProbeBW phase where BBR raises the amount of data in flight to probe for more bandwidth. Technically speaking, in this phase, one out of every 8 RTTs BBR inflates its rate to $1.25 \times BBR.Bw$ then immediately lowers it to $0.75 \times BBR.Bw$ in the next RTT to drain any excess packets out of the queues. $BBR.Bw$ is set to the maximum observed packet delivery rate over the last 8 RTTs [105]. Finally, after not measuring $BBR.min\_rtt$ for a 10 seconds period, BBR enters the ProbeRTT phase, during which the amount of data in flight is reduced to 4 MSS to drain any possible queue and get a real estimation of the minimum RTT. It is also worth noting that BBR reaction to packet loss is different from loss-based CCAs. Actually, simple packets losses are not considered as a congestion signal but as a change in the path parameters. For this reason, when packet losses occur and there are still packets in flight, BBR reduces the cwnd to match the current delivery rate and then make sure its sending rate during Loss Recovery never exceeds twice the current delivery rate. BBR considers a congestion as heavy only when it assumes all in flight packets are lost (i.e. upon an RTO expiration), in which case the cwnd is set to 1 MSS. Another important point worth noting is that Google has already conducted several experiments with BBR, and reported in [26] a 133 times throughput improvement as well as a 53 percent reduction in the median RTT when compared to Cubic in a shallow buffered network.

However, despite BBR's efforts, its performance in terms of both throughput and delay gets worse when competing with a loss-based CCA in a deep buffered environment. Based on BBR's way of operation and as demonstrated by the authors in [102], it is clear that BBR will always be affected by the overshooting of a concurrent loss-based CCA in deep buffered networks. As a result, BBR will eventually end up with less throughput than the concurrent loss-based CCA as well as with a significant increase in delay. Similarly, pure delay-based CCAs are also affected by the same issue. As a matter of fact, as long as loss-based CCAs dominate the Internet traffic [82] and everyone is free to use their own customized CCA, it is virtually impossible to solve the excessive delay increase (or bufferbloat) in deep buffered bottlenecks by just introducing a new CCA since it will be affected by the negative effects of the already-deployed CCAs. Indeed, this bufferbloat issue tends to be less obvious and less prevalent with wired broadband access networks.

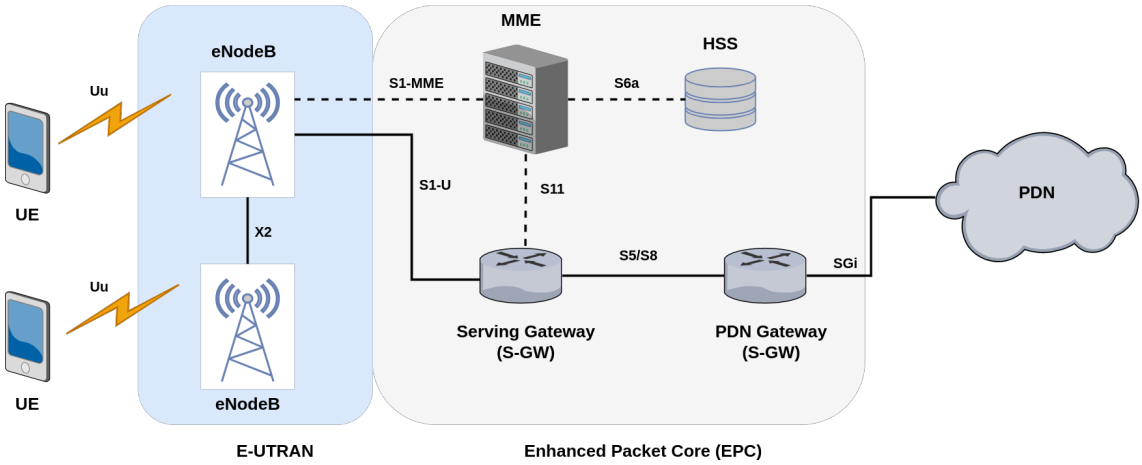


Figure 2.3 – LTE (4G) simplified network architecture

This is mainly due to the fact that the user's home network is rarely the bottleneck [103] and also because Internet Service Providers (ISPs) do not generally provision their networks with oversized buffers [103]. However, it is important to note that the same bufferbloat issue becomes more obvious and more prevalent with 4G/5G access networks, because they differ from their wired broadband counterparts in various ways as we will see in the next section, while analyzing the behaviors of TCP in case of cellular access networks.

2.2 4G/5G stack and related transport layer issues

Mobility and ubiquitous connectivity are among others some of the unique services that have allowed cellular networks to revolutionize the way we communicate. In order to provide these services with adequate levels of QoS and allow a universal interoperability between deployed networks or PLMNs (Public Land Mobile Networks), the 3GPP has defined a set of specifications for every generation of cellular network, from the second to the fifth generation. Although each generation comes with its unique characteristics, they all follow the same basic architecture defined since the GSM (Global System for Mobile Communications) specification, i.e., a Radio Access Network (RAN) connected to a Core Network (CN). Nevertheless, for the sake of this thesis, we will mainly focus on 4G and 5G networks, particularly from an end-to-end data traffic management perspective.

In its release 8, the 3GPP defines an architecture for 4G/LTE (Long Term Evolution) networks, which is composed of a radio access network referred by the specification as Evolved UMTS (Universal Mobile Telecommunications System) Terrestrial Radio Access Network (E-UTRAN) and a core network referred as Evolved Packet Core (EPC). In this architecture, the data traffic of each user is conveyed between the different components of the network through separate bearers, which are logical connections/tunnels (with some QoS attributes) between network entities. And since these entities are distributed in the LTE network as illustrated in Figure 2.3, it is necessary to do a one-to-one mapping between the bearers installed at different part of the network (as shown in Figure 2.4) so that each user is provided with an independent end-to-end logical tunnel, also called EPS (Evolved Packet Service) bearer, that guarantees a complete data traffic isolation throughout the whole LTE network (i.e., both access and core networks). In 3GPP

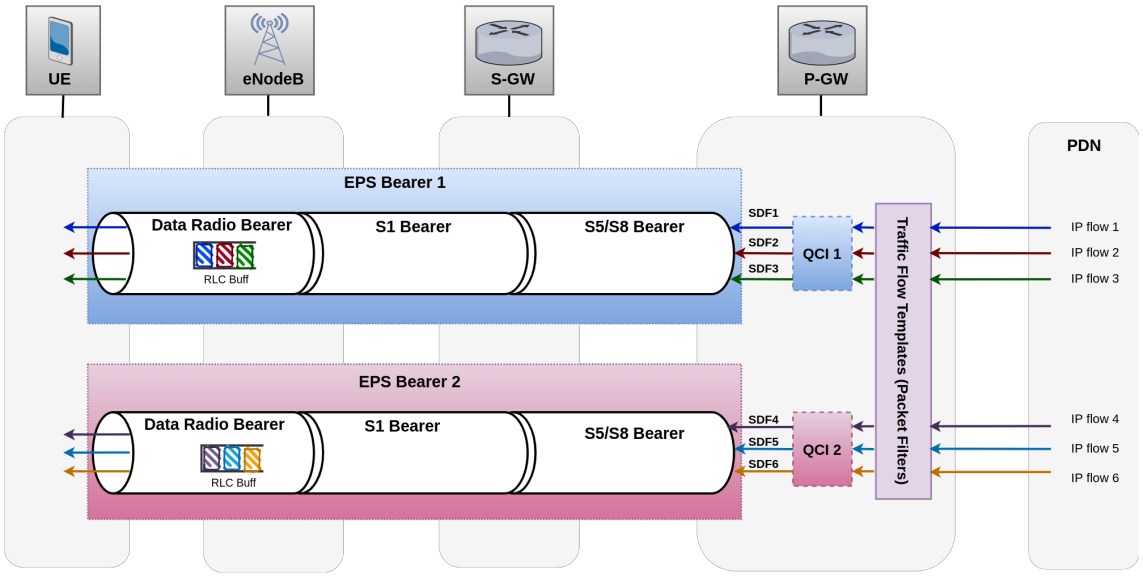


Figure 2.4 – LTE/4G end-to-end data flow

release 9, the network entity that is responsible for establishing and configuring the EPS bearers is called the Mobility Management Entity (MME) and it is located in the core network. More specifically, in the LTE architecture, the core network (EPC) consists of the following main components :

- The Packet Data Network Gateway (P-GW), which as its name suggests is the point of contact with the outside world. It is the component that gives IP address to the users and also the one that applies traffic engineering techniques to the user's data traffic. As such it serves as the mobility anchor for IP services. The P-GW generally communicates with multiple Packet Data Networks (PDNs), such as the network operator's IP Multimedia Subsystem (IMS) or the Internet and each PDN is identified by an access point name (APN).
- The Serving Gateway (S-GW), acts as a local router that forwards packets between the P-GW and the radio access network. Compared to the P-GW, the S-GW can be considered as a local mobility anchor point for the RAN. Therefore, the UE can be assigned a new S-GW if it moves sufficiently far while keeping the same P-GW (i.e., a tunnel is just established between the P-GW and the new S-GW).
- The Mobility Management Entity (MME), handles all the signaling exchanges between the RAN and the core network and between users and the core network. Even though it is not involved in the forwarding of the user's data traffic unlike the P-GW, the S-GW or the base station, it is important to note that the MME is the component that instructs them so that they can establish/configure the S5/S8 bearer (between the S-GW and the P-GW), the S1 bearer (between the S-GW and the base station) and the Radio bearer, which together form the full EPS bearer (between the UE and the P-GW). A typical PLMN is generally composed of several MMEs, each of which is responsible for a given geographical region, but at a given time a UE can be assigned to only a single MME, known as its serving MME. Similarly to the S-GW, the serving MME can also change if the UE moves sufficiently far.

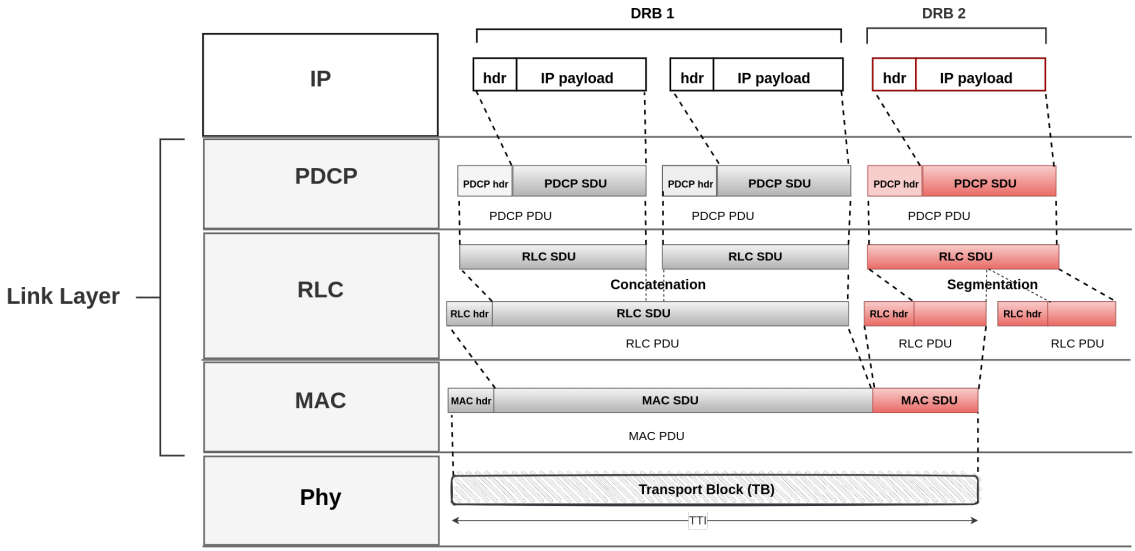


Figure 2.5 – LTE protocol stack in the air interface

- The Home Subscriber Server (HSS), is the subscriber database. It stores among others, the subscriber's unique identifiers (e.g., IMSI, MSISDN etc.) authentication information, the APNs the subscriber is allowed to use, the maximum throughput allowed for these APNs etc. The HSS is accessed by the MME using the DIAMETER protocol.

Regarding the Radio Access Network or EUTRAN, it is composed of base stations named evolved NodeB or eNodeB (eNB), which are the most complex devices in the LTE network. The eNodeB provides radio access to the users via the air interface (referred to as Uu interface in LTE) and handles the allocation of its scarce radio resources as well as the scheduling of the connected users. It consists of antennas that emit radio waves ; radio modules (also called Remote Radio Units or Remote Radio Heads : RRU/RRH) that perform the Digital-to-Analog-Conversion (DAC) and Analog-to-Digital-Conversion (ADC) on the transmitted and received signals over the air interface ; digital modules (or Baseband Units : BBU) that handle physical layer and link layer processing (e.g., modulation/demodulation, Forward Error Correction, PDU Segmentation/Concatenation/Multiplexing etc.). Additionally, the BBU also acts as a bridge to the core network by receiving/sending packets over the high-speed backhaul network [33]. Besides these particularities, it is important to note that, unlike the core network that relies on Ethernet at the physical and link layers, the interface between the UE and the eNodeB (or air interface) relies on completely different protocols/mechanisms at the physical and link layers in order to overcome the impairments of the radio environment. Figure 2.5 shows the protocol stack used in the air interface and highlights how data packets are handled from the IP to the physical layer. At the physical layer (Phy), as briefly mentioned in Chapter 1, the technique used on the air interface for radio transmission/reception between the eNodeB and the UE is called Orthogonal Frequency Division Multiple Access (OFDMA). This technique not only enables the base station to simultaneously communicate with multiple users (just like the other multiple access techniques), but also significantly reduces the Inter Symbol Interference (ISI). In fact, instead of sending the user's data at a high speed over a single carrier, OFDMA divides the data into several parallel sub-streams that are simultaneously

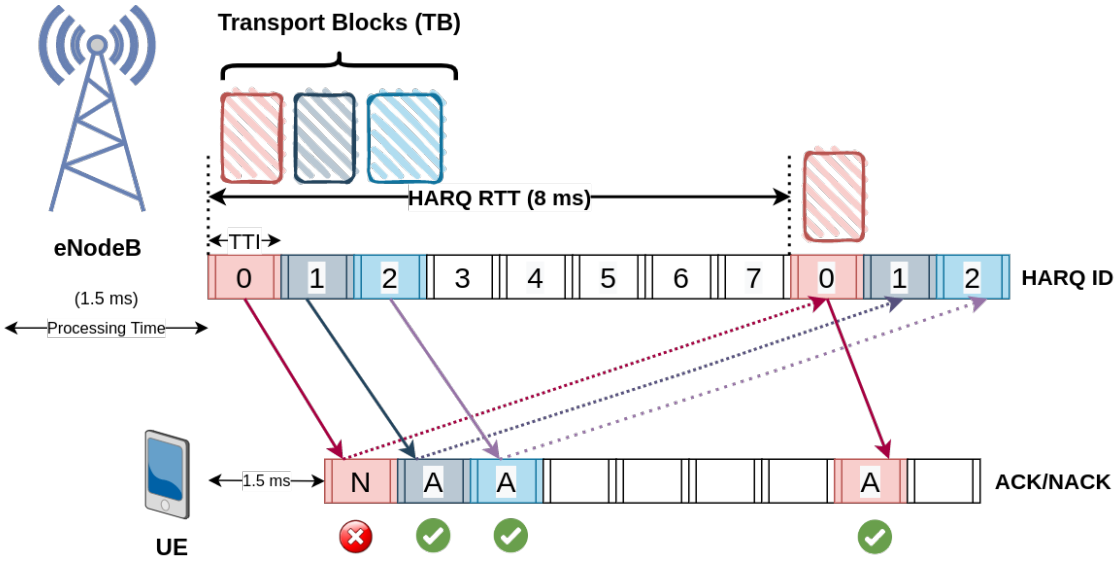


Figure 2.6 – Hybrid ARQ RTT in LTE-FDD

sent over many sub-carriers. That way, the total data rate remains the same while the symbol duration gets longer on each sub-carrier. Hence, successive symbols on different rays of the same sub-carrier (i.e., direct and longer reflected rays) have less parts that overlap at the receiver. The degree of ISI and error reduction directly depends on the number of defined sub-carriers. LTE uses a fixed sub-carrier spacing (SCS) of 15 kHz, which allows up to 1200 sub-carriers for a maximum 20 MHz LTE bandwidth (specified in Release 9). For data transmission, the eNodeB allocates the available sub-carriers to users at every Transmission Time Interval (TTI), which corresponds to 1 ms in LTE. To be more specific, the resource allocation is based on a time and frequency grid in which radio resources are decomposed into groups of sub-carriers in the frequency axis and time-slots in the time axis. Basically, in typical type 1 LTE frame structure (i.e. Frequency Division Duplex), a grouping of 12 sub-carriers (i.e., 180 kHz) during one slot (0.5ms) is called a Physical Resource Block (PRB). Since the symbol duration is 66.67 μs ($T = 1/15kHz$), each of the 12 sub-carriers can carry 7 symbols during one slot, which makes 84 symbols or Resource Elements (REs) per PRB. The PRB is the smallest unit of resources that can be allocated to a UE and is always allocated for 1ms, which corresponds to the duration of one TTI or one subframe (i.e., 2 consecutive 0.5ms slots). For a given user, the data block transmitted each TTI at the physical layer is called a Transport Block (TB), and its size, or TBS (Transport Block Size), during a given TTI depends on the number of PRBs allocated to the user during that TTI, and on the selected modulation and coding scheme (e.g.,BPSK, 16QAM, etc.), which are both handled at the link layer. As illustrated in Figure 2.5, the link layer consists of 3 sub-layers, namely, the Medium Access Control (MAC), the Radio Link Control (RLC) and the Packet Data Convergence Protocol (PDCP).

The MAC sub-layer controls at a high level some operations performed at the physical layer such as channel coding (for error correction), HARQ retransmissions, or Adaptive Modulation and Coding (AMC). It also handles among others, PRB allocation, user scheduling, RLC PDUs (Protocol Data Unit) multiplexing and prioritisation. In fact, the MAC Service Data Unit (SDU) or payload is composed of one or several RLC PDUs, grabbed by the MAC scheduler from the

user's RLC entities based on their priorities. Not only does the MAC scheduler decide which data to put into the MAC payload of a given user, but it also decides, based on some available metrics (e.g., users channel Quality Indicators, traffic load, QoS of radio bearers, etc.) which users to schedule for the next TTI, as well as the number of PRBs they are going to be assigned. As such the MAC scheduler is a complex piece of software that can use any type of scheduling algorithm (e.g. Round Robin, Proportional Fair, etc.), but for now, its implementation is left to the eNodeB manufacturer. However, regardless of the eNodeB manufacturer, all implementations of the MAC sub-layer control physical layer transmissions based on the same HARQ mechanism.
HARQ is a retransmission technique that operates in a stop-and-wait manner, meaning that the sender has to stop and wait for a feedback from the receiver (i.e., ACK or NACK) for each individual data block (i.e., Transport Block) before proceeding with the next transmission or retransmitting the previously sent data block. To avoid this blocking nature of the stop-and-wait technique, a certain number of HARQ processes are used in parallel so that the eNodeB can directly send a window of Transport Blocks to the user without waiting for any feedback. Considering the fixed node processing delays of 1.5ms [7] (which includes radio frame alignment) at both the UE and the eNodeB, the 1ms TTI duration, and the FDD frame structure, in which the default number of HARQ processes is fixed to 8 [7], this results into an HARQ RTT of 8ms in case of no retransmissions and $8 + n * 8$ in case of n retransmissions [7]. Figure 2.6 shows how the HARQ mechanism is used in a typical LTE-FDD configuration.
Despite this overhead in term of delay, HARQ retransmissions can be very effective in case of light BLER (Block Error Rate), hiding random losses from TCP, thus preventing the TCP sender from unnecessary decreasing its sending rate (e.g., when the TCP sender is using a loss-based CCA). However, in case of a temporary high BLER (resulting from bad radio conditions), the HARQ retransmissions might not be enough to recover the corrupted Transport Blocks, in which case, after a fixed number of unsuccessful retransmissions, the received data (i.e., with a certain amount of missing blocks) can be directly delivered to the upper layer depending on whether the associated RLC entity is using a reliable or unreliable delivery mode. Therefore, in the latter case, also called RLC Unacknowledge Mode (RLC UM), the TCP receiver sends duplicate ACKs for the missing segments. As a result, the TCP sender observes a delay increase (due to the HARQ retransmissions) followed by a packet loss (due to missing blocks) which generally result into a misinterpreted congestion signal regardless of the category of the CCA in use. Technically speaking, a loss-based CCA would directly halve its congestion window since it considers any packet loss as a congestion signal. Similarly, delay-based and rate-based CCAs would also reduce their sending rate to match the reduced delivery rate (due to retransmission delay) since they consider delay increase as strong indication of congestion. Figure 2.7 illustrates this phenomenon for different categories of CCA. It is important to note that these misinterpreted congestion signals (due to random losses) are observed only when the RLC entity uses Unacknowledged delivery Mode or RLC UM. In case of Acknowledged delivery Mode (or RLC AM) different phenomenons are observed due to the operations carried out at the RLC sub-layer.

The RLC sub-layer is where the user's data blocks (from the same or different radio bearers) are segmented/concatenated according to the size requested by the MAC sub-layer (to fill the TB). It is also responsible for mapping the user's radio bearers to RLC entities, each of which is associated with its own separate buffer. In other words, not only is the traffic of each user separated, but flows belonging to the same user can also use different buffers depending on whether they share the same data radio bearer or not. In fact there is a one-to-one mapping between a Data Radio bearer and an RLC entity, which, in case of AM operating mode (Acknowledge Mode)

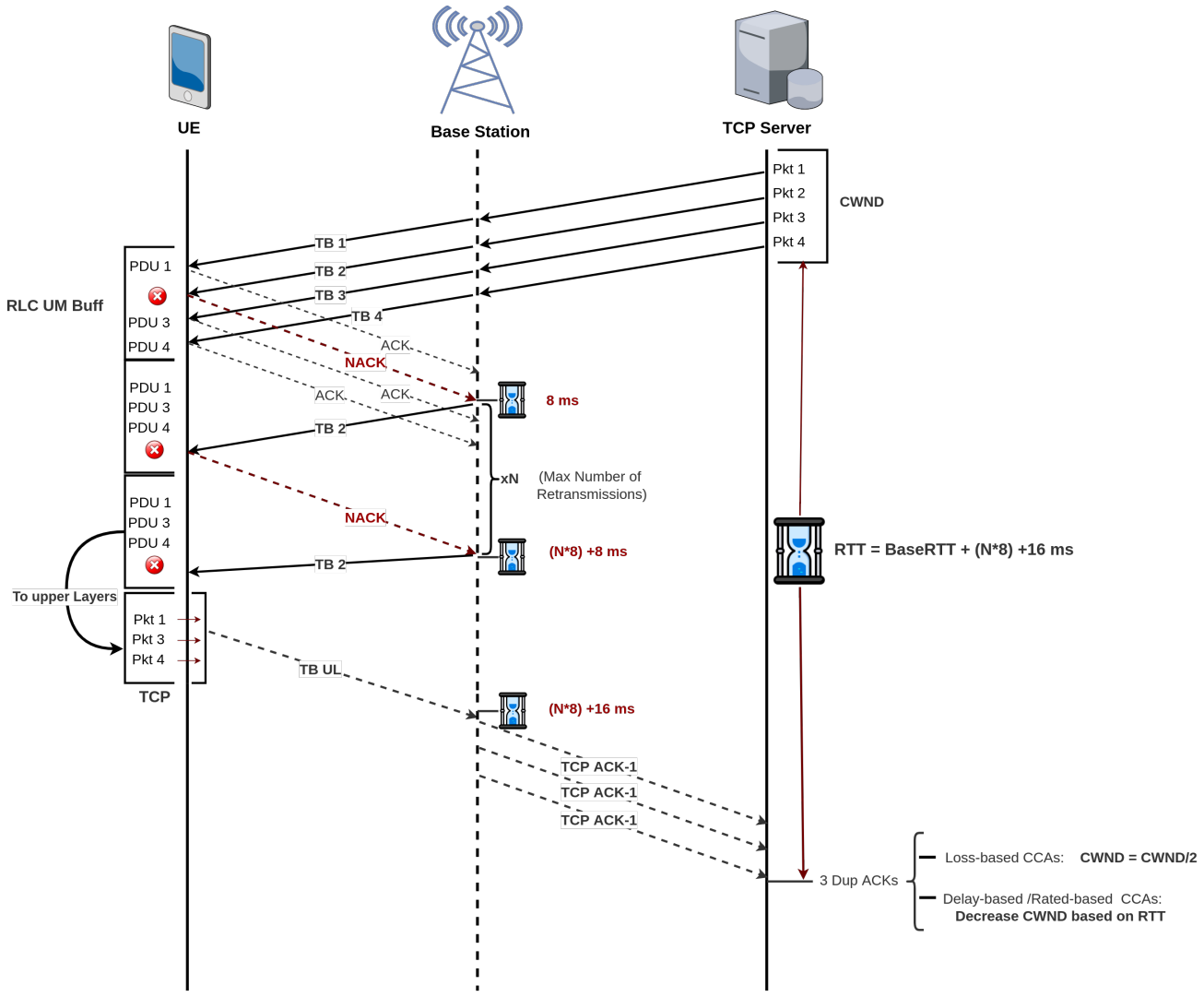


Figure 2.7 – Failed HARQ retransmissions impact on CCAs

provides a TCP-like reliable RLC PDU delivery. Unlike RLC UM, which relies solely on HARQ retransmissions at the MAC sub-layer, RLC AM uses an additional retransmission technique at the RLC sub-layer in addition to the HARQ used at the MAC sub-layer. This technique, called ARQ (Automatic Repeat request) aims to recover residual HARQ errors (i.e., missing blocks due to HARQ failures). ARQ is a window-based selective retransmission technique in which, the receiver's RLC sub-layer, fires a reordering timer each time a gap in the sequence of received PDUs is detected. Upon expiration of this timer, a status message containing the sequence numbers of the missing PDUs is sent to the sender's RLC sub-layer, which instructs the MAC sub-layer to resend the detected missing PDUs again via HARQ. That way, random losses are concealed even in case of high BLER since missing blocks from HARQ failures are always retransmitted.
However, although ARQ ensures a persistent reliability mechanism that completely hides random losses from TCP (unlike HARQ in RLC UM), it can also introduce severe bufferbloat and throughput collapse. In fact, since the RLC buffers are deep, too many retransmissions result in sporadic delay spikes and excessive buffering. As a result, a TCP RTO can expire before the corrupted PDUs are successfully retransmitted, thus causing a severe TCP throughput collapse or an undesired slow start as termed in [59] (since cwnd is set to 1MSS, and TCP reenters slow start). Also, the sporadic spikes in the end-to-end delay prevent rate-based and delay-based CCAs from correctly estimating the available capacity, and because of that, they generally end up underestimating or overestimating the available bandwidth since they adapt their sending rates based on the variations in the delivery rate and/or the variations in the end-to-end delay which are both affected by HARQ and ARQ retransmissions delays. This phenomenon has already been observed and studied by authors in [32] while evaluating BBR performance over LTE and millimeter waves, but their proposed BBR-S solution, despite being better than BBR in term of delay, introduces a non-negligible level of buffering.

Above the MAC and RLC sub-layers is located the PDCP sub-layer, which does not have a noticeable impact on TCP performance. In fact, the PDCP sub-layer is responsible for ciphering and optionally compressing (through RoHC : Robust Header Compression) IP packets and signaling messages. Additionally, it allows lossless handover by making sure the user's buffered data is properly transferred to the new eNodeB.

The main LTE network components and link layer mechanisms being individually described, we can now consider how a user's data flow is handled from a QoS perspective from the P-GW all the way to the UE. As illustrated in Figure 2.4, a given user can have multiple EPS bearers, each of which can convey several service data flows (SDFs). These SDFs are detected and classified thanks to Traffic Flow Templates (TFT) which consist of specific packet filters based on 5-tuple (i.e., IP source/destination addresses, source/destination ports and protocol) and each SDF is associated to a QoS Class Identifier (QCI) which indicates the level of QoS (in terms of delay, loss rate, etc.) this SDF can receive. Also, it is important to note that there is a one-to-one mapping between a QCI and an EPS bearer, so every SDF having the same QCI shares the same EPS bearer. Similarly, on the eNodeB side, there is also a one-to-one mapping between an EPS bearer and a Data Radio Bearer. Therefore, SDFs sharing the same EPS bearer will also share the same RLC buffer. For instance, some TCP-based services such as web, email or FTP have the same QCI value (as indicated in 3GPP TS 23.203), which means that they share by default the same EPS bearer, so the same RLC buffer. In fact, even if each SDF had a different QCI value, some of them would inevitably end up sharing the same DRBs, for the simple reason that the maximum number of DRBs per user is limited to 11 (as specified by 3GPP TS 36.331 - V15.3).
Indeed, this end-to-end traffic management method in 4G/LTE has the advantage of completely

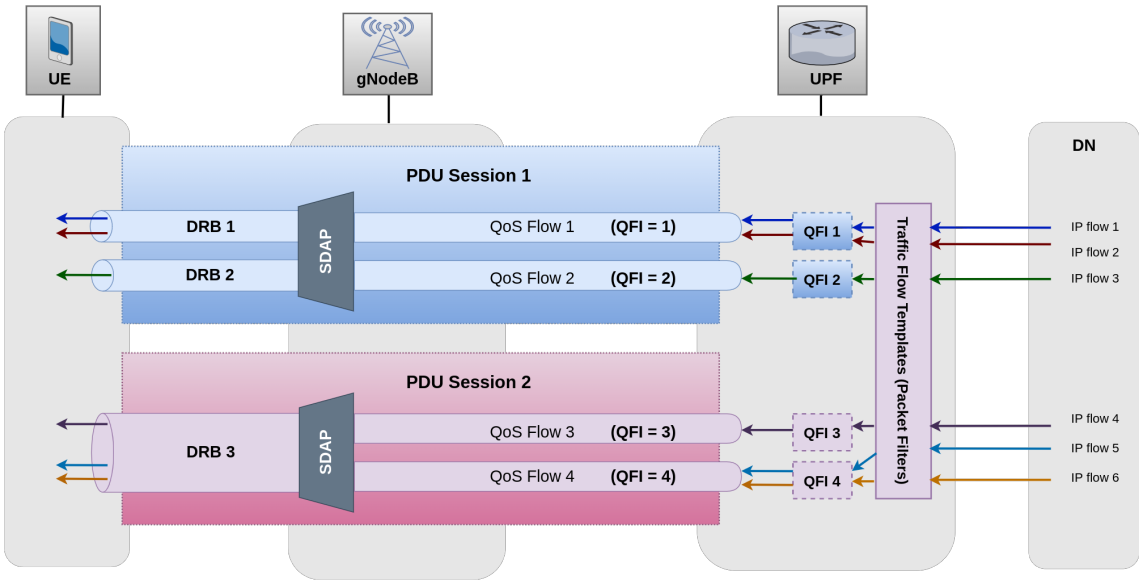


Figure 2.8 – 5G end-to-end data flow

isolating traffic from different users; however, it fails to isolate SDFs belonging to the same user. As a result, in addition to the previously mentioned TCP issues, a bufferbloat or a fairness issue also arises whenever a sheer-download SDF using an aggressive CCA (e.g., loss-based CCA) progresses alongside an interactive or delay-sensitive SDF (e.g web browsing or online gaming). To alleviate this self-inflicted bufferbloat issue (called self-inflicted because it is introduced by the user itself) and reduce end-to-end delay in general, the new 5G stack comes up with a different traffic management model as well as an additional sub-layer on top of PDCP. However, as will be seen below, these new measures are limited and fail to solve the aforementioned issues.

Actually, the 5G stack is quite similar to the 4G stack in terms of network components. For instance, the User Plane Function (UPF) handles the functions of the S-GW and the P-GW. The MME function has been split between the AMF (Access and Mobility Management Function), which handles connection and mobility management tasks, and the SMF (Session Management Function) which is responsible among others, for the establishment of PDU sessions (the equivalent of 4G's EPS bearers), the management of UE's IP address, the enforcement of QoS and other policies. The HSS has been replaced by the UDM (Unified Data Management). In addition to these basic functions, the 5G architecture also contains some few functions that have no equivalence in 4G, such as the NSSF (Network Slice Selection Function), which enables the selection of different network slices for the requested services, or the NRF (Network Repository Function) which holds an updated repository of all the Network Functions (NFs) available in the operator's network together with the services they support. However, when it comes to bufferbloat mitigation or delay reduction in general, the main innovation of 5G lies in the way it manages data traffic throughout its core and radio access network. In 5G, the radio access network is based on 5G New Radio (5G NR) which basically uses the same OFDMA technique as 4G EUTRAN but supports much larger operating bandwidths (up to 400MHz in mmWave band or FR2). 5G NR also allows flexible Numerologies (μ) for sub-carrier spacing (SCS). Basically, the SCS is determined by $15 \times 2^{\mu}$ with μ value ranging from 0 to 3, which allows sub-carrier spacings of 15kHz, 30kHz, 60kHz

and 120kHz. By allowing such variable SCS scheme, 5G NR enables a significant reduction in air latency, since larger SCS means shorter symbol duration, in other word shorter transmission time intervals (e.g., a SCS of 120kHz allows TTIs of 125μs). In addition to its flexible numerology scheme, 5G NR also introduces a new sub-layer above PDCP, named Service Data Adaptation Protocol (SDAP) which is implemented by both the UE and the gNodeB (the base station in 5G RAN). The main function of this new SDAP sub-layer is to map QoS Flows (consisting of one or several SDFs) to DRBs based on their QoS Flow Identifiers (QFIs). QFIs are identifiers used to mark and classify SDFs, just like QCI in 4G. However, unlike in 4G where there is a one-to-one mapping between a QCI and an EPS bearer or where all the SDFs sharing the same EPS bearer receive the same treatment (since they have the same QCI), in 5G, multiple SDFs sharing the same PDU session can have different QFIs and be treated differently. In other word, the QoS and traffic management approach in 5G is flow-based, meaning that QoS is enforced at flow level and not at EPS bearer level like in 4G. Furthermore, by using the new SDAP sub-layer to map QFIs to different DRBs, this flow-based traffic management principle is enforced across all the network, from the UE all the way to the UPF as illustrated in Figure 2.8. As a result, SDFs belonging to the same user, even if they share same PDU session, can be associated to different RLC buffers depending on their QFIs. This can mitigate to some degree the bufferbloat effect when the user is maintaining at the same time a sheer-download flow and a delay-sensitive or interactive flow.

However, even though the specification allows up to 64 simultaneous QFIs per user, a given UE can maintain only 8 DRBs at the same time. In other words, some QFIs will be mapped to the same DRBs, and end up sharing the same RLC buffers. Furthermore, since QFI marking is based on 5-tuple, it does not consider the behavior of the flow or its CCA. As a result, some TCP flows that are sometimes aggressive and sometimes interactive can be wrongly mapped to the same DRB as a delay-sensitive or a sheer-download flow. An example of applications that use such TCP flows is the Remote Desktop, in which opening a big file generates an aggressive download while moving the cursor around is rather delay-sensitive, so would be affected by bufferbloat. As a matter of fact, given that the behavior and requirements of the same TCP connection can change depending on the user's interaction, it is very challenging to avoid bufferbloat as long as the buffer is shared by more than one TCP flows.

2.3 Proposed solutions

Owing to their prevalence, TCP performance issues in wireless and cellular networks have been extensively studied in the literature. Although various solutions and mitigation measures have been proposed over the years, most of them fall in the following three categories : Random Loss Discrimination Algorithms, In-network Bufferbloat mitigation solutions and Cross-layer Congestion Control Algorithms.

2.3.1 Random Loss Discrimination Algorithms

As mentioned in Section 2.2, most of today's download data flows still rely on loss-based congestion control algorithms (e.g., Cubic, NewReno) which consider any packet loss as a congestion signal. Although such a technique is appropriate when losses are caused by congestion, it degrades performance in case of random errors caused by lossy wireless links. Therefore, in order to avoid unnecessary rate throttling, Loss Discrimination Algorithms (LDAs) have been proposed

to help transport layer protocols (especially TCP) to distinguish (random) wireless losses from congestion losses in wireless networks.

Samaraweera suggested NCPLD [101], a non-congestion packet loss detection scheme that implicitly identifies the type of packet loss using the variation of delay experienced by TCP packets.

TCP Veno [46, 91], proposed by Fu et al. estimates the number N of excess packets in the bottleneck buffer. In case of packet loss, Veno declares random loss if $N < 3$.

LDA_EQ [91], similar to TCP Veno, estimates queue usage using information available to TCP. In case of packet loss, congestion is declared when the estimated queue usage is larger than a certain threshold.

Ben-Jye Chang and Yi-Hsuan Li proposed a cross-layer-based adaptive TCP algorithm for 4G networks [29]. Their approach detects bottleneck location by analyzing client-side cross-layer radio information included in TCP ACKs and discriminates losses based on bottleneck buffer occupancy estimated via delay variation measurements. However, their solution does not show significant performance gain under low loss rates (i.e., $\leq$ 1%), which is the most frequent case in commercial LTE networks [30].

2.3.2 Cross-layer Congestion Control Algorithms

Cross-layer Congestion Control Algorithms, as they name suggest, rely on cross-layer radio information to adapt their sending rate to the available radio bandwidth. This is done by making the user's device report the relevant radio information via TCP acknowledgments or via a dedicated control channel. Unlike traditional end-to-end CCAs, which introduce excessive buffering or fail to appropriately discover the available RAN bandwidth (due to frequent capacity variations, delay spikes, RLC/MAC layer retransmissions etc.), Cross-layer CCAs have the advantage of providing a better balance between link utilization and unnecessary buffering (or delay increase). Some known algorithms in this category are :

PBE-CC (Physical-Layer Bandwidth measurements, taken at the mobile Endpoint) [107], which is a cross-layer CCA that adapts its sending rate based on the expected fair-share PRBs the user can get from the base station. The expected number of PRBs (i.e., achievable bandwidth) is computed on the user side by the PBE-CC mobile client, which decodes the cellular physical control channel and takes into account all the resource block allocations from the cells the UE is connected to in order to detect and consider idle PRBs. This information is then included in TCP ACKs and sent to the PBE-CC sender which adapts its sending rate accordingly. Also, upon detecting that the bottleneck has shifted from the cellular access link (i.e., certain increase in one-way delay), the PBE-CC sender falls back to a BBR-like mechanism to handle congestion in the new bottleneck (e.g., on the Internet link).

CDBE [110, 54] (Client Driven Bandwidth Estimation), which allows a client side cellular bandwidth estimation without directly invoking information from the UE MAC or physical layer unlike most cross-layer CCAs. The Bandwidth is estimated on client side by recording the number of packets received over specific time windows and then sent to the server. To avoid wrong estimations, CDBE relies on a two-window based bandwidth estimation method in which it combines several short-time window estimations over a longer window equivalent to one RTT. On the sender side, the reported bandwidth and one-way delay are used to compute the pacing interval and the congestion window. Based on the value of the one-way delay, the sender can switch between

four possible operating states (STARTUP, GROW, DRAIN and NORMAL), each of which uses specific gains in order to calculate the congestion window and the pacing interval.

CQIC [78], which controls its sending rate based on the channel quality indicator (CQI) and discontinuous transmission ratio (DTX) reported by the user. The CQIC receiver continuously predicts the available data rate and reports it to the sender. To achieve this, the receiver collects CQI information every 2ms in order to perform a CQI-to-rate mapping. the estimated data-rate is then multiplied by the DTX ratio so that the base station scheduling pattern is also taken into account. On the sender side, packets are spaced out according the reported data rate. The impact of data rate overestimation is alleviated by imposing a two times BDP limit on the number of bytes in flight. The original CQIC algorithm was designed based on HSPA+ (High Speed Packet data Access) and implemented for google's QUIC protocol [54]. It has recently been adapted to LTE for TCP by the authors in [111].

X-TCP [54], which is a cross-layer CCA designed for uplink (UL) traffic over 5G mmWave networks estimates the available UL bandwidth using assignment information contained in the Downlink Control Information (DCI). Upon the reception of each ACK, the X-TCP sender in the mobile increases/decreases its congestion window based on the estimated bandwidth, RTT and signal quality. Congestion is declared upon a certain increase in RTT (i.e., above a certain threshold beyond the minimum RTT) or in case of poor signal quality (SINR).

ABRWA [54] changes the receive window value in the mobile based on cross-layer information. It estimates the bandwidth of the radio channel from the signal quality and multiplies the estimated value by the minimum RTT. The resulting value is used to replace the receive window value in the ACKs. Later, Dynamic Receive Window (DRW) [54] was proposed as an enhancement to ABRWA. Unlike ARBWA, it uses information in DCI at the UE to estimate the available bandwidth when it detects a certain increase in the RTT.

2.3.3 In-network Bufferbloat mitigation solutions

We categorize as In-network bufferbloat mitigation solutions, any solution for bufferbloat or delay reduction that is deployed in the network between the user and the end-server. These solutions are either integrated into some existing entities in the network (e.g., eNodeB, P-GW, UPF, etc.) or deployed as stand-alone middle-boxes :

Authors in [94] proposed milliProxy, a performance enhancing proxy for 5G mmWave scenarios that transparently regulates the Receive Window of the TCP flows using a variable Flow Window (FW). The FW is computed using the estimated RAN data rate (derived from channel quality information), the RLC buffer occupancy and the estimated RTT.

In-band throughput guidance (TG) was introduced in [63]. This approach allows a functional element (TG provider) residing in the RAN to include RAN throughput information in the TCP header. This throughput information can then be exploited by a TCP server to regulate its sending rate.

LCTCP proposed in [17] for 5G networks relies on a out-of-band signaling of queue occupancy information between the base station and the end server. The transmitted information is used by the server to adjust its sending rate so that the amount of packets queued at the base station is always aligned with the throughput and delay requirements of the application. However, although LCTCP outperforms conventional CCAs in some scenarios, it has been shown that it cannot deliver both high throughput and low delay simultaneously.

The authors in [76] proposed a TCP proxy for multi-connectivity enabled 5G mmWave network that speeds up congestion window growth and prevents RLC buffer overflow. The former is achieved by separating congestion events of the radio from wired segments while the latter relies on receive window modifications based on an estimation of the available proxy buffer space.

NATCP (Network Assisted TCP), proposed in [10] addresses both the link utilization problem in cellular networks and the intra-fairness issue that arises when multiple flows shares the same per-user buffer. At a high level, NATCP operates as a cross-layer CCA. It relies on a sender side module that leverages explicit radio bandwidth and delay information from a functional entity named NetAssist (Network Assistance), deployed at the edge and working in conjunction with the base station. Additionally, it also exploits fairness information, reported by the user, which consists of the number of concurrent flows running in the user's device. After receiving the required information from both the UE and NetAssist, the NATCP sender matches its pacing rate to the estimated data rate reported and sets its congestion window to the estimated BDP divided by the number of connections (reported by the UE). The obtained cwnd value is scaled by a scaling factor (fixed to 2) in order to compensate for the difference between the cwnd and the real number of bytes in flight.

A solution based on the use of Active Queue Management (AQM) at the SDAP sub-layer in conjunction with reduced buffer sizes at the RLC sub-layer was proposed in [60]. The authors enhance the SDAP sub-layer, whose original and only function is to perform the mapping between QFIs and DRBs, from a simple mapper to a scheduler with the ability to manage multiple QFI queues. They conducted extensive experiments, using CoDel at different sub-layer of the 5G stack (SDAP and RLC) while restricting the DRB buffer sizes at the RLC sub-layer. Through their experiments, which consist of running a greedy TCP flow using Cubic alongside a delay-sensitive ICMP flow in various scenarios, they demonstrate that the use of CoDel for managing the QFI queues (i.e., at the SDAP sub-layer) in combination with restricted DRB queues (slightly above the maximum possible egress rate of the MAC scheduler) can reduce up to 4 times the delay of the delay-sensitive flow, while maintaining a throughput close to the maximum achievable for the greedy TCP flow.

DRLQL, 5G-BDP and USP were proposed in [61]. Similarly to the previous solution, it is worth noting that the last two techniques here, i.e., 5G-BDP and USP are also based on the assumption that the SDAP sub-layer can manage QFI queues. DRQL (Dynamic RLC Queue Limit) dynamically resizes the RLC buffer size based on the MAC sub-layer egress capacity. Basically, each time the MAC sub-layer dequeues data from the RLC queue, the amount of remaining bytes is recorded. If the recorded value is not null, the RLC buffer size is reduced by this value in the next TTI. However if the value is null and the buffer limit is reached, which may indicate starvation at the MAC sub-layer, then the buffer size is immediately increased.

5G-BDP, which aims to work at the Kleinrock operating point (i.e., maintaining a BDP worth of data), uses CQI, radio capacity information (i.e., last PRB) and RLC buffer state to pace data transfers between the SDAP and the RLC sub-layers. To achieve this, every TTI, the optimal number of bytes to be forwarded to the MAC sub-layer is estimated by computing the difference between the MAC sub-layer's current egress size and the remaining bytes in the RLC buffer. Then the SDAP sub-layer periodically determines how much data could have been sent to the RLC sub-layer since last TTI based on the elapsed time and on the amount of bytes transmitted during this period. If the estimated value is inferior to the transmitted bytes, the SDAP sub-layer forwards an amount of bytes equivalent to the difference of the two values, otherwise no data is forwarded from

the SDAP sub-layer. This mechanism allows the SDAP sub-layer to correctly pace data without overflowing the RLC buffer.

USP (UPF-SDAP Pacer), designed for scenarios where a TTI-level communication (i.e., around $1ms$) is possible between the UPF and the gNodeB, paces packets from the UPF to the SDAP sub-layer at a rate that maintains the SDAP buffer always occupied while avoiding excessive buffering. Basically, USP computes the optimal SDAP occupancy and this value is used by the UPF every time it sends data toward the access network. With the optimal occupancy value, the UPF decides how much data to send so that current occupancy can reach the optimal value. USP doubles the optimal value every half of TTI (i.e., 0.5ms for a SCS of 15kHz), thanks to which it can maintain a more aggressive pacing than 5G-BDP and avoid starving the SDAP queues. Also, it is important to note here that the SDAP sub-layer stops data transfers toward the RLC sub-layer each time it estimates the RLC buffer is in a congested state, and resumes the transfers once the congestion is gone.

2.4 Summary

In this Chapter, we have revisited TCP philosophies and congestion control algorithms after several decades of evolution. The main TCP built-in mechanisms and options have been presented so that TCP issues in 4G/5G networks can be correctly understood. We have briefly described 4G and 5G stacks from an architectural point of view and focused more on the techniques they use to handle user data traffic in the radio access and core networks. The main TCP issues caused by these techniques have been extensively discussed and analyzed. In particular, issues that arise from excessive buffering due to deep buffer sizes and link layer retransmissions have been thoroughly studied. We have also shown that 5G's built-in delay reduction techniques and new per-flow QoS enforcement approaches are limited against these issues. Lastly, mitigation solutions that exist today in the literature have been categorized (i.e., Random Loss Discrimination Algorithms, Cross-layer CCAs, In-network bufferbloat mitigation solutions) and presented.

CHAPTER 3

Multi-Access Edge Computing from a Transport Layer perspective

In this chapter we first present the Multi-Access Edge Computing framework proposed by the European Telecommunications Standards Institute (ETSI), highlight the main components and interfaces of its architecture, and show why it is necessary for unlocking the full potential of next-generation mobile networks. After that, we explore the services brought by MEC and give a particular attention to its Radio Network Information Service, which constitutes one of the key motivations of this thesis. We then shed light on what the emergence of MEC implies for transport layer congestion control in terms of challenges and opportunities. We then proceed to introduce our new congestion control algorithm, named SIGMA (Simple Increase in Goodput based on MEC Awareness), that takes advantage of some of these opportunities to enhance the uplink transmission. We detail the design and key concepts behind our solution and showcase its efficiency w.r.t. to existing solutions through various simulations on the NS3 network simulator. Lastly, at the end of the chapter, we discuss the limitations of our approach and propose some improvements for the future.

Conceptually, the edge computing paradigm can be defined as the fact of running applications/-services or hosting content at the edge of the network. Unlike with traditional cloud computing where applications are run on remote data centers, that are generally far from the end user, with edge computing, the data center environment is brought close to the end user, at the edge of the network. This allows among others, a significant reduction in the end-to-end delay, the reduction of the burden on centralized data centers, the reduction of congestion on the Internet, the possibility to perform computation offloading to prolong battery life or to free the end-user device from running computation-intensive tasks. The significance of these properties depends on the degree of proximity between the edge servers and the end-user. For instance, even with a negligible access network delay, it is physically impossible to provide a two-way delay of $1\ ms$ with an edge server located 200 kilometers away from the end-user (i.e., considering the speed of light). As such, the

degree of proximity plays a key role in the concept of edge computing in general. For this reason, cellular networks come as good candidates for unlocking the full potential of edge computing since they allow the edge servers to be as close to the end-user as the base station is.

Also, for 5G and future cellular networks to be able to provide Ultra Reliable Low Latency Communications (URLLC), it is crucial to deploy the server applications involved in these kinds of communications as close as possible to the base station. In fact, even though remarkable efforts have been made in 5G to significantly reduce the user-plane latency in the access network (as detailed in the previous chapter), delay-sensitive applications would still observe significant end-to-end delays if they are not deployed sufficiently close to the RAN. Hence, edge computing is considered as a key technology to unlock the full potential of 5G. It is expected to cover at least 25 percent of the entire 5G business and enable over 60 percent end-to-end latency reduction [42].

Driven by the importance of edge computing in cellular networks, several initiatives around mobile edge computing have been launched during the past few years. Among them, the Multi-Access Edge Computing (MEC) initiative, proposed by the European Telecommunications Standards Institute (ETSI) Industry Specification Group (ISG) receives particular attention thanks to its alignment with the ETSI Network Function Virtualization framework and 3GPP standards [42]. Moreover, as will be extensively described below, the ETSI MEC framework also brings other advantages besides providing low latency and relaxing the core network.

3.1 ETSI Multi-Access Edge Computing : MEC

According to the definition given by the ETSI ISG in [100] and [40], Multi-Access Edge Computing (MEC) enables Mobile Network Operators (PLMN) to offer application developers and content providers, a cloud-computing environment within the RAN, in close proximity to mobile subscribers. The RAN edge offers ultra-low latency and direct access to real-time radio network information, which can be used by application/service developers or third-parties partners to enhance end user's experience or offer context-related services based on specific insight from the radio network. These advantages make MEC a suitable support technology for 5G. However, it is important to note that MEC is not a 5G-only feature, since its main target is mobile networks and not a specific generation in particular. Following this principle, the ETSI industry specification group has proposed a reference architecture that is agnostic to the evolution of cellular networks. This allows for instance, a MEC environment deployed in a 4G network to be reused in 5G.

3.1.1 Reference Architecture

The ETSI MEC reference architecture proposed in [40] is composed of several functional entities communicating with each other through specific reference points. As illustrated in Figure 3.1, the main components of this architecture are :

- The MEC host : which provides the physical/virtual infrastructure (i.e., compute, storage and network resources) necessary in order to run the MEC applications and the MEC platform (described below). The MEC host also provides a configurable data plane that applies traffic steering rules (received from the MEC platform) so that incoming requests for registered edge applications are redirected towards the local network.
- The MEC Platform (MEP) : which hosts MEC services and provides an environment that allows MEC applications to discover, advertise, consume and offer MEC services. As men-

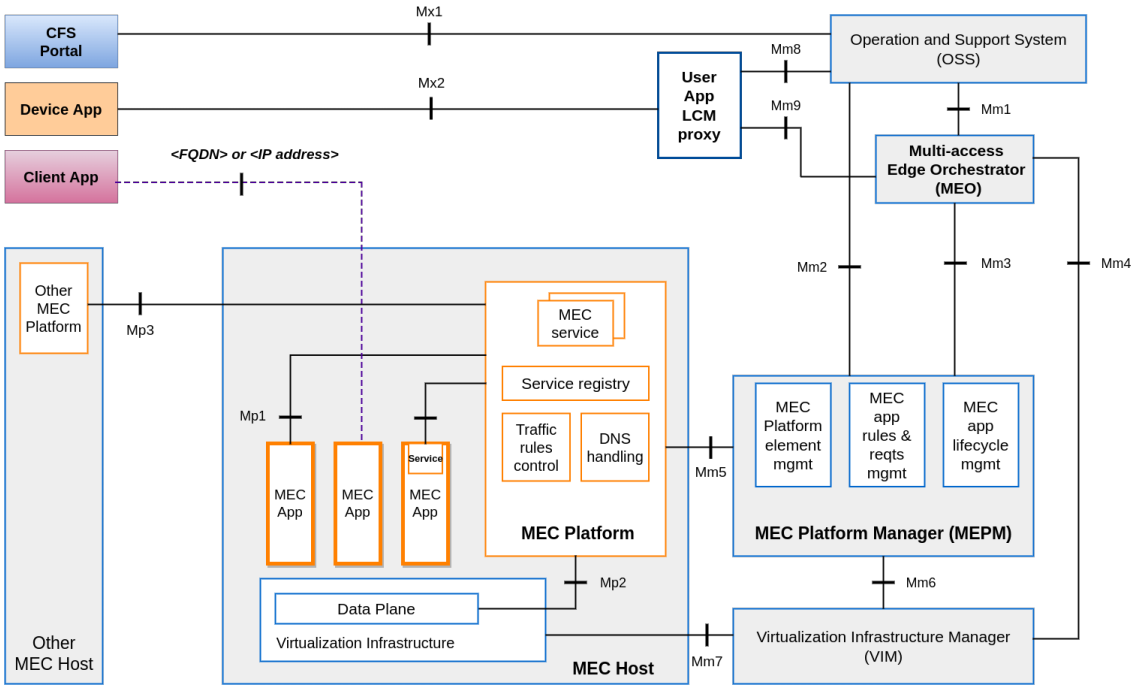


Figure 3.1 – ETSI MEC Reference Architecture (reproduced based on [100] and [40])

tioned earlier, it is also responsible for configuring the data plane by applying application specific traffic steering rules and DNS records. It receives these parameters from the MEC platform manager.

- The MEC Application (MEC App) : which is the actual application deployed at the edge (e.g., containerised application or inside a Virtual Machine) on top of virtual/physical infrastructure provided by the MEC host. As illustrated in the architecture, the MEC Apps can interact with the MEC platform through the Mp1 reference point to consume various services. These services, known as MEC services can allow the MEC apps to influence the traffic management/steering to some degree and to leverage valuable information such as location information or real-time information about the access network.
- The Device Application : which is an application on the developer's UE or laptop that can interact with the MEC system through the Mx2 reference point. The goal of the Mx2 API is to offer application life cycle management features, so that the device application can query/update/delete available or running MEC applications or instantiate new MEC applications. In short, the device application (not to be confused with the client application on the end-user device) allows the developers to manage and make their MEC applications available to the target audience (i.e., the end-users/app-users). Alternatively, developers can also manage their MEC applications though the web portal or Customer Facing Service (CFS) portal maintained by the network operator (if available).
- The User application Life Cycle Management (LCM) Proxy : provides and exposes the Mx2 API, so, it receives life cycle management requests from the device application (e.g., MEC Apps on-boarding, instantiation, termination requests, etc.) and communicates with

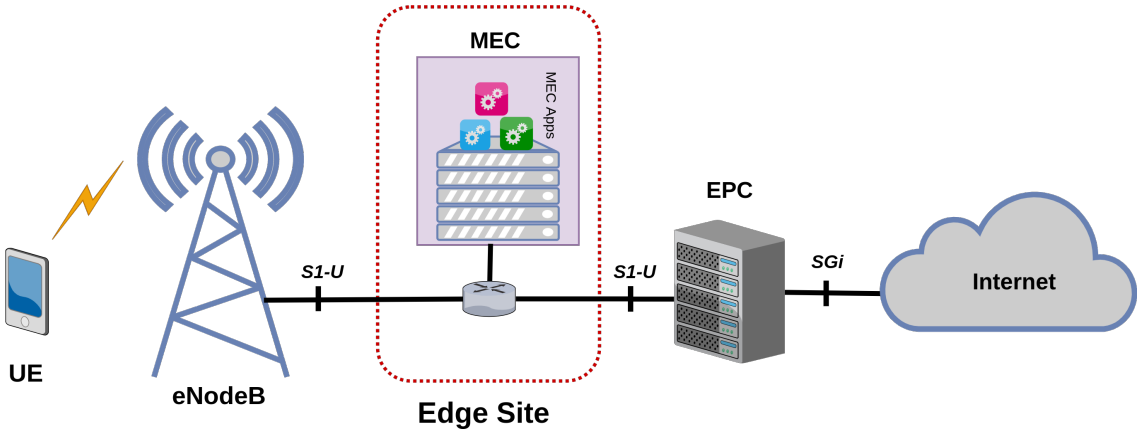


Figure 3.2 – MEC deployment following the Bump in the wire approach

the OSS (Operations Support System) and the MEC Orchestrator in order to process and execute them.

- The MEC Platform Manager (MEPM) : which handles life cycle operations on behalf of the LCM proxy or the MEC Orchestrator. It also takes care of the rules and requirements related to the deployment and operational-use of the MEC applications (e.g., service authorizations, traffic rules, DNS configuration, etc.).
- The MEC Orchestrator (MEO) : which constitutes the highest level of management in the entire ETSI MEC framework. The MEO maintains an overall view of the whole MEC ecosystem which may consist of multiple MEC hosts. It is responsible for managing the mutualized virtual/physical resources, handling application placement, on-boarding new application packages, performing integrity and authenticity checks when required, and enforcing other operator-specific rules, across multiple MEC hosts.

It is important to note that, all the entities described above represent functional components, meaning that they may be located altogether in one place or distributed across the operator network, depending on how the MEC Framework is implemented and deployed.

3.1.2 Deployment Options

The ETSI MEC system can be deployed in both 4G and 5G networks. Nevertheless, as we will see below, it should be noted that, its integration with 5G is more intuitive and more straightforward than with 4G. The reason being that, unlike 4G, 5G was designed with edge computing concept in mind, and therefore allows edge computing by default.

Starting with 4G, the ETSI ISG proposed in [48], two main approaches for the integration of MEC into the 4G architecture. These approaches are named : "Bump in the wire" and "Distributed EPC". The "Bump in the wire" approach encompasses all the deployment options in which the MEC system is deployed between the base station and the core network, in other words on the S1 interface (see Figure 3.2). In these cases, since the user data traffic is encapsulated inside GPRS Tunneling Protocol (GTP-U) over the S1 interface, the MEC host's data plane must be able to properly handle GTP-U encapsulated user packets (i.e., removing GTP headers for incoming traffic and adding GTP headers for outgoing traffic towards the core).

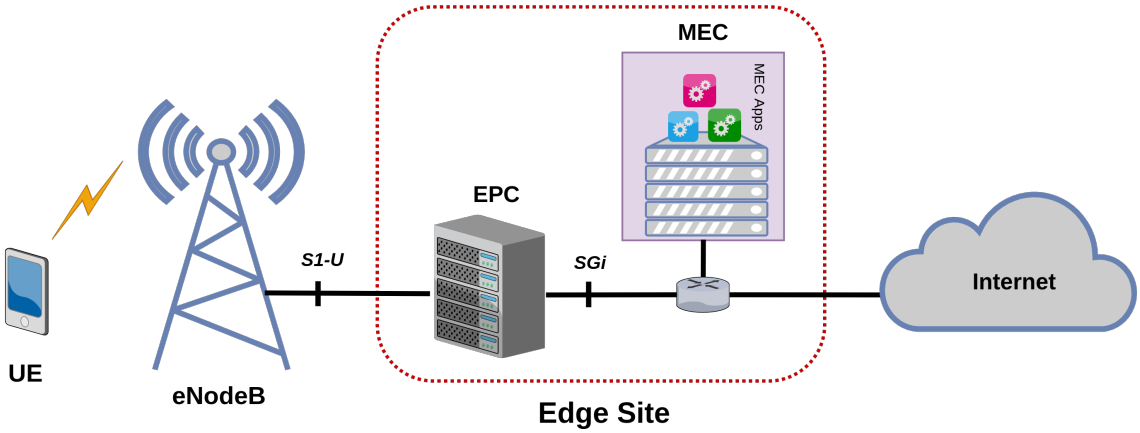


Figure 3.3 – MEC deployment following the Distributed EPC approach

Regarding the second deployment approach, i.e., the "distributed EPC" approach, it encompasses all the deployment scenarios in which the MEC host includes all or some part of the EPC (as shown in Figure 3.3). Unlike the previous approach, here the MEC data plane sits on the SGi interface, so does not need to support GTP-U encapsulated traffic. In fact, this deployment scenario requires less changes to the operator network, since the EPS tunnel is terminated by default by the P-GW or the S-GW in the MEC host, therefore the inner traffic (user IP traffic inside the tunnels) is automatically routed towards the right MEC application depending on the IP addresses or FQDNs (Fully Qualified Domain Name).

However, when it comes to deploying MEC in 5G, a completely different approach is taken. In fact, 5G already enables edge computing by allowing the core network (especially the SMF) to select a UPF close to the UE while allowing this UPF to steer the UE's data traffic towards a Local Area Data Network (LADN) [9]. As indicated in [9], the traffic steering rules in this context can be based on the UE's subscription data, UE location, or information from an Application Function (AF). Thus, taking into account these new edge computing enablers, ETSI ISG proposes in [70] an adapted deployment approach for 5G. As illustrated in Figure 3.4, in this approach, the data plane is provided by a local UPF and the MEC system management entities and the MEC platform act as AFs, thus communicating with 5G Network Functions (directly or through the Network Exposure Function) in order to get valuable information about the network and influence traffic steering. Technically speaking, in such a scenario, the MEP is expected to interact with the Policy Control Function (PCF) to request traffic steering by giving specific information about the traffic to be steered. The PCF will then transform the request into routing rules and policies, which will be used by the SMF to identify and appropriately configure the data plane of the UPF connected to the target MEC host. Also, for making real-time RAN information available to MEC Apps, the MEC platform can retrieve all the needed radio information either from the NEF (Network Exposure Function), or directly from the Centralized Units (CUs) and Distributed Units (DUs) of the target base stations (if authorized by the network operator).

3.1.3 Radio Network Information Service : RNIS

According to the definition in [40], a MEC service is a service provided and consumed either by the MEC platform or a MEC application. The ETSI ISG has already identified a certain num-

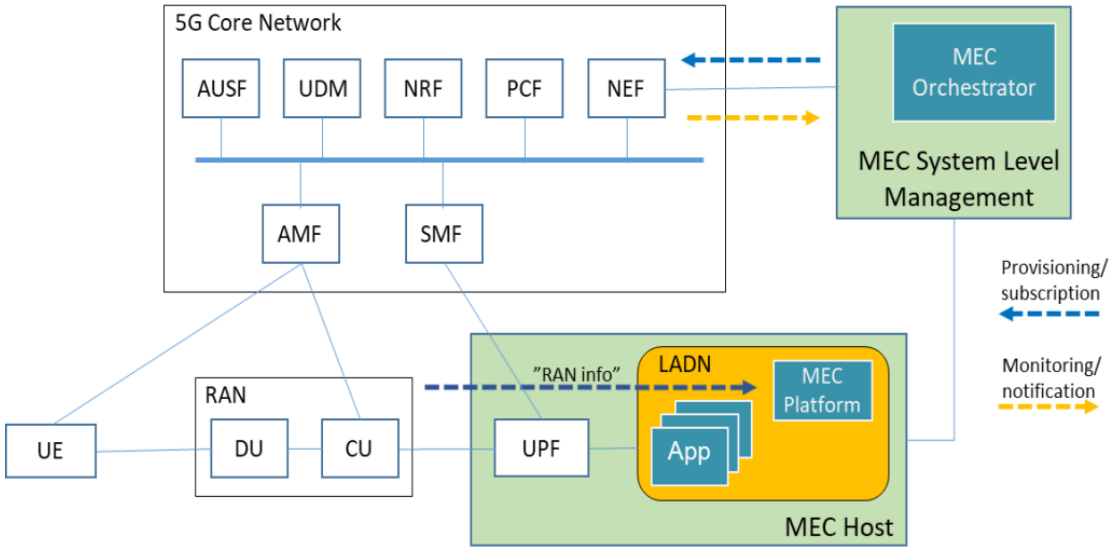


Figure 3.4 – ETSI MEC integration into the 5G architecture (copied from [70])

ber of default MEC services that can be exposed by the network operator and consumed by the deployed MEC applications through the Mp1 reference point. These services are :

- The Radio Network Information Service (RNIS) : which provides authorized MEC applications with real-time radio network related information. The MEC applications that subscribe to this service can also choose the granularity at which they receive radio information (e.g., per UE, per cell, or per period of time).
- The Location service : which provides authorized applications with location-related information (e.g., list of UEs in particular locations, Cell IDs, etc.).
- The BandWidth Management (BWM) service : which allows the authorized MEC applications to influence bandwidth allocation and prioritization of certain incoming or outgoing traffic.
- The Multi-access Traffic Steering (MTS) service : which allows the authorized MEC applications to steer, split, or duplicate application data traffic across multiple access networks.

Among these services, we give a particular attention to RNIS as it provides valuable information about the network conditions, therefore it may constitute a relevant asset from a congestion control perspective. As described in the ETSI GS MEC-012 specification [39], the applications that subscribe to this service, periodically receive various contextual information about the cells associated to the MEC host, such as : up-to-date radio network information regarding radio network conditions; measurement information related to the user plane; information about the connected UEs in the cells (e.g., information about the UE's radio bearers, channel quality, throughput, resource allocation, etc.); information about the cells Physical Resource Block usage; etc. The specification also defines the data model, messages and methods for implementing a RESTful Radio Network Information (RNI) API (Application Programming Interface). One of the advantages

of such an API, is that it gives the possibility to the edge applications to dynamically adapt their behaviors based on the current RAN conditions, as demonstrated by the ETSI RAVEN POC [99] which showcases a video optimization MEC application that dynamically adjusts the quality of the video streams according to the radio conditions of the users.

The use of RNIS in real-world experimentation has been enabled by FlexRAN [45], an open-source SD-RAN (Software Defined RAN) platform that decouples the RAN control plane from its data plane and allows them to interact through an adapted southbound API, which can be comparable to Openflow in the SDN (Software Defined Network) world. At a high level, the architecture of FlexRAN consists of a controller deployed somewhere in the network that controls and collects information from some agents collocated with the base stations. More specifically, the controller communicates with the agents through the FlexRAN protocol (which is based on TCP) in order to request specific information about the RAN or send configurations for base station's data plane. The agents on their turn, apply the provided configurations or fetch the requested information from the data plane via the FlexRAN southbound API, which defines a set of functions that allow the controller to fetch/set configurations ; request statistics from the RAN and receive various event notifications (e.g., random access attempts, handover, etc.). Finally, the controller makes all these features available to higher-level applications through its web-based northbound API.

In fact, comparatively speaking, the statistics that can be obtained from the FlexRAN controller are quite similar to the radio information indicated in the RNI API specification [39]. More specifically, the FlexRAN statistics provided by the controller include information such as : the cell's total PRBs, the used sub-carrier spacing, the Radio Network Temporary Identifier (RNTI) of each connected UE, the number of PRBs allocated to each UE, the Channel Quality Indication (CQI) reported by each UE, the corresponding Modulation and Coding Scheme (MCS), the corresponding Transport Block Size (TBS) computed by the base station for each UE, etc. Considering this property, FlexRAN fulfills the functional requirements of the RNIS specification, therefore it can be used in real-world scenarios as a solution for providing and consuming RNIS. This has already been demonstrated by the authors in [13] and [99] throughout their real-world experimentations in which FlexRAN is used as a provider of RNIS.

3.1.4 Opportunities for enhancing congestion control and transport-layer mechanisms

So far we have mostly highlighted how important is the ETSI MEC for enabling certain services such as URLLC or enhancing some applications (e.g., via computation offloading, via RNIS etc.). However, it should be noted that, the use of mobile edge computing also speeds up the rate at which ACK-clocked CCAs (i.e., most loss-based CCAs in general) increase their congestion window, thus shortening the download time, which alleviates to some degree, the effects of bufferbloat on delay-sensitive flows. For instance, let's consider the traditional scenario of a single user maintaining at the same time, a sheer download flow that uses a loss-based CCA and a delay-sensitive flow. In this scenario, the sheer-download flow would complete faster if the corresponding server application is deployed at a MEC host close to the user rather than on some remote location on the Internet. The reason being that the speed at which loss-based CCAs increase their congestion window is inversely proportional to the end-to-end RTT, hence it takes less or more time to download the same file depending on whether the file server is located close or far away from the end-user.

And if the sheer download takes less time to complete, it naturally follows that the delay-sensitive flow spends less time suffering from excessive buffering.

Besides enabling reduced download time, the ETSI MEC also gives accurate information about the location and the capacity of the bottleneck (between the MEC Apps and the end-users). The availability of such information (not explicitly available to CCAs until now), can tremendously simplify congestion control. For instance, the bandwidth discovery or probing techniques used in traditional CCAs become almost unnecessary since the MEC platform can already give more accurate information about the real-time available bandwidth. In light of this fact and also by considering the new network architecture resulting from the introduction of edge computing, we set out to rethink congestion control in the context of MEC-enabled cellular networks.

In fact, the main goal of any CCA is to properly exploit the capacity of the bottleneck pipe while limiting congestion. In order to better accomplish this task on the Internet, any CCA must consider at least three important characteristics of the underlying network (i.e., Internet) : (1) the bottleneck can be located anywhere on the Internet ; (2) the bottleneck capacity is not known in advance and can change over time ; (3) the bottleneck can be shared by other flows belonging to other users. These inherent aspects of the Internet explain why traditional CCAs need to rely on probing mechanisms (e.g., Slow Start, Cubic's growth function, BBR's ProbeBW etc.) in order to discover the available bottleneck bandwidth and adapt to some degree to bandwidth increase/decrease. However, if we consider that the CCA is no longer deployed on a random/remote location on the internet, but on a MEC host at the edge of a cellular network, then we can safely make the following assumptions :

- The bottleneck location is known. It is actually the radio link (between the user and the base station), since the network operators always provision their backhaul network (i.e., the segment between the base station and the core network) in such a way that the peak data rate or at least the aggregate average data rate of all the associated cells is supported [80]. Therefore, as illustrated in Figures 3.2 and 3.3, when the edge site is deployed on the backhaul segment (i.e., on S1/N3 interface), the only link that remains and that can constitute a bottleneck is the air interface. After all, if the backhaul does not support the cell's peak data rate, there will be no benefits for network operators to invest in large frequency bandwidth in the first place.
- The bottleneck capacity corresponds to the physical resources allocated to the UE by the base station and can be accurately provided either by the UE itself or by the MEC platform through the RNI API.
- The bottleneck queue is not shared by other users. Each user has its own separate queue, meaning that competition for bandwidth can exist only between flows of the same user, but not between flows belonging to separate users.

In fact, from a transport-layer standpoint, we are convinced that the characteristics described above undoubtedly pave the way for designing lightweight CCAs and more sophisticated transport-layer optimization solutions for MEC-enabled cellular networks. In order to prove that, we first designed and evaluated new a lightweight CCA, that significantly increases uplink TCP throughput while limiting excessive buffering. As will be shown below the design of this CCA mainly relies on the fact that the bottleneck location and capacity are known to the UE.

3.2 SIGMA : a lightweight Uplink-oriented and MEC-aware CCA

In this section we present a new CCA named SIGMA (Simple Increase in Goodput based on MEC Awareness) that speeds up uplink TCP transmissions while limiting excessive buffering on the user's device. We evaluate SIGMA performances in terms of goodput and delay and show that it significantly outperforms existing CCAs in all the considered scenarios.

3.2.1 Motivation

The proliferation of instant messaging and social network applications, the common-use of cloud storage and the increasing use of wearable devices (e.g.,smart watches) have caused a significant increase in the uplink traffic of cellular networks. According to recent studies, the ratio between uplink and downlink traffic is getting closer to 1 [14, 88], which triggers the alarm for proper uplink congestion control and traffic management. In fact, most CCAs are deployed on servers, and therefore, they only focus on the downlink. The congestion control in the uplink direction is handled by the CCA configured on the user's device, which is set by default to Cubic on Linux, Android and Windows operating systems [51, 92]. However, as extensively described in the previous chapter, Cubic and all the other loss-based CCAs are known for overshooting the radio bottleneck and creating excessive buffering. Moreover, in the case of uplink traffic this excessive buffering occurs at the user's device and not at the base station (as it is the case with downlink traffic). The reason being that the maximum amount of bytes the user's device can send per TTI is limited by its actual number of allocated uplink resources. In other words, in the uplink direction, the base station always receives the amount of bytes that corresponds to the physical resources it allocates to the UE, hence a buffering occurs on the UE when its applications generate too much data than its allocated physical resources can convey. This type of buffering, termed on-device buffering in [51] can lead to a phenomenon known as on-device bufferbloat, which affects both uplink and downlink transmissions as shown in [51].

Indeed, several solutions have been proposed over the years, in order to mitigate this issue and enhance uplink congestion control. However, it appears that, most of them cannot provide both high throughput and acceptable delay at the same time, especially in cases where multiple uplink data flows are progressing in parallel in the user's device :

For instance, X-TCP described in Chapter 2 sets its congestion window to the uplink BDP, estimated from the uplink physical resources allocated to the UE and the minimum RTT. Then it considers the link as congested when the current RTT exceeds the minimum RTT value by a threshold value of $10\ ms$, at which point, the estimated BDP is multiplied by a scaling factor $\lambda = 0.85$ [14]. In other words, the connection tries to exploit 85% of the actual link capacity in case of congestion. Indeed, such a technique is very effective for avoiding/reducing excessive buffering in the case of a single uplink data flow. However, in case of multiple uplink data flows, each individual flow would try to exploit 85% of the maximum uplink data rate, which inevitably result in excessive buffering. In such a context, using BBR instead of X-TCP might provide a better trade-off between delay and throughput, since with BBR, each individual flow would decrease its sending rate in order to match the reducing delivery rate and rediscover the minimum RTT. However, since BBR is unaware of the radio information (exchanged between the UE and the base station), it periodically probes for bandwidth, therefore it may require several RTTs in order to discover and grab released resources (e.g.,when some UEs leave the cell or release their uplink resources). Furthermore, because of the periodicity of the uplink Scheduling Requests (SRs) and

the delay variations on the radio segment, BBR tends to underestimate/overestimate the minimum RTT, hence experiences unnecessary throughput oscillations [73, 92].

After analysing the existing approaches, we found out that : (a) some misconceptions about the cellular uplink traffic, and also about the user-side TCP stack (e.g.,TCP stack on the user's device) make the development of effective uplink-oriented CCAs very challenging; (b) the new architectural changes brought by edge computing and MEC are not taken into account although they greatly simplify congestion control; (c) some traditional TCP mechanisms are no longer necessary in the era of MEC-enabled cellular networks. In light of these findings, we set out to design a new uplink-oriented CCA that is aligned with the particularities of cellular uplink traffic and the growing trend towards edge computing and MEC.

3.2.2 Understanding On-Device TCP and Cellular Uplink Traffic

Android is today the most popular operating system in the world, with over 2.5 billion active users [114]. With such a high popularity, together with the growing load of uplink traffic, it is important to understand how these devices perceive and handle uplink flows.

On-Device TCP Congestion Control Algorithm : Aside from their popularity, another important information to note about Android systems is the fact that they use Cubic as their default TCP congestion control algorithm [51, 92] just like Linux or Windows. This implies that, all the TCP flows generated by our Android devices (i.e., uplink TCP traffic) use Cubic on the uplink direction, no matter the CCA in use on the server side. Indeed, the user's device can use only one CCA at a time, unless its applications are launched on separate VMs (which is not the case in current Android systems). So, by default, we can consider that all the uplink TCP flows on the user's device use the same CCA (e.g., Cubic or any other configured CCA). Also, since the air interface is the only bottleneck and the data traffic of each user is isolated (thanks to the per-user queues), the uplink flows cannot be affected by the behaviors of the CCAs running on the other devices in the cell and vice-versa. Therefore, it does not matter whether the CCA configured on the user's device is fair to other CCAs. In fact what matters the most in the case of uplink scenarios, is whether the CCA in use is fair to itself. In other words, if all the flows controlled by this CCA are treated equally or proportionally.

Importance of real-time UDP traffic over TCP upload : Many applications such as cloud storage, health/fitness or social network apps usually generate background TCP traffic in the uplink direction [83], which may occur while the user is maintaining a real-time UDP traffic in the same direction (e.g.,voice or video chat). In such a context, the attention of the user is mainly drawn towards the real-time traffic that they are directly perceiving. For instance, in case of video calls/-conferences (e.g., via whatsapp, Google meet apps, etc.), the user starts experiencing bad quality when the one-way delay exceeds 150 ms [89], which may quickly happen when a concurrent TCP upload (i.e., an aggressive Cubic flow) is progressing in the background. A good uplink-oriented CCA should be aware of such a phenomenon and must always give more priority to competing real-time UDP flows in order to avoid hindering the user's experience.

3.2.3 Replacing TCP Slow Start by Max Start

The main goal of TCP Slow Start and other TCP Startup algorithms is to gradually discover the bottleneck capacity without introducing too much congestion. The Slow Start phase starts with an Initial Congestion Window (IW), whose size controls the number of unacknowledged segments

sent at the beginning of the connection, hence greatly affects the transfer speed (especially in case of short-lived connections). An IW that does not reflect the capacity of the network generally introduces performance penalties. For instance, a too small IW value can unnecessarily prolong short data transfers while a too large IW (i.e., beyond what the network can handle) may lead to unnecessary packet losses and retransmissions. For these reasons, it was initially recommended to use an IW value of 1 segment [98], which was a relatively safe value at the early days of TCP. Then, with the evolution of the network speeds, it has been changed to 2, then to 4 segments [98]. Today the default IW value in most linux-based systems is 10 segments. This has mainly been motivated by the global adoption of broadband Internet access and also by the possible reduction in page load times that may be obtained, as demonstrated in [38]. Actually, the authors in [38] demonstrate that 90% of HTTP responses from the top-100 and top-500 sites (in 2010) as well as 90% of Google search and Gmail responses fit in about 15kB, which roughly corresponds to 10 TCP segments. Therefore, setting the IW value to 10 segments, allows these responses to be delivered in only one RTT, which significantly reduces page load times, especially in cases where the RTT or the BDP are large. However, it should be noted that Internet and access technologies have tremendously evolved since 2010, therefore the chosen IW value of 10 segments at that time, neither reflect the capacity of today's access networks, nor the response size of current websites or recent/future bandwidth-intensive applications (e.g., AR/VR traffic). But, choosing a one-fit all value is challenging and almost virtually impossible, considering the fact that most users use shared wireless access technologies, which are known for their quick capacity variations over time. However, the good news is that, in the specific context of cellular uplink, we found that such a challenge should not even exist in the first place, and can be easily solved. More specifically, we found that relying on the traditional Slow Start approach along with a static IW value in the case of mobile-generated uplink TCP traffic is unnecessary and can be quite counterproductive.

In fact, as mentioned earlier, the main goal of the Slow Start approach is to safely discover the bottleneck bandwidth, but if the available bandwidth is already known in advance, there is no need to waste several RTTs probing for a known value. As a matter of fact, it appears that, in cellular networks, the user's device is aware of the amount of physical resources it has been allocated by the base station and can also decode if needed the amount of resources allocated to the other users in the cell. Basically, the base station transmits Downlink Control Information (DCIs) on the Physical Downlink Control Channel (PDCCH) every TTI, which include among others, the uplink scheduling grants, i.e., the number of uplink PRBs allocated to the UEs scheduled on the Physical Uplink Shared Channel (PUSCH). And since this information is received and decoded by all the UEs connected to the base station, it becomes quite straightforward for any UE to compute , at any time, its actual assigned radio bandwidth as well as the maximum radio bandwidth it can reach after considering the unused/unallocated physical resources available at the base station. Based on this logic, instead of using a global and static IW for all the TCP connections in the UE, it would make more sense to allow each individual TCP connection (at the beginning of the connection) to dynamically configure their IW value based on the reachable radio bandwidth, and then let the CCA in use make sure the maximum bandwidth is not exceeded based on some observed network metrics (as it is traditionally done). Technically speaking, with this approach, the TCP connection just sets its IW value during the 3-way handshake to the product of the maximum reachable radio bandwidth and the minimum observed RTT (i.e., sets the IW to the BDP) and let the CCA configured in the user's device handle the rest. However sending back-to-back such a large burst of packets in the first RTT creates a temporary queue, termed "good queue" in [86] as

it will dissipate in the next RTT. To avoid this temporary queue, it is necessary to pace the data at the bottleneck rate, i.e., at the computed radio capacity, at least during the first RTT.

We named the approach of using an IW value equivalent to the BDP together with a pacing rate at the bottleneck capacity during the first RTT, "Max Start" because it allows the flow to start directly sending at the maximum reachable rate, as opposed to Slow Start, which is forced to progress gradually since it doesn't know the maximum rate. Considering the very high data rates promised by 5G (up to 20 Gbps) and the fact that most connections are short-lived and transfer small amounts of data [38, 59], the Max Start approach would significantly reduce the transfer time of web objects and other interactive data. In other words, it would directly improve the perceived connection speed on the end-user side. Also, in the case of medium and large data transfers, the traditional Slow Start algorithm generally exceeds several times the BDP and can provoke excessive and severe on-device buffering. Besides, as demonstrated by the Kleinrock's optimal operating point [26], increasing the amount of bytes in flight beyond the BDP does not bring any throughput gain, but rather just introduces more buffering and inflates the RTT. Therefore a BDP worth of data is the optimal amount of bytes in flight that allows maximum throughput with minimum delay. And if this amount of bytes can be reached and maintained since the very first RTT (by using Max Start), then the obtained performances (throughputs and delays) would always surpass that of a traditional Slow Start, unless the time to reach the BDP using exponential increase is negligible (i.e., in case of very short RTTs or small BDPs), at which case Max Start would deliver at least the same throughput as Slow Start, but with zero or significantly lower degree of buffering. To better illustrate this principle, let us consider the simple case of a file upload scenario. In this scenario, let us define : D the size (in terms of MSS) of the file ; N the BDP of the link ; W the amount of bytes in flight, and D_{n} the sum of data already uploaded from the first to the $(n)^{th}$ RTT, i.e., when the amount of bytes in flight is equal to the BDP (when $W = N$). Then, by considering the traditional exponential increase during Slow Start and the default initial window value of 10 MSS, we can express the amount of data uploaded from the first RTT till the moment the amount of bytes in flight reaches the BDP as follows :

$$D_{\text{n}} = 10 \times 2^0 + 10 \times 2^1 + ... + 10 \times 2^{n-1} = 10 \times \frac{(1-2^n)}{(1-2)} = 10 \times 2^n - 10\,. \tag{3.1}$$

From Equation 3.1, since we know that $10 \times 2^{n-1}$ corresponds to N (i.e., the BDP), we can also express D_{n} as :

$$D_{\text{n}} = 2(10 \times 2^n \times 2^{-1}) - 10 \implies D_{\text{n}} = 2 \times N - 10\,. \tag{3.2}$$

The above Equation shows that the amount of bytes in flight reaches the BDP in cases where the size of the file to be uploaded is equivalent to two times the BDP. If the size of the file is bigger than that, the Slow Start phase will start introducing excessive buffering since the bottleneck can only transmit N data segments per RTT. With these considerations, and by knowing that it takes n RTTs to reach N, it becomes simple to compute the time T_{D} it takes to transfer the whole data (i.e., D) :

$$T_{\text{D}} = [RTT \times n] + [RTT \times \frac{1}{N}(D - D_{\text{n}})], \quad \text{where} \quad n = \log_2\left(\frac{N}{10}\right) + 1\,. \tag{3.3}$$

The above Equation 3.3 is applicable only when the size of the data to be uploaded (D) is superior or equal to two times the BDP. Otherwise the transfer ends before the BDP is reached, therefore :

$$\text{When} \quad D \leq D_{\text{n}}, \quad \text{then,} \quad T_{\text{D}} \leq T_{D_n} \leq [RTT \times n]\,. \tag{3.4}$$

In Equation 3.3, the value of n dominates the upload time when the second half of the equation is not significant. Basically in cases where N (i.e., the BDP) is large or D is small. This is clearly illustrated by Equation 3.4, in which it can be seen that the upload time is directly proportional to n since D is less than or equal to D_{n}. In any case, regardless of the file size, reducing n will reduce the upload time in both equations, which brings us to Max Start. In Max Start, the BDP is reached since the first RTT, which means that the value of n becomes 1, so n always remains at its minimum value in both Equations 3.3 and 3.4. This clearly demonstrates that the upload time with Max Start will always be inferior to the upload time with the traditional Slow Start. The upload time enabled by Max Start, denoted $T_{D_{max}}$ is given by the two equations below, in which n is replaced by 1 :

$$T_{D_{max}} = RTT + [RTT \times \frac{1}{N}(D - D_{\text{n}})], \tag{3.5}$$

$$\text{When} \quad D \leq D_{\text{n}}, \quad \text{then,} \quad T_{D_{max}} = RTT\,. \tag{3.6}$$

However, it is important to note that the adoption of Max Start does not prevent excessive on-device buffering, since multiple uplink flows may exist at the same time on the user's device. In such a case, each individual flow would start sending at the bottleneck rate, which would naturally create a degree of buffering proportional to the number of flows. Therefore it becomes necessary to use a suitable CCA, that will take over after the Max Start mechanism in order to make sure the allocated radio bandwidth is appropriately distributed among the competing flows while guaranteeing an optimal or near optimal radio link utilization. For this purpose, we have introduced a new uplink-oriented CCA, named SIGMA, that adjusts its congestion window according to the RTT and the radio link utilization.

3.2.4 SIGMA Design

At a high level, SIGMA is an uplink-oriented CCA that prevents bufferbloat while maximizing the link utilization. To do that, it decreases its congestion window value proportionally to the RTT increase when the most recent RTT is two times above the RTTmin. Then it gradually increases the congestion window when the radio link is not fully used. The default value of the congestion window is none other than the estimated achievable radio BDP. This is computed as follows : The sender first estimates the maximum fair-share PRBs it can receive (from the base station) denoted P_{max} by considering the total number of PRBs in the cell denoted P_{cell}, the number of active users denoted N and the number of unused PRBs denoted P_{unused} :

$$P_{max} = \frac{P_{cell}}{N} + P_{unused}\,. \tag{3.7}$$

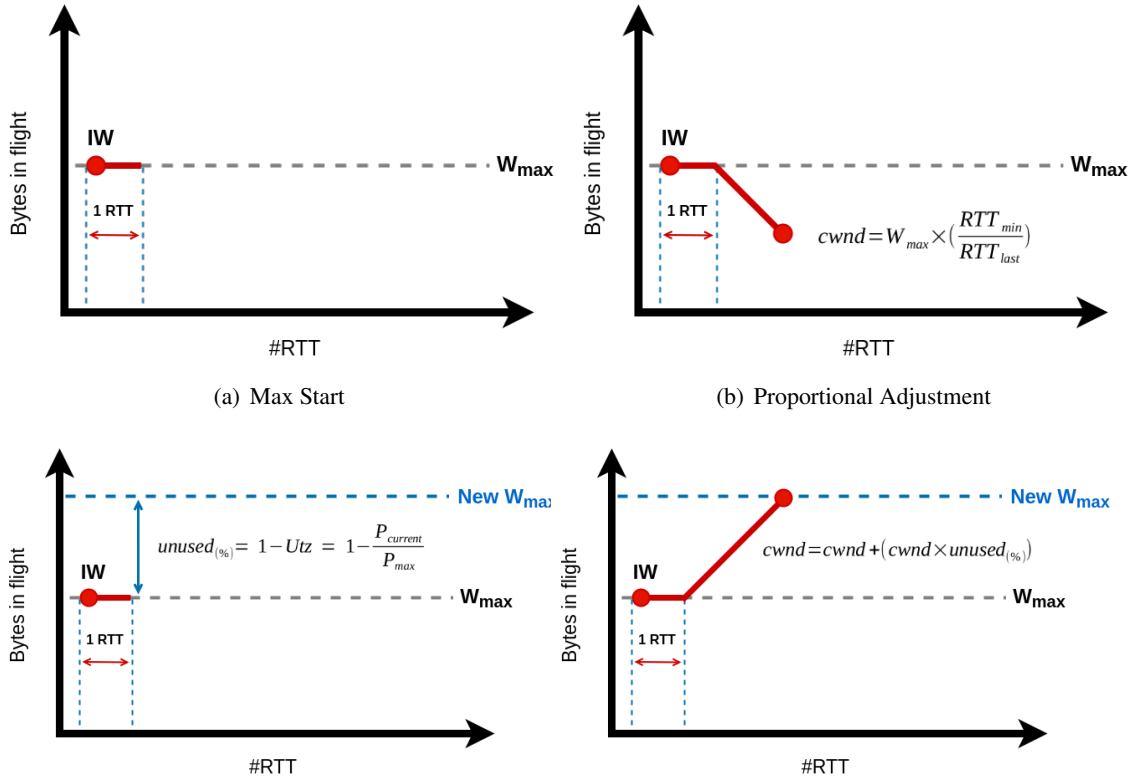
Then, the obtained P_{max} value is associated with the current MCS and mapped to the corresponding TBS using the TBS mapping tables and indications in 3GPP TS 36.213 [3] and 3GPP TS 38.214 [5]. After obtaining the TBS that corresponds to P_{max}, denoted TBS_{max}, it becomes simple for the sender to compute the maximum achievable data rate in $bits/s$ (denoted C_{max}) since it knows the physical level transmission time or TTI :

$$C_{max} = \frac{TBS_{max}}{TTI}\,. \tag{3.8}$$

The computed data rate (C_{max}) is then combined with the minimum RTT (RTT_{min}) in order to calculate the current BDP (denoted W_{max}), which will be used as the default and initial value of the congestion window :

$$W_{max} = C_{max} \times RTT_{min} \,. \tag{3.9}$$

In fact, the first computation of W_{max} occurs during the 3-way handshake. At this step it is used as the initial window (IW) by the Max Start mechanism, which lasts only one RTT. After the Max Start phase, the congestion window is shaped according to two new additional states. We named these two states Proportional Adjustment and Safe Increase. Figure 3.5 gives a high level view of SIGMA's behavior during each of its operating states, which are : Max Start, Proportional Adjustment and Safe Increase.



(a) Max Start

(b) Proportional Adjustment

(c) Safe Increase (1/2) : Estimating unused BW

(d) Safe Increase (2/2) : Exploiting unused BW

Figure 3.5 – Congestion Window evolution during the three operating states of SIGMA

Proportional Adjustment state : As it names suggests, during this state, the congestion window is adjusted proportionally to a given parameter. In SIGMA, this parameter is the RTT increase, or more specifically the ratio of the current RTT (RTT_{last}) over the minimum RTT (RTT_{min}). This ratio basically tells how many times the BDP of the link has been exceeded. Therefore, in order to drain the queue created by this overshooting and match the available bottleneck delivery rate, one may think that we simply need to multiply the previous congestion window value by the inverse of the aforementioned ratio. Indeed, this completely drains the queue, but contrary

to intuition, the resulting congestion window with this method can be half the supported BDP, especially when the competing flows are synchronized. In fact, it is important to remember that the bottleneck can transmit one BDP worth of packets per RTT only if these packets are spaced out according to the bottleneck per-packet transmission time. If we denote t_{pkt} this transmission time, then it is basically given by $t_{pkt} = packet_{size}/C_{max}$. Consequently, if a BDP (i.e., W_{max}) worth of packets arrive at once for instance from two flows, using each a congestion window of $W_{max}/2$, the bottleneck would take at least $t_{pkt} \times (W_{max}/packet_{size})$ to transmit them. In other words, at least one additional RTT is added, which doubles the current RTT despite the fact that the BDP is not exceeded twice. So, multiplying the previous congestion window of these two flows by RTT_{min}/RTT_{last} would result in each flow having a congestion window of $W_{max}/4$, which naturally causes the flows to operate at half their achievable throughput.
For this reason, we designed the Proportional Adjustment state in such a way that it operates at twice the RTT_{min} and sets the congestion window based on the actual BDP and not based on the previous congestion window. Technically speaking, we make sure the Proportional Adjustment state is entered only if the current RTT or RTT_{last} is twice greater or equal to the RTT_{min}, in other words, when $RTT_{last} \geq 2 \times RTT_{min}$. Then each time the Proportional Adjustment state is entered, we set the congestion window to the product of the actual BDP and the ratio RTT_{min}/RTT_{last} :

$$cwnd = W_{max} \times \frac{RTT_{min}}{RTT_{last}} . \tag{3.10}$$

With this method, when two flows send at the same time $W_{max}/2$ worth of data and introduce twice the RTT_{min}, the congestion window of each flow is set to half the current BDP, i.e., $W_{max}/2$, instead of $W_{max}/4$. Also, it is important to note that the misleading delay increase due to the arrival of a BDP worth of packets (at once) does not occur in case of one flow, since the Max Start mechanism spaces out the packets in flight according to the bottleneck rate (i.e., thanks to pacing), thus avoiding the bursty arrival at the bottleneck since the first RTT. With two flows, as explained above, the equilibrium is reached at the first proportional adjustment. On the other hand, in case of K competing flows (with $K > 2$), the congestion window follows a sawtooth pattern, bouncing between $2W_{max}/K$ and $W_{max}/2$ (because a temporary queue is introduced at every RTT) until the equilibrium is finally reached.

Safe Increase state : This state is entered when the current RTT is inferior to twice the minimum RTT (i.e., when $RTT_{last} < 2 \times RTT_{min}$). This generally occurs when a competing upload flow on the user's device finishes its transmission, or when some users in the cell release some or all their physical resources. The goal of the Safe Increase state is to discover and use these released resources without creating an excessive buffering in the process. In order to do that, it first computes the actual radio link utilization, denoted Utz by dividing the actual allocated amount of PRBs ($P_{current}$) by the maximum expected amount of PRBs (P_{max}) :

$$Utz = \frac{P_{current}}{P_{max}} . \tag{3.11}$$

Then, it deduces that the radio link is not fully used when the utilization is below 1 (i.e., $< 100\%$). In this case the previous congestion window is immediately increased by a certain percentage equivalent to $(1 - Utz) \times 100$. In other words, the previous congestion window (which represents the previous bandwidth) is increased by a percentage that is proportional to the unused or added portion of bandwidth. This method prevents excessive buffering even in the worst case scenario

where K competing flows (with $K > 2$) discover at the same time that $\lambda\%$ of the radio link is not used. Because in this case each flow increases its congestion window of W_{max}/K by $\lambda\%$, hence the aggregate amount of bytes in flight increases by $K \times (W_{max}/K) \times \lambda\%$, which corresponds to $W_{max} \times \lambda\%$. In other words the added or unused portion of bandwidth is adequately distributed among the competing flows. Also, we make sure each flow spreads out the added amount in their congestion window over the whole RTT so that unnecessary bursts can be limited. This is actually inspired from the way Reno increases its congestion window by 1 segment per RTT during the Congestion Avoidance phase [19]. Technically speaking, each time a new acknowledgment arrives, the congestion window in the Safe Increase state becomes :

$$cwnd = cwnd + [MSS \times (1 - Utz)]\,. \tag{3.12}$$

With this method, the last acknowledgment generates a $cwnd$ value of $cwnd + cwnd \times (1 - Utz)$. The congestion window is increased per RTT following this pattern until the current RTT grows two times above RTT_{min}, i.e., until $RTT_{last} \geq 2 \times RTT_{min}$, at which point the Proportional Adjustment state is entered. And during the Proportional Adjustment state, when the RTT decreases slightly below $2 \times RTT_{min}$, the flow enters the Safe Increase state. This bounce and back between Proportional Adjustment and Safe Increase continues until the end of the flow. A typical SIGMA flow is basically characterized by this behavior, which is clearly shown in the SIGMA pseudo-code described below in Algorithm 1.

3.2.5 Implementation and Evaluation

We have implemented SIGMA on the ns-3 network simulator by modifying the UE and writing a new CCA. More specifically, we have extended the ns-3 mmWave UE stack so that the UE can share some radio level information with the TCP layer, such as the maximum number of symbols per slot, the number of allocated slots, the slot duration, the number of connected UEs, etc. Then, we've implemented SIGMA at the TCP layer and exploited these shared radio-level metrics based on the pseudo-code shown in Algorithm 1. In our simulation setup, as illustrated in Figure 3.6, the UEs communicate with the base station through a 28 GHz mmWave air interface, which provides a maximum uplink data rate of 900 Mbps. The base station is connected to the core network via a 10 Gbps backhaul link. The core network is connected to several TCP servers via a 10 Gbps. In accordance with the distributed EPC approach, these TCP servers play the role of edge/MEC applications. Other simulation parameters are detailed in Table 3.1. With our simulation setup, we evaluated the performance of SIGMA in terms of throughput and delay and compared it with Cubic and BBR, since they are currently the most used CCAs on the Internet. In our simulations, we repeated each individual experiment five times and reported our results within a 95% confidence interval. Through our experiments we decided to evaluate the following scenarios :

TCP uploads from a single UE : This first scenario aims to showcase the efficiency of SIGMA both in case of single and multi flow scenarios. The goal here is to show that SIGMA always outperforms the traditional CCAs (NewReno, Cubic, BBR) regardless of the upload size or the number of competing flows in the user's device. To better illustrate that, we consider a first sub-scenario in which the UE uploads a single file at a time ; and a second sub-scenario in which the UE uploads 4 files at the same time. We repeat these two basic sub-scenarios for different

Algorithm 1 SIGMA Pseudo-Code

```
1:  C_max ← TBS_max/TTI
2:  W_max ← C_max × RTT_min                ▷ Radio Capac. and BDP are computed every TTI
3:
4:  procedure MAX_START :                                          ▷ Start of MaxStart
5:      IW ← W_max
6:      Pacing ← true
7:      Pacing_rate ← IW/RTT_min
8:
9:      if (Segments_Acked) ≥ (IW/Segment_size) then
10:         Pacing ← false
11:         goto PROPORTIONAL_ADJUST        ▷ One RTT has passed => End of MaxStart
12:
13: end procedure
14:
15: procedure PROPORTIONAL_ADJUST :
16:
17:     if (RTT_last ≥ 2 × RTT_min) then
18:       cwnd ← W_max × (RTT_min/RTT_last)
19:     else
20:         goto SAFE_INCREASE
21:
22: end procedure
23:
24: procedure SAFE_INCREASE :
25:     foreach ACK do
26:     if (RTT_last ≥ 2 × RTT_min) then
27:
28:         goto PROPORTIONAL_ADJUST
29:     else
30:       Utz ← P_current/P_max
31:       cwnd ← cwnd + Segment_size × (1 − Utz)
32:
33: end procedure
```

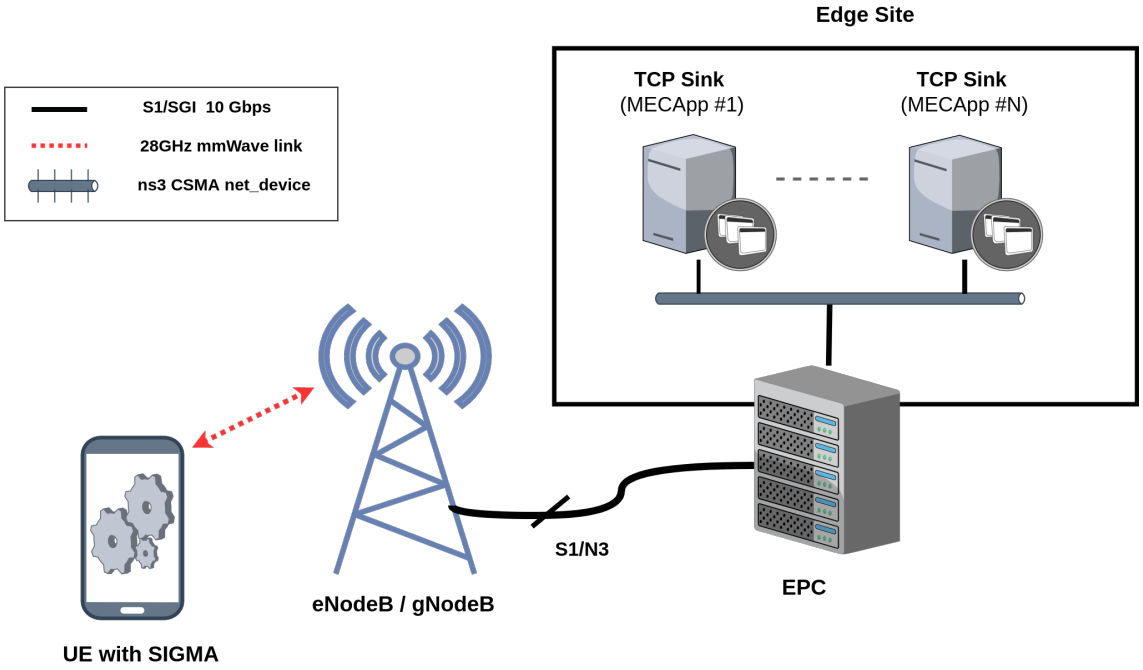
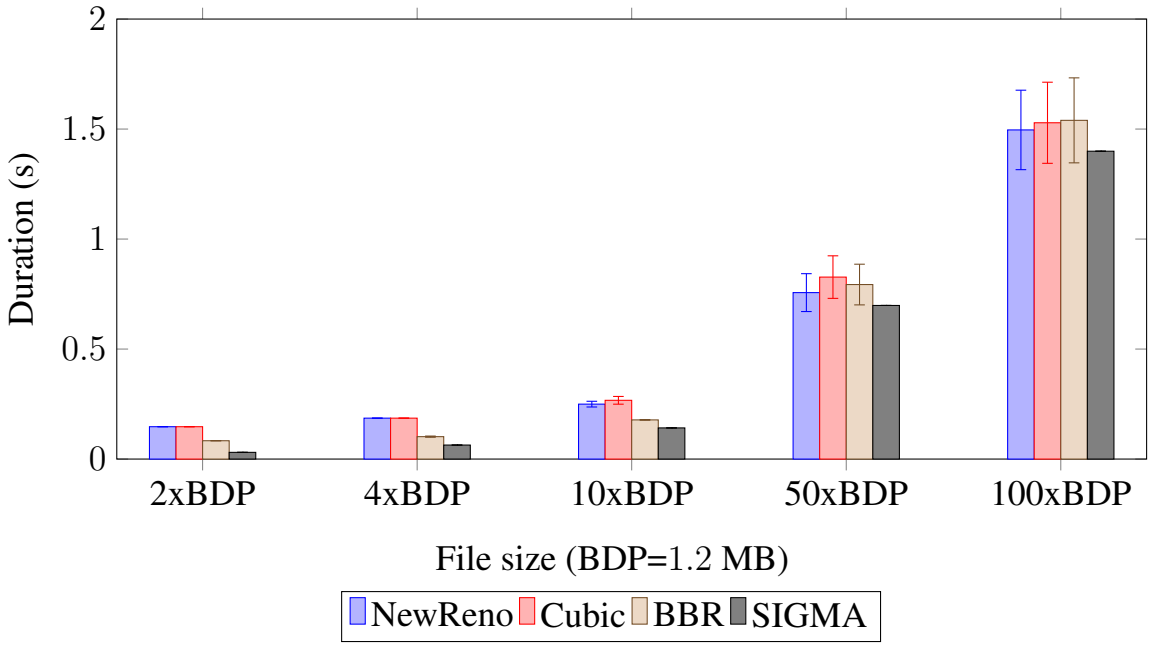


Figure 3.6 – SIGMA Evaluation Testbed

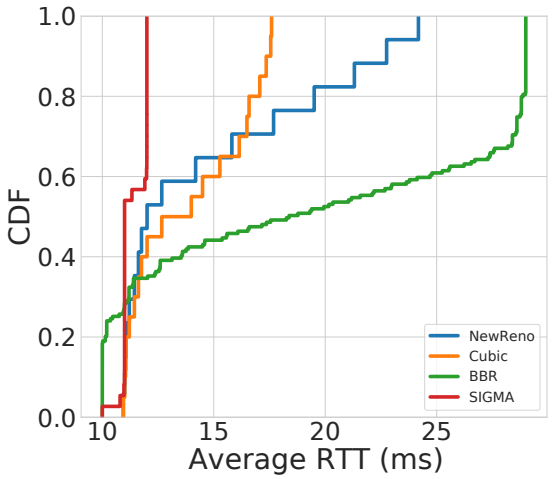
TABLE 3.1 – Global ns-3 simulation parameters

Parameters	Values
Carrier Freq.	28 GHz
Duplex mode	TDD
Bandwidth	200 MHz
MAC Scheduler	Round Robin (FlexTtiMacScheduler)
SR Periodicity	100 μs
Frame Duration	10 ms
Subframe Duration	100 μs
Max. OFDM Symbol per Subframe	24
OFDM Symbol Length	4.16 μs
Subframe per Frame	10
RLC Mode	Ack. (AM)
RLC Buffer	10 MB
TCP MSS	1440 B
Default TCP IW	10 MSS
Peak Phy. Uplink Data rate	900 Mbps
End-to-End RTT	10 ms

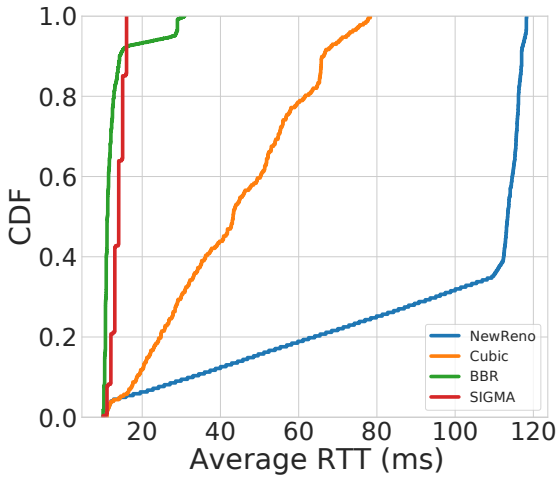
CCAs and with different file sizes ranging from small/medium (corresponding to few BDPs) to very large files (equivalent to several dozens of BDPs).



(a) TCP Upload Duration for different file size



(b) RTT Distribution during the 10xBDP upload

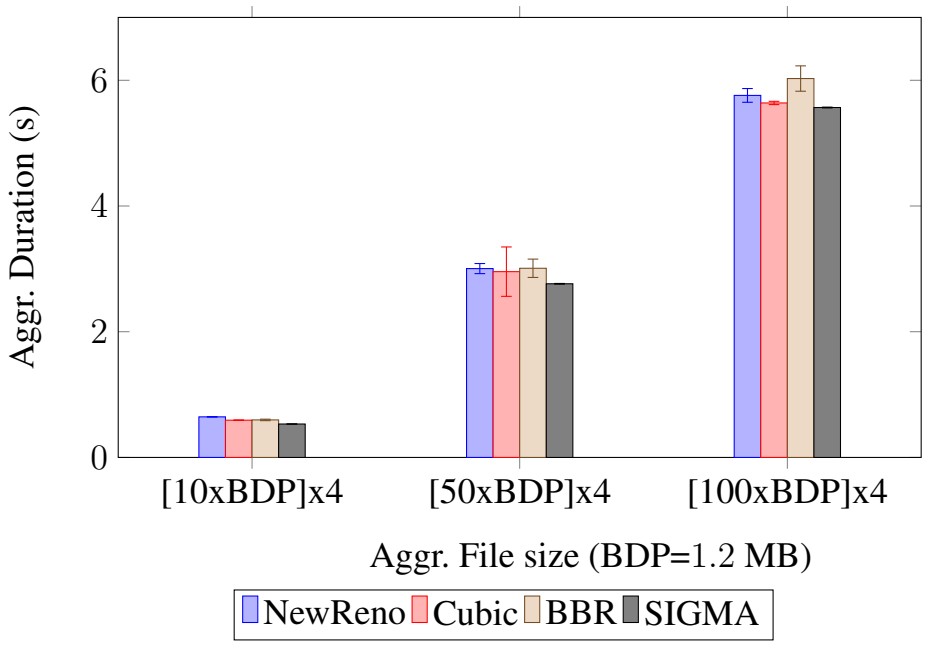


(c) RTT Distribution during the 100xBDP upload

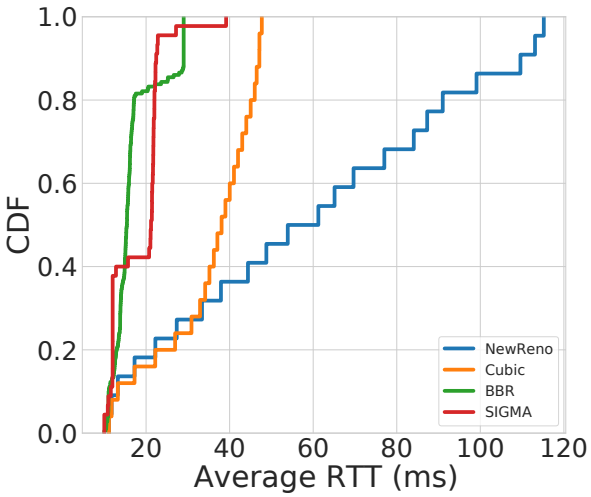
Figure 3.7 – Single TCP upload from a single UE for various file sizes

Figures 3.7, 3.8 and 3.9 show the results of these experiments for five successive runs with a 95% confidence interval.

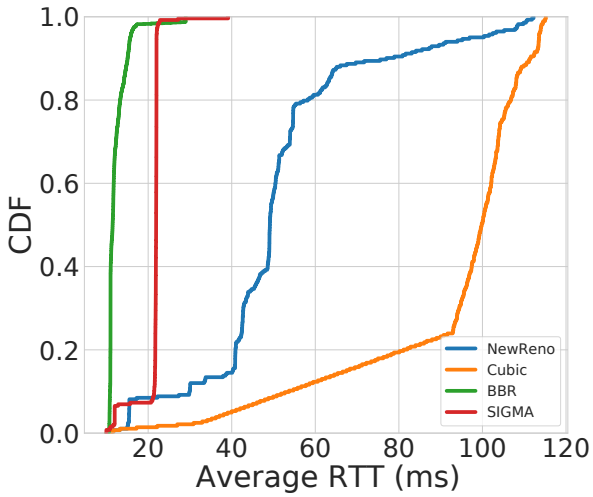
Figure 3.7 shows the upload durations and RTT distributions for the first sub-scenario, i.e., when the UE is using a single flow. From these results, when we look at Figure 3.7(a), we can



(a) Aggregate Upload Duration of 4 simultaneous flows



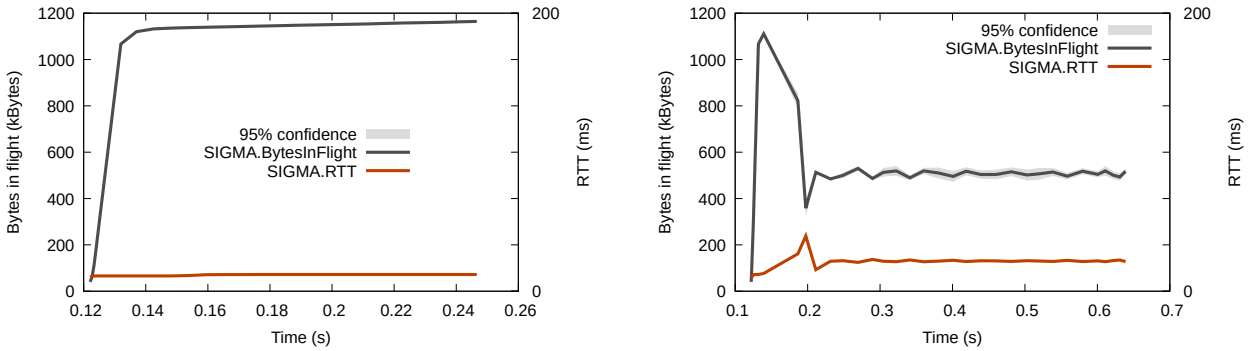
(b) Aggregate RTT Distribution for {10xBDP}x4



(c) Aggregate RTT Distribution for {100xBDP}x4

Figure 3.8 – Multiple simultaneous TCP uploads from a single UE for various file sizes

clearly see that SIGMA outperforms the other evaluated CCAs (NewReno, Cubic, BBR) in terms of upload duration for all the considered file sizes. More specifically, in the considered cases, we observed that SIGMA enables up to 80% decrease in the upload duration with respect to NewReno and Cubic, and up to 60% decrease with respect to BBR. As expected, we can see that the performance boost brought about by SIGMA is more significant for short and medium flow sizes. This is normal because, in the case of very large/fat flows, the time it takes to reach the BDP



(a) Bytes in flight and RTT in case of 1 connection

(b) Bytes in flight and RTT in case of 4 connections

Figure 3.9 – Behavior of a 10xBDP SIGMA flow in the case of one and 4 connections

W_{max}, which corresponds to $n \times RTT$ is almost negligible compared to the time it takes to send the rest of the data ($D - Dn$), especially when the end-to-end RTT is small. For example, in our case, for a flow size of 120MB (i.e., 100×1.2MB), it takes $n = 7$ RTTs (which corresponds to 7×10ms=70ms) in the traditional slow start phase to reach an amount of bytes in flight equivalent to our BDP of 1.2MB. And these 70ms are clearly negligible compared to the time it takes to send the remaining 118.8MB, which corresponds to 99×10ms (i.e., $(118.8/1.2) \times 10$ms) or roughly 1s. However, it is important to note that, when the end-to-end RTT is relatively large (e.g., when the edge server is far from the base station), the reduction in upload duration would become more significant even for a fat flow, since the Max Start phase would allow the flow to save a non-negligible amount of time. Regarding the RTT increase during this first sub-scenario, it can be seen from Figures 3.7(b) and 3.7(c) that SIGMA is better than BBR and the other CCAs both in case of short/medium and large flow sizes. In fact, when compared to BBR (which clearly provides a better trade-off between delay and throughput than NewReno and Cubic), we observe that SIGMA enables up to 55% average RTT reduction in the $90th$ percentile. This is explained by the fact that BBR is known for building queues during its Start-up and ProbeBw phases, while SIGMA Safe Increase phase prevents unnecessary cwnd increases or queue buildups when the radio link is already fully used. The behavior of SIGMA in this particular context is explicitly shown in Figure 3.9(a), in which it can be observed that, in case of one flow, SIGMA consistently maintains an amount of bytes in flight roughly equivalent to one BDP (which is 1.2MB in our case). Compared to Cubic and NewReno, SIGMA allows up to 5-factor and 10-factor average RTT reduction, respectively (see Figure 3.7(c)). The fact that the obtained RTT reduction when compared to Cubic is lower than that of NewReno was expected since we already know that the Hystart algorithm used by Cubic reduces excessive buffering during Slow Start to some degree (as explained in Chapter 2).

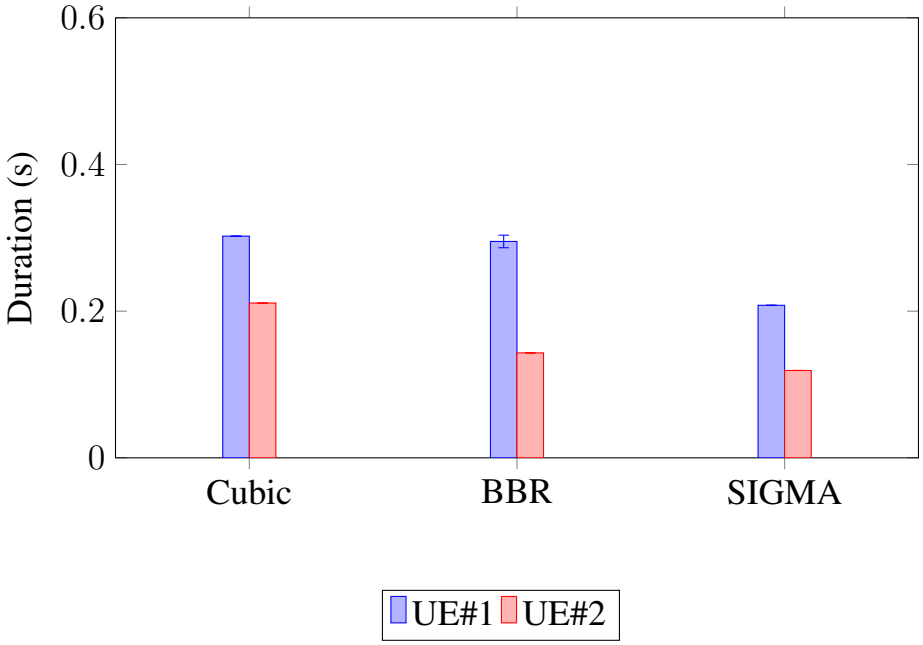
The results of the second sub-scenario, i.e., the case of multiple parallel flows are shown in Figure 3.8. To better understand the reported results, it is important to note that Figure 3.8(a) shows for each CCA, the combined upload duration of all the competing flows, which basically corresponds to the time frame between the moment the first flow is established until the moment the last flow finishes its upload. We chose this metric since all the 4 competing flows in our experiments start at the same time and send the same amount of data; therefore, any appropriate CCA that is fair to its own flows should allow these 4 uploads to complete around the same time,

as it would allow one big upload (equivalent to the sum of these 4 flows) to complete. Similarly, Figures 3.8(b) and 3.8(c) show for each CCA, the combined RTT distribution of all the competing flows. When observing these results, we can notice that, in terms of goodput or upload duration, SIGMA always outperforms the other CCAs as expected. However, the obtained performance boost is lower compared to the previous sub-scenario, since here the aggregate upload size is much more significant. To be more specific about the performance gains, we observe that SIGMA allows up to 18% and 10% decrease in upload duration with respect to NewReno and Cubic; and up to 10% decrease with respect to BBR. Regarding the average RTT distribution, it can be seen from Figures 3.8(b) and 3.8(c) that the RTTs exhibited by SIGMA in the $90th$ percentile are slightly above that displayed by BBR. This is due to the fact that SIGMA spends most of its time in the Proportional Adjustment state in case of multiple flows (as shown in Figure 3.9(b)). And since this state is designed to operate at $2 \times RTT_{min}$, it is normal that the observed RTTs remain most of the time around $2 \times RTT_{min}$, which corresponds to 20ms in our case. By authorizing such an RTT increase, we allow SIGMA to exploit the radio capacity in a much better way than BBR (which partly contributes to the 10% decrease in upload duration). Also, when compared to NewReno and Cubic, we can observe that SIGMA enables over a 5-factor average RTT reduction in the $90th$ percentile. By considering these improvements together with the previously described results, we can clearly confirm through these first experiments that, regardless of the size or the number of competing flows in the user's device, SIGMA always offers a better tradeoff between goodput and delay than Cubic, NewReno, and BBR.

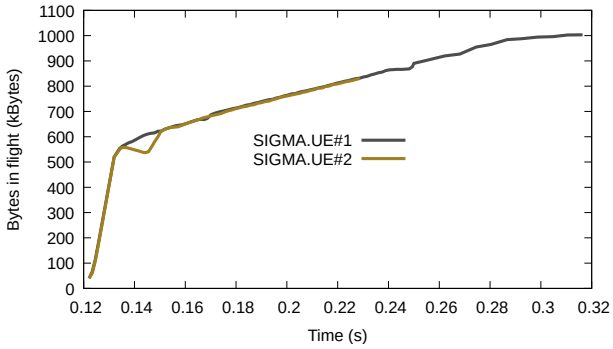
.

TCP uploads from multiple UEs : In this scenario, we highlight SIGMA's ability to rapidly discover and use idle or released physical resources in the cell. For this purpose, this scenario considers two users (UE#1 and UE#2) connected to the same base station and uploading at the same time two files with different sizes. Aside from the multi-user aspect, the most important parameter to consider in this scenario is that the files the two users are uploading don't have the same sizes. More specifically, the first user uploads a medium-sized file equivalent to $10BDP$, while the second user uploads a smaller file with a size of $4BDP$. With this configuration, the second user finishes its upload and releases some physical resources, while the first user is still uploading. In such a situation, an adapted CCA would immediately discover the released resources and use them, thus allowing the first UE to obtain higher throughput. With that in mind, through the results obtained from this second scenario, we evaluate how fast is SIGMA, compared to Cubic and BBR at discovering and adapting to bandwidth increase.

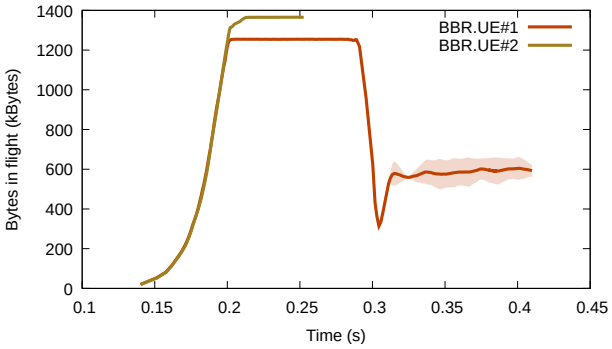
Figure 3.10 summarizes the results of this experiment by showing the upload duration of the two UEs when using Cubic, BBR, and SIGMA (see Figure 3.10(a)) and by highlighting the behavior of SIGMA and BBR in the presence of idle/released resources (see Figures 3.10(b) and 3.10(c)). From Figure 3.10(a), it can be seen that SIGMA outperforms Cubic and BBR in terms of upload duration or goodput. With respect to CUBIC, SIGMA enables 33% decrease in upload duration. This performance gain is enabled not only thanks to SIMAS's Max Start mechanism, but also because of the fact that Cubic continues to follow an incremental growth even after UE#2 releases some resources, meanwhile SIGMA quickly grabs the released bandwidth thanks to its Safe Increase mechanism. Likewise, compared to BBR, we observe that SIGMA decreases the upload duration by 30%. As expected, the performance gain with respect to BBR is slightly lower than that obtained over Cubic for the simple reason that BBR does not leave the Start-up phase prematurely (i.e., no HyStart mechanism), and also because BBR's ProbeBw mechanism grabs released bandwidth more quickly than Cubic's incremental increase. Nevertheless, as shown



(a) Upload duration with different CCAs



(b) Bytes in flight in case of SIGMA



(c) Bytes in flight in case of BBR

Figure 3.10 – Two UEs uploading a big and a short file at the same time

in Figures 3.10(b) and 3.10(c), it appears that BBR's ProbeBw, is still slower compared to SIGMA's Safe Increase. In fact, while the latter requires at least 8 RTTs in order to adapt to bandwidth increase (see Chapter 2), the former takes just 1 RTT, since it is able to detect and exploit the percentage of unused resources within one RTT (as shown Algorithm 1).

From the results obtained throughout this second scenario, we can also confirm that SIGMA is better than traditional CCAs, including BBR, at adapting to bandwidth variations, which are common in cellular networks.

In order to simplify the reproducibility of the presented scenarios and experiments, we made the source code of SIGMA as well as the script to reproduce the reported results publicly available in our git repository [115]

3.2.6 Limitations and Future Improvements

Despite SIGMA's simplistic design and superior performance w.r.t. traditional CCAs, it is important to note that its adoption in real-world cellular networks requires some modifications on the end-user's device. In fact, in order to make a mobile phone decode control messages targeting other users in the cell, it is necessary to customize the LTE/5G firmware inside the phone, which is generally proprietary. Therefore, an effort is required from the equipment manufacturers side to facilitate access to this information. Nevertheless, we plan to bypass this limitation by enhancing SIGMA so that it can infer the available/achievable radio bandwidth without directly retrieving information from the LTE/5G stack. The feasibility and efficiency of such a technique has already been demonstrated by the authors in [110].

Also, since SIGMA is based on the assumption that all the uplink TCP flows originate from the end-user's device, its design does not consider the possibility of other CCAs sharing the uplink bandwidth with SIGMA. However, there is one specific scenario that still allows such a situation to occur. This scenario is mobile tethering/hotspot, i.e., when the user shares its connection with other users. In this case if the other users are using a loss-based CCA rather than SIGMA, a large bufferbloat can occur and penalize delay-sensitive or real-time uplink traffic. To avoid that, a lightweight TCP split proxy can be installed alongside SIGMA on the device in tethering/hotspot mode. This will allow all the traversing TCP connections to be intercepted and controlled by SIGMA.

Another important point to consider is that we designed SIGMA based on the assumption that the air interface becomes the bottleneck in case of MEC deployment. This assumption is also applicable when the end-server is located on a CDN or on an edge cloud [54]. In fact, even if the bottleneck is located on the wired segment, SIGMA would eventually adapt to the available capacity, since it would decrease its congestion window proportionally to the RTT increase. However, a certain degree of buffering may still be introduced since SIGMA's congestion window reduction and increase are done based on the radio bandwidth. With that in mind, in our future work, we plan to evaluate SIGMA in scenarios where the location of the bottleneck is unknown. This will allow us to tell whether it is necessary or not to include a bottleneck detection mechanism into SIGMA's design.

3.3 Summary

In this chapter, we have explored the ETSI MEC framework and its RNI service and highlighted their impact on TCP traffic. Some new opportunities/aspects related to the adoption of edge computing or MEC that could help enhance/simplify TCP congestion in cellular networks have been identified and studied. This allowed us to introduce a new uplink-oriented CCA, named SIGMA, which makes full use of some of the identified opportunities/aspects. Besides presenting SIGMA's main building blocks and algorithm, we have also dissipated some misconceptions about cellular uplink and shared our vision on how a good uplink-oriented CCA for cellular network should behave. Through our experiments with SIGMA, we have shown that it offers a better trade-off between goodput and delay than both Cubic and BBR. We have demonstrated through some specific scenarios that the approaches used by SIGMA are more adapted for cellular uplink than the traditional bandwidth probing approaches used by today's most widely used CCAs, i.e., Cubic and BBR. To the best of our knowledge, SIGMA is the first uplink-oriented cross-layer

CCA that is built with edge computing aspects in mind and that can trivially manage multiple parallel connections without creating an excessive on-device buffering/bufferbloat.

We have pointed out SIGMA's limitations, which are mainly related to the fact that the end-user's device needs to be modified in order to decode some specific control messages. In order to bypass these limitations, we plan to enhance SIGMA's so that it can infer some radio information without explicitly invoking the 4G/5G stack. In this chapter, we focus on improving TCP uplink performance directly on the user's device. Although such an approach may seem simple at first, its wide adoption in real-world network is quite difficult because of the additional codes that must be added to the Android kernel and to the manufacturer's firmware.

Also, despite the fact that SIGMA mitigates the on-device bufferbloat issue and improves the link utilization in the uplink direction, it is important to remember that it does not solve the downlink transport layer issues. Unlike in the uplink direction, where all the competing flows originating from the user's device use the same CCA, in the downlink, these same flows may use different CCAs. Because of this, it becomes challenging to solve the transport layer issues with a new CCA (as it was the case with SIGMA), since this CCA would be affected by the negative behavior of any competing CCA. With that in mind, instead of creating a new CCA, we decided in the next chapter, to focus on improving the kind of CCAs that are used by the majority of today's TCP traffic, that is, loss-based CCAs [81].

CHAPTER 4

MEC-based approach for Loss Discrimination in 4G/5G

In this chapter we first highlight the limitations of traditional loss discrimination algorithms in correctly differentiating and discriminating losses in 4G/5G cellular networks. We then proceed to introduce our new RAN-aware loss discrimination scheme, named MELD, which leverages real-time RAN statistics via the ETSI MEC RNI service as well as TCP or UDP segment size in order to decide whether it is necessary to discriminate a particular packet loss. We present the design and implementation details of this solution and evaluate its efficiency with NewReno and Cubic in a real-world 4G testbed. We show through various experimentations that our solution improves NewReno and Cubic achievable goodput by 80% *and* 8%*, respectively. Lastly we comment on the experimental results and discuss future improvements ideas.*

4.1 Motivation

Most of today's download data flows are controlled by loss-based CCAs that consider any packet loss as a congestion signal. The default behavior of such CCAs once being notified of a packet loss is to decrease their sending rate by reducing the current window size (e.g., NewReno, Cubic). However, since all packet losses do not necessarily mean congestion (e.g., because they may also occur randomly due to a lossy wireless link), such a technique may lead to unnecessary throughput reduction [91]. LDAs have been proposed as a way to alleviate this issue by identifying and discriminating random losses. However, although the proposed LDAs seem effective on WiFi networks [91, 46], it is important to emphasize that most of them were designed with little or no consideration of cellular networks. They generally misinterpret the effects of some cellular RAN mechanisms presented in Chapter2, which make them inefficient and not adapted for today's 4G or 5G networks.

In fact, most LDA approaches proposed in the literature (e.g., NCPLD, TCP Veno or LDA_EQ) either rely on Round Trip Time (RTT) increase or buffer occupancy measurements in order to differentiate losses. These two techniques are not accurate in LTE for the following reasons :

First, detecting a loss at the link layer requires at least 8ms in LTE FDD (Frequency Division Duplexing) and another 8ms for the retransmission (see Figure 2.6). Meanwhile, in WiFi, link

layer loss detection is almost immediate (i.e., within a few μs) and retransmission may take similar time depending on the number of connected stations. Based on that, it is clear that RTT variations during link error losses are more significant in LTE, which does not necessarily mean congestion.

Second, unlike wireless access points, 4G/5G base stations are generally provisioned with large buffers in order to accommodate rapid changes in link capacity and to compensate for the additional processing overhead required for each User Equipment (UE). Therefore, those buffers must be appropriately occupied in order to reach optimal link utilization. In that case, relying only on buffer occupancy to infer congestion losses can lead to link under-utilization.

Last, depending on the number of PRBs allocated to the UE and on the Modulation and Coding Scheme (MCS) selected by the base station (based on the actual radio conditions), the resulting physical layer TBS can be several times lower than the current TCP/UDP segment or datagram size. In such a case, the 4G/5G link layer subdivides every single TCP or UDP packet into several transport blocks and transmits them one after the other. In fact, as indicated in 3GPP TR 36.912 [6] and as illustrated in Figure 2.6, unless spatial multiplexing is used, only one transport block can be sent to the UE per TTI, i.e., every millisecond (in LTE), and since each transmission is controlled by an independent Hybrid Automatic Repeat Request (HARQ) process, it follows that the number of transport blocks that can be sent successively without interruption (i.e., without waiting for an HARQ ACK or a NACK) directly depends on the total number of HARQ processes. Therefore, it becomes clear that the time required to send a single TCP or UDP packet is inversely proportional to the current physical layer TBS. Consequently, a large mismatch in size between transport blocks and the corresponding TCP/UDP packets and/or retransmissions at lower layers can lead to excessive buffering or congestion in the RAN.

In this context, it is not a good idea to discriminate all link layer losses like traditional LDAs do, simply because these losses may also lead to excessive buffering or congestion because of the aforementioned reasons or because of ARQ/HARQ retransmissions. Instead, a better approach would be to :

- First, consider the existence of two types of congestion : a first type of congestion that results from the overshooting of the CCA in use (e.g., loss-based CCAs) and a second type of congestion caused by the 4G/5G link layer mechanisms (e.g., TBS, ARQ/HARQ etc.) ;
- Second, discriminate only link layer losses that do not lead to the second category of congestion (when ignored).

The rationale of this approach is that excessive buffering due to overshooting is almost inevitable given the presence of deep buffers at the base station. This kind of congestion is generally caused by aggressive CCAs and occurs naturally even in the absence of link layer losses. Meanwhile, link layer losses are mostly random and some of them may cause additional buffering which adds up to the buffering already introduced by the CCA. As a result, ignoring all link layer losses would just further exacerbate the degree of congestion instead of improving the link utilization. Therefore, it is necessary to make sure that only link layer losses that introduce a little or an acceptable degree of buffering are ignored while link layer losses that are likely to exacerbate the degree of congestion are considered as congestive losses. In order to achieve this goal and infer such congestive losses, it is important to clearly understand to what extent some specific RAN information (e.g., CQI, TBS, RLC queues, etc.) can be used in conjunction with upper layers (TCP/UDP) information to predict excessive buffering at link layer. Retrieving and making such

radio information available to upper layers require either additional services at the base station or modifications in the UE. The latter approach is used by recently proposed cross-layer CCAs such as PBE-CC [107] or CQIC [111] that rely on client-side radio measurements to adjust their sending rate. Although these approaches guarantee higher link utilization, they also introduce computational overhead and additional power consumption at the UE. Instead, we advocate for a solution that does not involve the end user, more specifically, a solution that considers the use of the ETSI MEC RNI service [39] to collect radio information rather than relying on the user's device. In fact, as shown in the previous chapter, applications deployed at the edge can subscribe to the RNI service, which allows them to receive real-time radio information.

Thus, based on these considerations, in this chapter, we first highlight to what extent some specific information transmitted over the RAN can be used as relevant indicators of congestion in the access network. We then propose a new loss discrimination scheme for reliable transport protocols, which discriminates losses based on up-to-date RAN information collected via the RNI API and on current TCP/UDP packet sizes. We named this approach MELD, which stands for MEC-based Edge Loss Discrimination.

4.2 MELD : a MEC-based packet loss discrimination scheme for 4G/5G networks

At a high level, MELD is a novel mechanism that discriminates packet losses observed at the transport layer based on up-to-date link layer information collected from RNIS and on TCP/UDP segment size. This approach comes in two variants, MELD-DE (for Download Enhancement) and MELD-ME (for Mixed flows Enhancement). The former discriminates all losses that did not introduce a significant buffering overhead at the link layer while the latter discriminates losses not only based on link layer overhead, but also on the global degree of buffering in order to minimize the effect of bufferbloat on the competing short or interactive flows. These two variants are both designed for edge delivery scenarios such as short and large file download from servers located at the edge of the network (i.e., close to the RAN). Also, it is important to note that MELD could help improve the performance of any loss-based CCAs regardless of the underlying transport protocol (e.g., TCP or QUIC [93]). In our experiments, we evaluated its performance with the QUIC transport protocol, which is by design more resilient than TCP, to random losses and high RTT variations, which are common in wireless networks [106, 74]. In fact, by showing that MELD improves QUIC goodput, we also prove that QUIC can also benefit from LDAs like TCP, despite its enhanced design. To this end, we implemented MELD as a new loss notification algorithm on top of a well-known implementation of the IETF QUIC [93] protocol called picoquic [113] and evaluated its efficiency through controlled experiments conducted in the R2lab wireless testbed *.

In the remainder of this section, we present the building blocks of our approach by exposing the correlation between some specific radio information and excessive buffering at the link layer or congestion at the transport layer. After that, we introduce MELD, and outline the basic components of its design and its implementation.

*. R2lab Testbed : http://r2lab.inria.fr

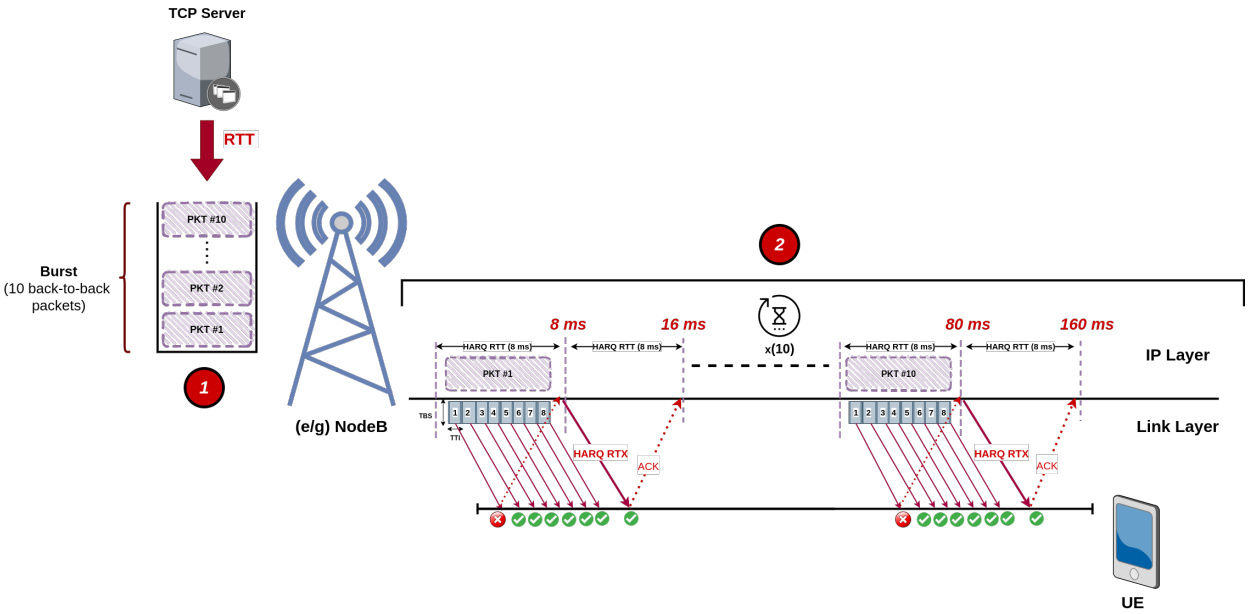


Figure 4.1 – Downlink transmission of 10 back-to-back IP packets by a BS with a capacity of 1 packet per HARQ RTT while considering 1 HARQ retransmission within each HARQ RTT.

4.2.1 Correlation between radio information and transport-layer congestion

Recent papers and experiments [58, 96] show that most reliable transport layer protocols experience throughput degradation and rate throttling over LTE networks. This phenomenon is mainly due to misinterpretations of some LTE radio access techniques/mechanisms by transport layer protocols (e.g., TCP, QUIC). For instance, Junxian Huang et al. in [58], observed that more than 12% of TCP flows in traces collected from a commercial LTE network experience undesired slow start due to the loss of a single packet. More precisely, they identified that the retransmission timeout (RTO) expired due to excessive buffering or bufferbloat. Indeed, by looking at such an observation, one could assume this excessive buffering is mainly caused by the aggressive behavior of the loss-based CCA in use. However, this assumption holds true only if the radio conditions are very good and/or if the UE is assigned sufficient PRBs. Especially, when the resulting TBS from the allocated PRBs and the selected MCS is large enough so that one TCP/UDP packet can be conveyed in one TTI or in one HARQ RTT. In contrast, when the resulting TBS is so low that sending only a single TCP/UDP packet takes more than one HARQ RTT, the link layer starts introducing significant overhead in terms of delay. In fact, in such a case, the second packet in a burst of already received packets (at the base station) would wait at least 1 HARQ RTT before the beginning of its transmission, and the next packet, 2 HARQ RTTs and so on. Technically speaking, in this particular condition, a burst of L packets would take at least $L \times HARQ_{\text{RTT}}$ to be completely transmitted and acknowledged, assuming no HARQ retransmissions occurred in the process. In case of n HARQ retransmissions, the transmission delay of the burst would increase at least by $n \times HARQ_{\text{RTT}}$. Also, in this state, all incoming packets are buffered, which eventually leads to a significant degree of buffering.

It is also important to note that, in some cases, the delay introduced because of link layer retransmissions can be more significant than the delay introduced by the bursty behavior of the

CCA. For example, lets consider a burst of 10 back-to-back packets that arrive at once at the base station. Let's also suppose the base station can send only one packet per HARQ RTT (i.e., every 8 ms). In that case, it would normally take 80 ms to completely deliver the burst to the UE. However, if 2 HARQ retransmissions occurred during each HARQ RTT, an additional delay of 160 ms (i.e., $2 \times 10 \times 8$) would be added, increasing the overall burst delivery delay from 80 ms to 240 ms. Also, note that if the base station was able to send 2 packets per HARQ RTT (i.e., if the TBS were larger at least by a factor of two), then the additional delay would have been halved (i.e., 80 ms instead of 160 ms). Figure 4.1 better illustrates this phenomenon. Therefore, we can say that the link layer mechanisms can significantly contribute to the excessive buffering depending on the radio conditions and the size of the physical layer TBS. To be more specific, it depends more on how many packets the base station can transmit in one HARQ RTT, in other words, on how many times the physical layer TBS is inferior or bigger than the upper transport layer (TCP/QUIC) packet size. However, to fully understand this phenomenon, we must also consider the parameters that affect the size of the physical layer transport block.

In fact, in LTE, each UE sends (periodically or not) a Channel Status Information (CSI) report that takes at least 8ms to reach the base station as indicated in 3GPP TR 36.912. The CSI report includes, among others, the UE computed CQI value that corresponds to a predefined MCS at the base station side. As illustrated by the authors in [71], a low CQI value indicates a low signal-to-interference and noise ratio (SINR), hence less bits per symbol in the modulation. The CQI therefore has a direct influence on the transport block size, regardless of the number of allocated PRBs. So, with a given number of allocated PRBs, the number of transport blocks required to transmit a single TCP or UDP packet highly depends on the CQI value. Consequently, the resulting number of transport blocks must be less than or equal to the number of HARQ processes so that the corresponding TCP or UDP packet can fit in one HARQ RTT. Otherwise, some transport blocks conveying some parts of the packet will be sent during the next HARQ RTT, thus increasing the risk of bufferbloat at the base station and/or eventual RTO expiration. Technically speaking, if we denote TBS, the current physical layer transport block size; S_{TB} the minimum size the transport blocks should have so that a layer-4 (TCP/QUIC) packet can be transmitted in one HARQ RTT; T_{PDU} the size of the layer-4 packet; and N_{HARQ} the number of HARQ processes, then the transmission delay of a single layer-4 packet does not exceed one HARQ RTT only if :

$$\text{TBS} \geq S_{\text{TB}}, \quad \text{where} : S_{\text{TB}} = \frac{T_{\text{PDU}}}{N_{\text{HARQ}}}. \tag{4.1}$$

Based on the above equation, we can say that when the current TBS value is less than S_{TB}, the link layer mechanisms are likely to introduce additional queuing delays and eventually excessive buffering. Therefore, ignoring packet losses during such a state (i.e., low CQI and/or small number of PRBs assigned to the UE) would most likely exacerbate the degree of congestion in the RAN. This assumption is further strengthened by the fact that losses are detected at layer-4 only after several retransmission attempts at lower layers, in other words, after a certain level of buffering has already been introduced. And since Equation 4.1 is not verified, we are sure that the introduced level of buffering is going to be significant. Conversely, packet losses occurring when the current TBS is large enough so that one packet can be completely transmitted within an HARQ RTT (i.e., high CQI and/or sufficient number of PRBs assigned to the UE) can be attributed either to a buffer overflow due to the greedy behavior of the CCA in use or to fast fluctuations of the SINR between CSI reports. In any case, losses due to buffer overflow must be taken into account. However, by basing our loss discrimination policy only on Equation 4.1, losses due to buffer overflow would

be declared only in cases where the current TBS is too small (i.e., less than S_{TB}). As a result, in cases where the TBS is large enough (i.e., good radio condition and/or high PRB allocation), every packet loss that arises from a buffer overflow is ignored, which can eventually create a persistent or a lasting congestion state, thus penalizing any competing delay-sensitive or interactive flows. The exact same phenomenon occurs in cases where the TBS is barely above the minimum required, but several packets in the burst were either lost or underwent a certain number of retransmissions. In other words, the retransmissions in this case introduce an even more significant level of buffering or even a buffer overflow. In order to prevent such situations, it is necessary to add a second condition that makes sure the level of buffering introduced in the aforementioned cases does not exceed a certain threshold. This threshold must explicitly limit the authorized level of additional buffering. All packet losses that are detected after this threshold is reached must be declared. Technically speaking, we have to make sure that the length of the associated RLC queue is not above a certain value. It is also important to make sure that this value is large enough in order to guarantee full link utilization in the case of favorable radio conditions. Therefore, it must take into account both the arrival rate at the base station (i.e., the sending rate of the CCA) and the maximum possible departure rate (i.e., the TBS that can be reached in ideal radio conditions).

Basically, most loss-based CCAs are ACK-clocked, meaning that the sender sends new data at the same rhythm it receives incoming acknowledgments [11]. The time gap between these acknowledgments is controlled by the client, which generates new ACKs at the rhythm it receives packets from the bottleneck [86] (assuming standard TCP with no SACK or delayed-ACK mechanism). In other words, if the bottleneck takes k time units to send one packet, the ACKs from the client will also be spaced by k time units, so will be the next packets sent from the sender after one RTT (especially when it is in steady state [86]). Therefore, by extension, if the base station can transmit only one packet per HARQ RTT, all the incoming packets from the server will be spaced by one HARQ RTT, starting from the second RTT. Similarly, if the base station can transmit L packets per HARQ RTT, then upon the first transmission (after one end-to-end RTT), it should start receiving a group of L back-to-back packets every HARQ RTT. Therefore, for the link to be fully utilized, at least L packets should be available in RLC buffer every HARQ RTT ; otherwise, a transmission opportunity will be missed. Based on this principle, if we denote S_{TBm}, the maximum TBS that is reachable under ideal conditions (i.e., at CQI=15), then under ideal radio conditions, the sender can fully utilize the radio link only if it is allowed to introduce a buffering of at least $S_{\text{TBm}} \times N_{\text{HARQ}}$ (which corresponds to the number of packets the base station can transmit per HARQ RTT).

Thus, to put it all together, if we denote S_{TxQueue}, the current RLC queue length, and Q_{th}, the allowed level of buffering, then the link capacity is fully usable only if Q_{th} is equivalent at least to $S_{\text{TBm}} \times N_{\text{HARQ}}$. A higher Q_{th} value yields unnecessary buffering, and a lower value prevents full link utilization, so the value must always be set to the upper bound indicated in the following equation :

$$S_{\text{TxQueue}} \leq Q_{\text{th}} \leq S_{\text{TBm}} * N_{\text{HARQ}}. \tag{4.2}$$

With this second equation, it becomes possible to check if the introduced buffering is acceptable before discriminating/ignoring packet losses when the conditions in Equation 4.1 are met. In order to enforce this approach at the transport layer, we designed and implemented a new loss discrimination scheme around it, and named this new scheme MELD.

Algorithm 2 MELD LDA logic

```
Result: NotifyCCA
NotifyCCA ← TRUE;
 if Pktloss then
    S_TB ← ComputeTbsThresh(D_TBS, N_HARQ, T_PDU);
     Q_th ← ComputeQThresh(N_PRBs, D_CQI, N_HARQ);
     if (D_TBS ≥ S_TB) ∧ (S_TxQueue ≥ Q_th) then
        NotifyCCA ← 1;
         exit();
    end
    if (D_TBS < S_TB) then
        NotifyCCA ← 1;
         exit();
    else
        NotifyCCA ← 0; // Silently retransmit lost packets
    end
end
```

4.2.2 MELD algorithm

At a basic level, MELD is a loss notification algorithm that exploits RAN information through the RNIS API in order to decide whether a lost TCP/QUIC packet should be retransmitted silently or reported to the congestion controller (i.e., CCA). Therefore, this algorithm must be invoked at each time a packet loss is detected by the built-in loss detection mechanisms (e.g., 3 dup-ACKs, RTO, RACK, etc.).

When looking at Algorithm 2, it can be seen that our algorithm is conceptually based on the loss discrimination approach we previously exposed. Basically, S_{TB} is computed in accordance with Equation 4.1, by using the current downlink TBS (D_{TBS}), the number of HARQ processes (N_{HARQ}) and the size of the transport layer TCP/QUIC packet (T_{PDU}). Similarly, the calculation of Q_{th} is based on Equation 4.2. However, in this case we need first to find the maximum possible TBS value, so the $ComputeQThresh$ function first estimates this value by using the best possible downlink CQI (D_{CQI}) and the current number of allocated PRBs (N_{PRBs}). More specifically, it uses the MCS selected by the base station for the D_{CQI} (which is vendor-specific [97]) and the allocated PRBs to retrieve the corresponding TBS from the 3GPP standardized TBS mapping table (i.e., Table 7.1.7.2.1-1 in [1]). After that, it computes Q_{th} by just multiplying the retrieved TBS value by the number of HARQ processes. During the calculation of these values, the algorithm also records the length of the RLC queue allocated to the UE at the base station (S_{TxQueue}). Then, in case of loss, the congestion controller is notified when TBS is less than S_{TB} (which could result from a low CQI or a decrease in the number of allocated PRBs) and when the transmission queue at the base station exceeds Q_{th}. The latter condition limits unnecessary buffering, thus reducing the effect of bufferbloat on short and delay-sensitive flows. In all other cases (i.e., higher TBS together with acceptable level of buffering), the packet is silently retransmitted.

4.2.3 Design and Implementation

MELD is built on top of picoquic on the server side; the client does not require any modification. Similarly to the model described in [13, 45], the RNIS API is provided by the FlexRAN controller, which receives real-time RAN statistics from the FlexRAN agent every 8 millisecond through a dedicated control channel. The relevant RAN information is then retrieved by a Python process through the provided API, and published on an Advanced Message Queuing Protocol

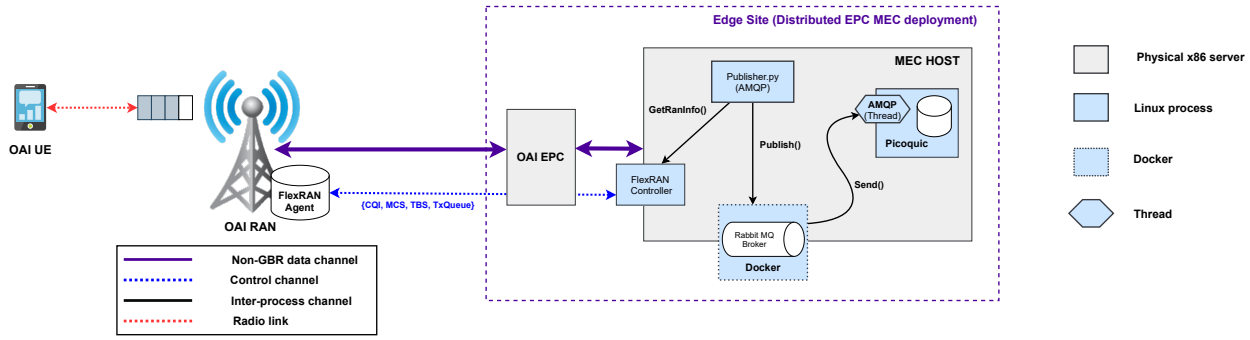


Figure 4.2 – MELD experimentation setup.

(AMQP) topic as illustrated in Figure 4.2. The picoquic server deployed at the MEC host then uses a separate thread to subscribe to the topic in order to receive real-time changes from the FlexRAN agent. Once the picoquic server receives the radio information (TBS, S_{TxQueue}, N_{HARQ} etc.) via AMQP, it computes the TBS threshold (S_{TB}) and the queue threshold (Q_{th}). Then in case of packet loss(es), it decides whether it should notify the CCA or just retransmit the lost packet(s) silently. This process is actually enforced by modifying the picoquic loss notification logic. Note that S_{TB} is the value of TBS that guarantees the transmission of one transport layer packet within HARQ RTT.

4.2.4 Overhead of exploiting radio information

Our approach based on the analysis of up-to-date radio information requires the FlexRAN agent (collocated with the eNodeB) to report RAN statistics to the controller (in the MEC host) after each CSI report (i.e., at least every 8 ms). The authors in [45] measured 100 Mbps network overhead for 50 UEs while using a polling period at TTI granularity (i.e, every ms). In our case, this overhead is reduced by a factor of 8 (thanks to a polling period of 8ms). Therefore, a single UE requires a control channel data rate of 250 kbps corresponding to 4.8% and 22.3% of the average goodput gain when MELD is used with NewReno and Cubic, respectively.

4.3 Experimentation and Results

To evaluate the performance of our approach, we deploy an edge delivery QUIC server that uses MELD as part of its loss detection/recovery mechanism. Table 4.1 describes the characteristics of our experimentation setup on the R2lab wireless testbed. We generate controlled interference in the RAN using an Additive White Gaussian Noise (AWGN) generator that introduces 0.8% random packet loss [30]. We analyze the introduced packet loss rate profile on continuous ICMP ping traffic sent at 560kbps, (10 runs of 120 seconds), see Figure 4.3. During all the following experiments, we use the same level of interference and evaluate the performance of a 20MB file download over the 15Mbps radio link.

We repeat each test 10 times in order to account for the variability of link capacity. We first evaluate the global goodput using a loss discrimination algorithm based only on TBS, as suggested by Equation 4.1 ; then we analyze the proposed MELD algorithm. Figures 4.4 and 4.6 show the

TABLE 4.1 – MELD/OAI Experimentation parameters

Parameters	Values
LTE Duplex mode	FDD
LTE Bandwidth	5 MHz
Max. number of PRBs	25
RLC Mode	Unacknowledged (UM)
RLC UM Buffer size	1 MB
Max TBS	2292 Bytes
Radio Link Capacity	15 Mbps
Max. # of HARQ retransmissions	4
Number of HARQ processes	8
End-to-End RTT	55 ms
Layer-4 Protocol	QUIC
Max. QUIC Datagram size + IPv4 hdr	1460 Bytes

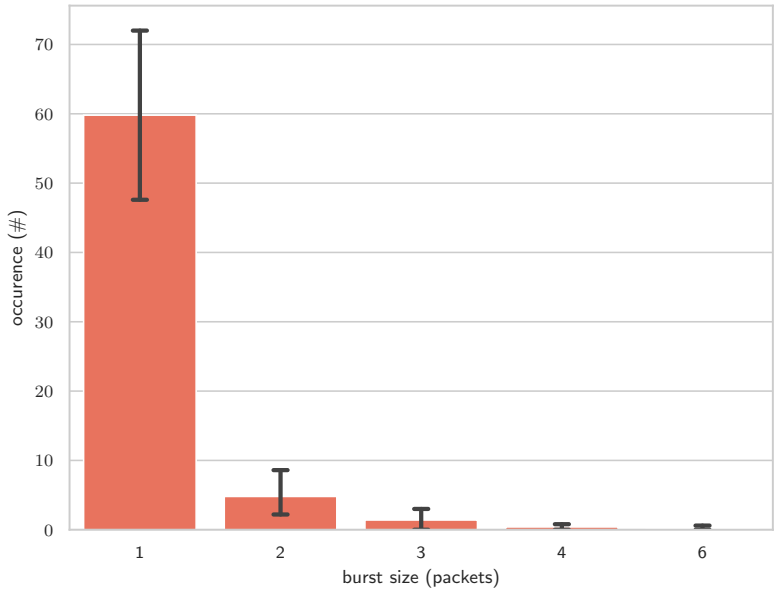


Figure 4.3 – Burst loss occurrences.

different test results with a 95% confidence interval. The source code of MELD as well as the steps to reproduce these experiments are publicly available in our git repository, at [116]

4.3.1 TBS-only loss discrimination

In this first experiment, we take the decision to notify lost packets to the CCA only based on the size of the transport block. In other words, here, the packet losses are discriminated only

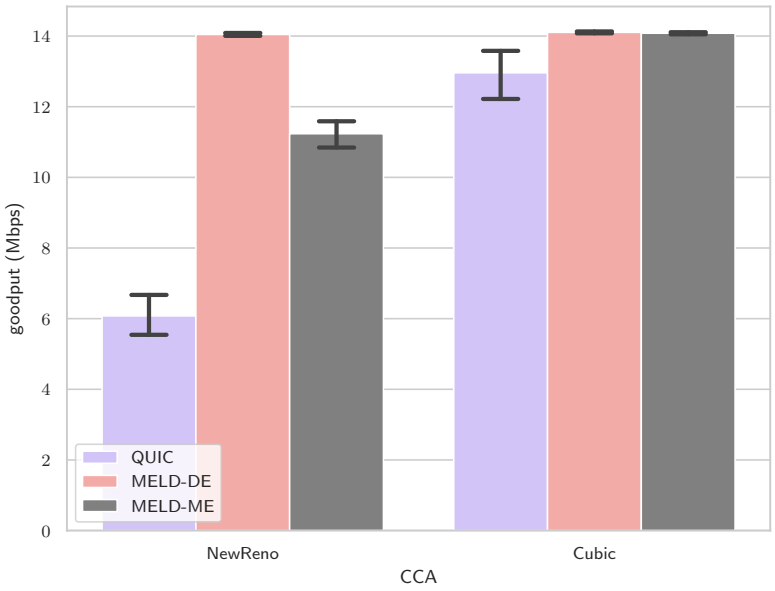


Figure 4.4 – CCA performance
with MELD-DE/ME.

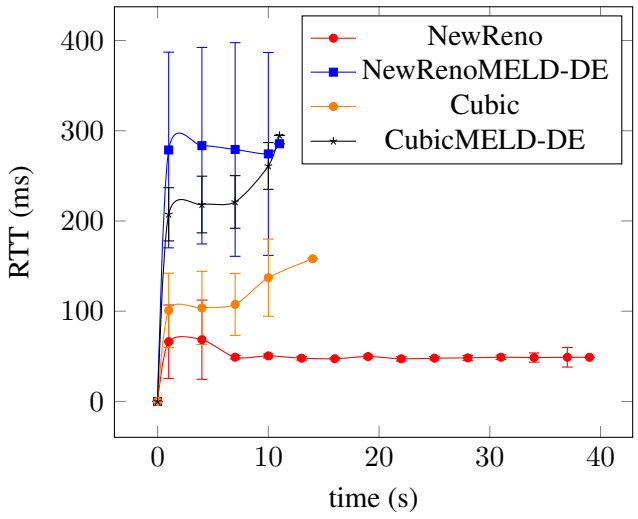


Figure 4.5 – RTT for MELD-DE
vs legacy QUIC.

on the basis of Equation 4.1. We refer to this first/intermediate variant of MELD as MELD-DE. We assume that the channel quality is very poor or that the UE is not allocated enough resources when the selected MCS corresponds to a TBS less than S_{TB}. Since N_{HARQ} is equal to 8 and the maximum QUIC datagram size (set to 1440 Bytes) encapsulated in a 20 Bytes IPv4 header is 1460 Bytes, then the transport blocks sizes (S_{TB}) must be at least greater than or equal to 183

Bytes (i.e., $1460/8$) in order to validate the conditions in Equation 4.1. In other words, the CCA is notified of the loss and congestion window reduction is applied whenever TBS is less than 180 bytes. The test results with NewReno using the TBS threshold show a significant performance gain. As illustrated in Figure 4.4, a 131% increase in goodput is observed, mostly due to the fact that the congestion window is not halved in case of random losses. On the other hand, the same experimentation with Cubic does not double the goodput. We only observe 9% performance gain over the legacy version (see Figure 4.4). Since Cubic already uses an aggressive growth function, the use of an LDA increases the link utilization to a near optimal value. The difference in download time can be observed in Figure 4.5. Results show that flows under MELD-DE have shorter duration because they exploit more bandwidth. However, regarding the RTT increase (see Figure 4.5), we can observe significant RTT increase mainly due to excessive buffering, which is normal since the conditions in Equation 4.1 do not consider the introduced degree of buffering when discriminating losses.

4.3.2 TBS and queue length-based loss discrimination

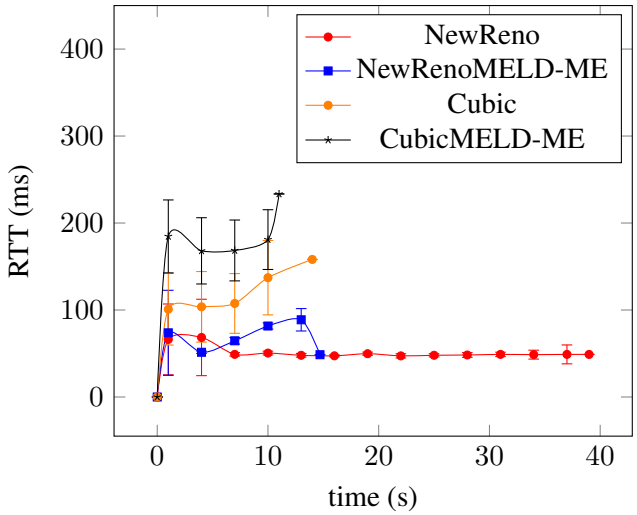


Figure 4.6 – RTT for MELD-ME vs legacy QUIC.

This experiment evaluates the complete MELD algorithm proposed in Section 4.2.2, in terms of average goodput and RTT increase. We refer to this second/complete variant of MELD as MELD-ME. The results show for New Reno over 80% increase in goodput (see Figure 4.4) when compared to legacy picoquic. As expected, taking queue length into account while discriminating losses reduces RTT increases but slightly affects the global throughput if the CCA in use is slow during the steady state. As illustrated in Figures 4.4 and 4.6, MELD-ME prevents large RTT increase (bufferbloat) for NewReno and Cubic at the cost of slightly lower goodput, especially for NewReno since it is too conservative during the congestion avoidance phase (or steady state). On the other hand, with Cubic, although a decrease in delay is observed (compared to Cubic with MELD-DE), the same 8% gain in goodput is conserved. Such a behavior was expected since Cubic's growth function is more aggressive than the traditional Additive Increase Multiplicative Decrease (AIMD) during the Congestion Avoidance phase (CA). As shown in the test results,

for both CCAs the RTT is kept under a lower value, which was not the case with MELD-DE. This behavior confirms the efficiency of Q_{th} in Equation 4.2, which allows MELD-ME to limit unnecessary buffering even when the UE experiences ideal radio conditions (i.e., with CQI=15).

As shown in the different results, MELD-DE outperforms MELD-ME in terms of goodput but also introduces a significant level of excessive buffering. On the other hand, MELD-ME exhibits slightly less goodput (only for NewReno) but enables a significant decrease in delay, thus limiting the effect of bufferbloat on eventual competing flows.

4.4 Summary

In this chapter we demonstrated that some information transmitted over LTE RAN such as PRB, CQI, or TBS can be exploited as relevant congestion signals at the transport layer. To the best of our knowledge, our proposed MELD scheme is the first loss discrimination algorithm for LTE that exploits such information to proactively discriminate packet losses while taking into account the overhead of the link layer mechanisms. Our performance evaluation showed a significant improvement for loss-based congestion controllers. Based on the presented results, we can conclude that MELD-DE can be used for sheer download scenarios where bufferbloat is not important or where there are no competing short- or delay-sensitive flows. Meanwhile, MELD-ME can be used in mixed scenarios (i.e., when a UE is downloading short and long flows at the same time), since it improves the goodput while minimizing the RTT increase. As such, it is more adapted for real-world use cases.

Also, it is important to note that our proposed MELD approach is also applicable to 5G networks since its resource allocation and lower layers mechanisms are similar to LTE's. Furthermore, since 5G New Radio is prone to high penetration losses in FR2 (Frequency Range 2, i.e., mmWave), large delay increase and packet losses are expected due to obstacles blocking the Line of Sight (LoS). Therefore, proper loss discrimination schemes are required in order to provide enhanced Mobile Broadband (eMBB) on top of loss-based CCAs.

With that being said, it is important to note that the use of MELD does not prevent a loss-based CCA from being greedy and aggressive. Indeed, MELD-ME undoubtedly decreases to some extent the degree of excessive buffering. However, since MELD is invoked only in case of packet losses, the CCA continues with its default greedy behavior in case of no loss (i.e., good radio conditions). Therefore, even if all the loss-based CCAs deployed on the Internet used MELD, the excessive buffering due to their greedy behavior would still persist. Also, considering the huge number of CCAs that are deployed on the Internet and the fact that the content providers are free to use any CCA of their choice, it becomes very difficult and almost impossible to solve the bufferbloat issue at the CCA level. In such a context, how can we still manage to mitigate the bufferbloat issue at the transport layer regardless of the negative effects and aggressive behaviors of existing CCAs ? The next chapter answers this question and introduces a new approach that can transparently control the aggressive behavior of active flows regardless of their CCAs.

CHAPTER 5

MEC-based approach for addressing bufferbloat and CCA fairness

In this chapter we first highlight the limitations of the bufferbloat mitigation solutions presented in chapter 2 and show why such solutions are not adapted for ensuring fairness and low delay in today's cellular networks where a single user can maintain in parallel multiple TCP flows that may use different CCAs or require different network conditions. We then proceed to introduce our new intelligent proxy approach, named RAPID, that leverages the Radio Network Information Service provided by the ETSI MEC framework as well as specific transport layer knowledge and capabilities in order to ensure fairness, low delay and near optimal link utilization regardless of the CCAs in use. Following, we present the design and implementation details of our solution and evaluate its performance first in a simulated environment and later in a real-world 4G testbed. Lastly we comment on the experimental results of our approach, discuss the observed limitations and propose some improvements for the future.

5.1 Motivation

As shown throughout the previous chapters, the use of per-user queues in the radio access network along with logical tunnels in the core network have almost eliminated the likelihood of flow interactions or CCAs interference between separate users [54] in current cellular networks. However, flows from the same user may still use different CCAs, and depending on the design and aggressiveness of the involved CCAs, a severe bufferbloat can be introduced given the presence of deep buffers in the access networks [66, 96]. For instance, any flow using a traditional loss-based CCA will naturally end up creating excessive buffering and penalize all the other concurrent flows that are delay-sensitive. And since data traffic is isolated on a per-user basis, this excessive buffering cannot propagate to another user in the same cell. In other words, a user can suffer only from a bufferbloat caused by its own flows, hence the term self-inflicted bufferbloat. In fact, the probability of having multiple concurrent flows on the same UE is quite high in today's cellular

networks, mostly due to the increasing number of mobile services which create more opportunities for parallel data transfer [87]. This probability becomes even higher when multiple devices are using the same mobile device's Internet connection (i.e., in case of tethering or 4G/5G routers). Furthermore, recent studies [82] show that, over 30% of the top 20,000 Alexa websites are using Cubic and roughly 18% are currently based on BBR. This study also reveals that delay-based TCP variants such as Vegas/Veno and other unknown variants are used on 2.8 and 17.6% of the evaluated websites, respectively. Based on these findings, it appears that the most majority of today's TCP traffic uses loss-based CCAs which are known to introduce excessive buffering in mobile networks (as discussed in Chapter 2). As a result, recent delay-aware or rate-based CCAs such as BBR and its variants which aim to reduce delay increase eventually end up suffering from excessive buffering when competing with other flows and become unable to grab their fair share of the bandwidth. In fact, given the ever-growing number of CCAs that could potentially compete for bandwidth, it is virtually impossible to solve the self-inflicted bufferbloat issue by just introducing a new CCA, since it cannot control the behavior of the concurrent CCAs. Also, it is worth noting that the self-inflicted bufferbloat issue becomes even more significant in mmWave bands mostly due to the inherent properties of high frequencies. According to 3GPP TS 38.104, 5G New Radio (NR) can operate in two distinct frequency ranges, Frequency Range 1 (i.e., FR1 or sub-6GHz) and Frequency Range 2 (i.e., FR2 or 24.25 GHz to 52.6 GHz) which falls in the mmWave region. In fact, frequencies in the millimeter wave region are prone to high penetration losses due to obstacles blocking the Line-of-Sight. Therefore, in case of NLOS conditions, the base station buffers incoming packets, which eventually leads to a high delay increase and/or a persistent self-inflicted bufferbloat [108].

Indeed, as presented in Chapter 2, some solutions aiming to prevent bufferbloat and/or maximize the radio link utilization already exist in the literature. However, as will be shown below, it appears that most of them are either limited to some scenarios or merely delay the occurrence of bufferbloat. For instance, when we look at cross-layer CCAs, it is clear that they undoubtedly improve the link utilization since they adapt their sending rate according to the actual radio conditions and bandwidth. However, they do not take into account the number of flows sharing this bandwidth in the user's device. Therefore, when multiple flows using cross-layer CCAs progress in parallel in the same device, each individual flow matches its sending rate with the speed of the radio bottleneck, which causes the radio capacity to be exceeded as many times as there are flows relying on a cross-layer CCA. In other words, the amount of buffered data is increased by one BDP for each additional flow that uses a cross-layer CCA, and this naturally ends up creating a self-inflicted bufferbloat situation. In fact, even if among the concurrent flows, there is only one flow that relies on a cross-layer CCA, excessive buffering may still occur, since some or all of the remaining flows might be using a loss-based CCA. Also, it is important to note that most cross-layer CCAs require client-side modifications [78, 107, 111] in order to include some radio information in the TCP acknowledgments sent to the server. As such, they may introduce computational overhead and additional power consumption in the UE.

Likewise, most existing in-network bufferbloat mitigation solutions also come with some drawbacks and also struggle to prevent excessive buffering in case of multiple flows per user : milliProxy, proposed in [94], significantly mitigates bufferbloat in case of single flow by transparently overriding the advertised receive window. However, it fails to prevent excessive buffering in case of multi-flow scenarios since its window computation scheme does not take into account the number of parallel flows progressing in the UE nor the characteristics and demand of each individual flow.

In-band throughput guidance introduced in [63], assigns an equal share of the radio bearer capacity to each flow in cases where multiple flows share the same bearer. However, since the concurrent flows may have different requirements and demands, such an approach is not sufficient to ensure fairness and definitely results in an underutilization of the radio link. For instance, some delay-sensitive or interactive flows may require just a small fraction of the total bandwidth (because they are app-limited flows) while some concurrent sheer download flows may need more bandwidth. The exact same limitations are applicable to NATCP (proposed in [10]) since it also divides the available radio bandwidth by the number of parallel flows in the UE regardless of their respective demands.

The multi-connectivity TCP proxy proposed in [76] for 5G mmWave networks, effectively improves the download time in some scenarios. However, it is not adapted for bufferbloat mitigation since it modifies the receive window based on a static buffer size that does not reflect the actual radio conditions, which could result in large delay increase, especially in the mmWave band.

The AQM-based solution in [60], when configured with the right RLC buffer sizes and the right CoDel parameters, can significantly reduce the delays experienced by low-latency flows that share the same DRB with a greedy TCP flow. However, it is important to note that CoDel indiscriminately drops packets when its target queue-sojourn time is exceeded. As a result, a considerable number of packets belonging to low-latency flows may be dropped because of a concurrent greedy TCP flow that is too aggressive or not very reactive to packet drops. Also, the real-world adoption of such a solution might take quite some time, since it requires the presence of QFI queues as well as a scheduler at the SDAP sub-layer, which is not yet the case in the related 3GPP specifications.

The DRQL, 5GBDP, and USP algorithms proposed in [61], also significantly reduce the bufferbloat effect on low-latency flows in some scenarios. However, like the previous solution, they require modifications at the SDAP sub-layer. Additionally, to work effectively in certain scenarios, these algorithms need to be combined together appropriately and with other queuing disciplines such as Stochastic Fair Queuing (SFQ) and/or CoDel. For instance, in cases where multiple flows share the same QFI, the best delay reduction for a delay-sensitive flow is obtained by using SFQ and USP at the UPF combined with 5G-BDP at the SDAP sub-layer. Basically, in this scenario, USP avoids congesting the QFI buffers at the SDAP sub-layer by controlling the egress rate of the UPF based on SDAP occupancy ; SFQ fairly serves the flows with the same QFI in a round-robin fashion so that the egress capacity is fairly shared among them ; and 5G-BDP allows to pace data between SDAP and RLC, based on the MAC actual egress rate (i.e., actual radio capacity). It is clear that such a combination undoubtedly benefits delay-sensitive flows regardless of the aggressiveness of any concurrent flow. However, due to the use of a TTI-level pacing between the UPF and the access network (i.e., USP), it is worth noting that this solution is applicable only for cases where the one-way delay between the UPF and the access network is around one TTI.

Thus, in light of the lessons learned from the above approaches, we set out to find a solution that can work in more scenarios and that requires no modifications to the user's device, to the server, or to the 4G/5G protocol stack. With that in mind, we design and evaluate a transparent proxy approach, which collects up-to-date information about the radio access bottleneck via the Radio Network Information API (RNI API) exposed by the MEC host, thus completely avoiding any client-side or server-side modifications. It then combines the collected radio information together with specific TCP metrics and capabilities in order to :

- Determine the behavior and requirements of each individual TCP flow (coming from the same UE) in terms of bandwidth consumption ;

- Transparently control the sending rate of each TCP flow, regardless of their CCAs, in such away that excessive buffering is prevented for all flows without dropping any packets from any flows ;
- Make sure the radio bandwidth allocated to the UE is appropriately distributed among its active flows.

We named this approach RAPID which stands for "RAN-aware Proxy-based Flow Control for High Throughput and Low Delay flows". The details about its design and the mechanisms it uses in order to provide the attributes listed above, as well as other important properties, are given in the next section. Aside from its design and mechanisms, it is also important to emphasize that, at a high level, RAPID is based on the principle that excessive buffering should never occur in the first place if the aggregate sending rate of the competing flows correctly matches the bottleneck's actual delivery rate. So, instead of dealing with excessive buffering once it occurs, as is done in traditional approaches, RAPID rather prevents the competing flows from causing excessive buffering in the first place, by controlling their sending rate appropriately.

5.2 RAPID : a RAN-aware Performance Enhancing Proxy for both high throughput and low delay flows

As briefly introduced earlier, RAPID is our solution for preventing self-inflicting bufferbloat. Basically, it is a RAN-aware performance enhancing proxy that intercepts TCP connections and distributes proportionally the available RAN bandwidth among the active flows. It relies on the RNIS API and on packet arrival rates to estimate the aggregated RAN bandwidth and categorize the concurrent TCP flows in the UE. This approach does not involve the end user unlike existing cross-layer CCAs that introduce computational overhead and additional power consumption at the UE as they require client-side modifications.

RAPID's main originality with respect to the solutions presented in Section 2 is twofold. First, it mitigates self-inflicted bufferbloat while preserving high throughput regardless of the characteristics of the CCAs in use or the number of concurrent flows sharing the per-user queues. Second, it brings a demand-aware bandwidth allocation scheme, which makes it possible to dynamically allocate bandwidth to each individual flow based on their behavior and estimated demand. In other words, it enforces a demand-aware fairness principle among the different flows competing in the user's device.

5.2.1 Proxy architecture

RAPID is a TCP proxy that seats close to the base station, in the MEC host, and that leverages up-to-date radio information exposed by the RNIS service [39], similar to RAVEN [99] and MELD (proposed in our previous work [36]). As illustrated in Figure 5.1, RAPID relies on several modules organized in layers, meaning that each module provides services to its upper module.

At a basic level, RAPID first splits each incoming end-to-end TCP flow from a UE into two separate connections, referred here as TCP_{RAN} (i.e., TCP connection between the UE and RAPID) and TCP_{WAN} (TCP connection between RAPID and the corresponding end server). After that, it invokes the per-flow bandwidth estimation module in order to compute the Bandwidth Delay Product (BDP) available to each intercepted flow. Finally, the computed BDPs are used to override

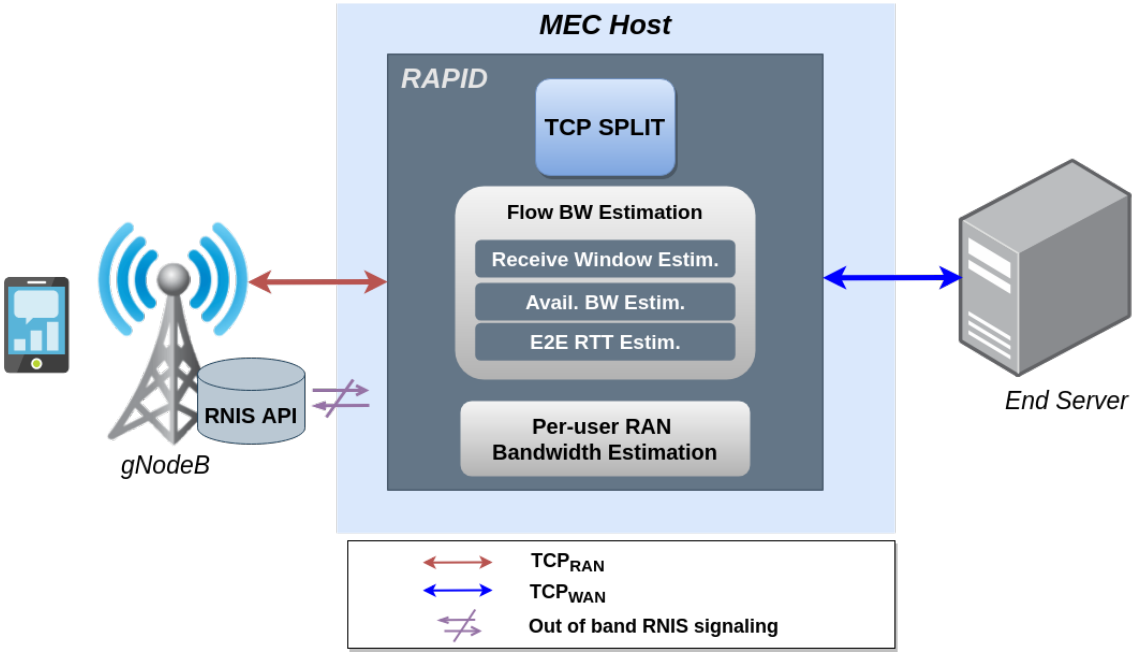


Figure 5.1 – RAPID high-level architecture

the Receive Window (RW) value in the TCP acknowledgments towards the corresponding end servers, thereby limiting the bytes in flight from those servers to the BDPs estimated for their respective flows. It should be noted that while RAPID has no control over the choice of the end-server CCA, it can still control all the parameters of the connections established in the RAN segment (i.e., TCP_{RAN}). Therefore, by default and in order to prevent excessive buffering in the proxy, TCP_{RAN} relies on a simplified CCA that always sets its congestion window to the estimated flow BDP.

5.2.2 RAN bandwidth estimation

As illustrated in Figure 5.1, RAPID periodically receives RAN statistics through the RNIS service. Similar to the statistics reported by FlexRAN agents [45], these statistics include the cell bandwidth, the sub-carrier spacing, the Radio Network Temporary Identifier (RNTI), the number of allocated Physical Resource Blocks (PRBs), the computed Transport Block Size (TBS) and the selected Modulation and Coding Scheme (MCS) information for all the UEs connected to the cell. This information allows RAPID to estimate the bandwidth allocated to each UE by the base station in units of PRBs. However, the number of allocated PRBs does not always reflect the achievable bandwidth since the MAC scheduler at the base station may also take into account the RLC buffer state [84] or the incoming data rate [107] when allocating resources to users. Therefore, in order to estimate the achievable bandwidth, we first calculate the number of unused PRBs, $Pu_{\text{i}}(t)$, by using the total number of PRBs in the cell N_{PRB}, the number of connected UE M and the number of PRBs allocated to each UE $Pa_{\text{i}}(t)$:

$$Pu_{\text{i}}(t) = N_{\text{PRB}} - \sum_{i=1}^{M} Pa_{\text{i}}(t)\,. \tag{5.1}$$

The expected number of PRBs, $Pe_{\text{i}}(t)$ for a given UE i is then computed as follows :

$$Pe_{\text{i}}(t) = Pa_{\text{i}}(t) + Pu_{\text{i}}(t)\,. \tag{5.2}$$

With the expected number of PRBs and the selected MCS values, RAPID computes the maximum transport block size at time t, denoted $TBS_{\text{i}}(t)$, for each UE by either using the tables in 3GPP TS 36.213 [3] or the indications in 3GPP TS 38.214 [5]. The former is used for 4G while the latter is used for 5G. For 4G, we first use the MCS value (which is a direct mapping of the Channel Quality Indicator reported by the UE) to retrieve the corresponding TBS index (also known as I_{TBS}) from the MCS and I_{TBS} mapping table in [3]. Then the expected number of PRBs together with the I_{TBS} are mapped to the corresponding TBS value using the TBS mapping table (i.e., Table-7.1.7.2.1-1 in [3]). On the other hand, in case of 5G, the theoretical number of information bits that can be transmitted is computed by multiplying the number of sub-carriers in the allocated PRBs (e.g., 12 sub-carriers per PRB in case of 15 kHz sub-carrier spacing) by the modulation order (deduced from the MCS table in [5]) and by the coding rate (also deduced from the MCS table). If the resulting value is inferior or equal to the standardized 3824 bits [5] (which is chosen based on the maximum code block size that can be handled by 5G's channel coding technique without segmentation), then the corresponding TBS is directly fetched from the TBS table in [5] (i.e. Table-5.1.3.2-1), otherwise, the TBS is determined thanks to the standardized 3GPP formula given in [5], which takes into account the channel coding overhead in case of segmentation. In any case, once the necessary mappings are done and the TBS value that corresponds to the expected PRB combined with the reported MCS is found, the highest achievable throughput for a given UE denoted $C_{\text{i}}(t)$ (in units of bits/s) can be expressed as follows :

$$C_{\text{i}}(t) = \frac{TBS_{\text{i}}(t)}{TTI}\,. \tag{5.3}$$

where TTI is the Transmission Time Interval (e.g., TTI = 125 μs for Numerology μ=3). It is worth noting that this real-time RAN bandwidth estimation makes it possible to detect fast capacity variations as well as bad radio conditions (e.g., NLOS conditions in mmWave), which, to some degree, can cause excessive buffering at the base stations.

5.2.3 Per-flow bandwidth allocation via intelligent and transparent TCP flow control

The major contribution of RAPID with respect to other existing solutions is its bandwidth allocation scheme. Rather than relying on a static allocation scheme, the bandwidth estimation module distributes the aggregated bandwidth of the UE according to the requirements of the active flows in terms of bandwidth. At the beginning of the connection, the initial Receive Window denoted $RW_{\text{ij}}(t)$ of a given flow j, from a UE i, is computed by dividing the expected bandwidth estimated for the UE (in $bits/s$) by the number of active flows :

$$RW_{\text{ij}}(t) = \frac{C_{\text{i}}(t)}{N}\, RTTmin_{\text{ij}}(t)\,. \tag{5.4}$$

where N is the number of active flows in the UE and $RTTmin_{\text{ij}}(t)$ is the minimum RTT of the j^{th} flow on the wired segment (i.e., from RAPID to the original end server) at time t. This initial phase is followed by a dynamic allocation phase after $K_{\text{ij}}\, RTT_{\text{WAN}}$, where the allocated bandwidth is changed depending on the category of the flow. We define two distinct categories of flows :

slow interactive flows (i.e., browsing, instant messaging, etc.) and fast download flows (e.g., HD streaming, file transfer, Augmented and Virtual Reality AR/VR, etc.). The former are sensitive to delay but require low bandwidth, while the latter may require both high bandwidth and reasonable delay variations. In such a context, allocating the same amount of bandwidth to all flows not only penalizes fast download flows (since slow flows only require a small fraction of the bandwidth), but also results in radio link under-utilization and wastage of the bandwidth allocated to the UE. With that in mind, we devise a flow categorization scheme based on a common behavior of TCP flows during the startup phase. In fact, since the congestion windows of most TCP flows follow an exponential increase pattern during the initial phase, we can directly compute the congestion window value at different RTTs as follows :

$$cwnd = \begin{cases} MSS * 2^0 & \text{at the } 1^{st} RTT \\ MSS * 2^1 & \text{at the } 2^{nd} RTT \\ MSS * 2^{(K-1)} & \text{at the } K^{th} RTT\,, \end{cases} \tag{5.5}$$

where MSS is the negotiated Maximum Segment Size. From (5.5), by replacing $cwnd$ by RW, we can deduce the number of RTTs it takes to reach a congestion window value equivalent to the receive window :

$$RW = MSS * 2^{(K-1)} \implies K = \log_2\left(\frac{RW}{MSS}\right) + 1\,. \tag{5.6}$$

For the j^{th} flow of the i^{th} UE and at the instant t, Equation 5.6 becomes :

$$K_{\text{ij}}(t) = \log_2\left(\frac{RW_{\text{ij}}(t)}{MSS}\right) + 1\,. \tag{5.7}$$

where K_{ij} is the number of RTTs that the j^{th} flow of i^{th} UE takes in order to reach an amount of bytes in flight equivalent to the receive window RW_{ij}.
Additionally, since we know that the flow is going to reach the receive window RW_{ij} after K_{ij} RTTs (i.e., at instant $t + K_{\text{ij}}$), we can also express the arrival rate or incoming throughput the flow is expected to achieve at that time as :

$$R_{\text{ij}}(t + K_{\text{ij}}) = \frac{RW_{\text{ij}}(t)}{RTT_{\text{WANij}}}\,. \tag{5.8}$$

where RTT_{WANij} is the current smoothed RTT of the j^{th} flow on the wired segment (i.e., between the proxy and the end server.

At this point, with the information at our disposal, we can analyze the behavior of the flow and assign it to a predefined category. Basically, our categorization is based on the assumption that a flow that requires a large bandwidth would follow an exponential increase and fully consume the allocated bandwidth by the end of the K_{ij}^{th} RTT, while an app-limited or interactive flow might not. We refer to the period of time between the last modification of the receive window and the end of the $(K_{\text{ij}})^{th}$ RTT as the categorization period. To put it another way, at the end of each categorization period, the actual incoming throughput or arrival rate (noted $r_{\text{ij}}(t + K_{\text{ij}})$) of a flow that has fully consumed its allocated receive window should match its expected arrival rate ($R_{\text{ij}}(t + K_{\text{ij}})$, given in Equation 5.8). In this case, the flow is categorized as a "fast download flow" because it consumed the allocated bandwidth as expected. On the other hand, if the actual arrival

rate of the flow at the end of the categorization period is less than the expected value, the flow is considered as a "slow flow", meaning that it does not require/need all its allocated bandwidth. However, such a strict threshold might cause some fast flows to be wrongly flagged as slow, since it does not take into account the time it takes to move the packets from the Network Interface Card (NIC) to the TCP buffer, or the fact that some flows might leave the slow start phase prematurely due to the Hybrid slow start algorithm (HyStart). For all these reasons, we decided to use a more conservative categorization threshold by considering a flow as slow only if its actual arrival rate is less than a certain percentage (set empirically to 80%) of its expected arrival rate.

In any case, a flow categorized as slow strongly indicates that the previously allocated bandwidth was too high compared to the flow actual demands. Therefore, for this type of flow, we set the receive window to a value that reflects the flow's actual demands. For a given flow, the demand or required bandwidth is computed based on the observed arrival rates during the categorization period. Technically speaking, at each RTT, a sample of the arrival rate is taken and used to compute an exponential weighted moving average value, following an inverted version of the smoothed RTT calculation method in TCP. In fact, in our case, the weights are inverted in order to strongly reflect the increase in throughput over time. The rationale behind this choice is the fact that most CCAs keep increasing their congestion window or sending rate as long as the bottleneck pipe is not full but not the other way around. Thus, following our method, the actual demand of the flow is computed at each RTT until the end of the categorization period using the following expression :

$$\hat{r}_{\text{ij}}(t) = \alpha * \hat{r}_{\text{ij}}(t-1) + (1-\alpha) * r_{\text{ij}}(t)\,. \tag{5.9}$$

where $\alpha = 1/8$ (as in RFC 6298), $\hat{r}_{\text{ij}}(t)$ and $\hat{r}_{\text{ij}}(t-1)$ are the exponential weighted moving average of the arrival rate calculated during the current RTT and during the previous RTT, respectively.
Once the demand or the required bandwidth of the slow flow is captured thanks to Equation 5.9, we decrease its bandwidth allocation, i.e., its receive window in order to match its demand. This is done by multiplying the previous receive window by the ratio of the captured smoothed arrival rate over the expected arrival rate. In other words, the previous receive window is multiplied by a value that roughly indicates the percentage of consumed bandwidth. As a result, we obtain a receive window value that is proportional to the captured demand or smoothed arrival rate. This computation is performed using the following equation :

$$RW_{\text{ij}}(t+K_{\text{ij}}) = \frac{\hat{r}_{\text{ij}}(t+K_{\text{ij}})}{R_{\text{ij}}(t+K_{\text{ij}})}\, RW_{\text{ij}}(t)\,. \tag{5.10}$$

where $\hat{r}_{\text{ij}}(t+K_{\text{ij}})$ is the exponential weighted moving average of the arrival rates at time $(t+K_{\text{ij}})$. It is also important to note that the use of the exponential moving average of all the past arrival rates also allows to capture irregular throughput variations during the categorization period (i.e., during K RTTs), even though less importance is accorded to very old throughput samples in our case. After each change of receive window, a certain amount of bandwidth is generally released, especially when the change is made for a slow flow. So, in order to fully exploit the radio capacity and increase the link utilization, the released bandwidth must be reallocated to fast flows, which need more bandwidth, unlike slow flows. For that, it is important to keep track of the released bandwidth both at flow level and at UE level. Basically, every time a flow is categorized as slow, the released bandwidth at flow level, denoted A_{ij}, is recorded by computing the difference between the previous and the new receive window. Then the available bandwidth at UE level, denoted A_{ij}, is immediately updated by computing the sum of all the released bandwidth from the slow flows

belonging to the UE. The following two equations are used to compute the unused bandwidth at flow level, and at UE level, respectively :

$$A_{ij}(t + K_{ij}) = |RW_{ij}(t) - RW_{ij}(t + K_{ij})| . \tag{5.11}$$

$$A_{i}(t + K_{ij}) = \sum_{j=1}^{Z} A_{ij}(t + K_{ij}) . \tag{5.12}$$

where A_{ij} is the available (unused) bandwidth at the flow level, A_i is the aggregated unused bandwidth at the UE level and Z is the number of identified slow flows. After these two operations, the available unused bandwidth is evenly distributed among the fast download flows of the UE. So, the receive window of each fast flow is immediately increased with an equal fraction of the available bandwidth as follows :

$$RW_{ij}(t + K_{ij}) = RW_{ij}(t) + \frac{A_i(t + K_{ij})}{N - Z} . \tag{5.13}$$

Equation 5.13 is also used whenever a flow is categorized as fast. It is important to note that, in this equation, unlike for slow flows, the $RW_{ij}(t)$ value is always recomputed as indicated in Equation 5.4.

For a given UE, the presented dynamic categorization and allocation operations are repeated in an infinite loop for each intercepted flow, so that the bandwidth allocated to this UE by the base station can be better distributed between its active flows, which naturally leads to a more refined radio link utilization. Also, it is important to note that, even if a fast flow is wrongly classified as a slow flow, it is still allocated a bandwidth that is proportional to its average arrival rate. Therefore, unless the flow has very irregular arrival rates, the misclassification should be automatically corrected in the next categorization round with almost no consequences on the flow's global performance. This is explained by the fact that, if a misclassified flow is really a fast flow, this flow is very likely to reach at least 80% of its average rate at the end of the subsequent rounds, unless its arrival rates are very irregular, in which case it may take more than one round to correctly solve the misclassification. Indeed, with this method, the allocated radio bandwidth is adequately distributed among the concurrent flows regardless of the CCA in use. However, it is important to note that, even when they are provided with the same receive window values, CCAs with more aggressive growth functions yield higher link utilization than CCAs with very conservative growth since the former reach the target Receive Window faster.

5.2.4 RAPID's demand-aware fairness

The degree of fairness between two CCAs is typically measured by computing their Jain's Fairness Index (JFI) [64], which gives a number between 0 and 1. A JFI of 1 (i.e., the highest degree of fairness) indicates that the two CCAs equally share the bottleneck bandwidth. However, equally sharing the bandwidth does not necessarily lead to real fairness since some CCAs, because of their design, may not be able to fully exploit their fair share, hence penalizing those with higher demands or potential to grow. Furthermore, as indicated in [104], JFI is known for being demand unaware, in the sense that it prohibits any flow from claiming the available bandwidth not used by other competing CCAs (e.g., because of their low demand or design). Therefore, in order to avoid these limitations, we design RAPID with a demand-aware bandwidth allocation goal in mind.

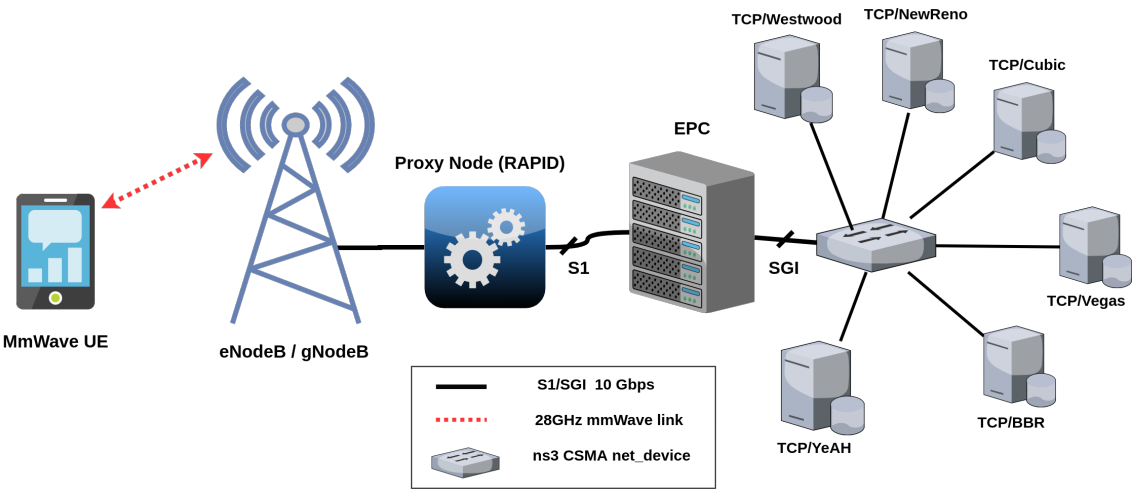


Figure 5.2 – RAPID ns-3 experimentation testbed

Basically, RAPID continuously monitors the demand of a flow by observing the arrival rate of the incoming packets (from this flow) through periodic categorization periods. The flow is then assigned a bandwidth that is proportional to the captured demand, but at the same time bounded by the available bandwidth and the number of concurrent flows. Upon the next categorization period, the flow is given a new opportunity to grow if the captured demand is not inferior to the previous one. Through this repetitive process, RAPID periodically gives each flow an opportunity to grow depending on its demand and allows sharing the unused bandwidth among the flows with high demands, unlike an equal-rate or JFI-based bandwidth allocation scheme.

For example, let us consider two concurrent flows sharing a 15 Mbps bottleneck link, the first one using Cubic, the second one using a dummy CCA that always sends 2 MSS per RTT. In such a configuration, although it is obvious that the second flow would greatly under-utilize the link, an equal-rate or JFI-based bandwidth allocation scheme would still allocate an equal share of the bandwidth to the two flows. In other words, the first flow would be penalized because of the impairments in the dummy CCA's design. In contrast, RAPID's demand-aware bandwidth allocation scheme would allocate just a small fraction of the available bandwidth to the dummy CCA. All the remaining bandwidth would then be used by the first flow since it has higher demands.

5.3 Evaluating RAPID with NS-3

In the following, we first describe the implementation of our proxy in ns-3 [55]. Then, we evaluate its efficiency under some relevant scenarios that reproduce the self-inflicted bufferbloat issues mentioned in Section 5.1. We repeat each scenario 10 times and show the observed results with a 95% confidence interval.

5.3.1 NS-3 experimentation testbed

The ns-3 mmWave module paves the way for the simulation of end-to-end 5G environments. Besides providing large bandwidths in the millimeter wave band for eMBB scenarios, it also enables the simulation of low-latency communications thanks to a customizable sub-carrier spacing

TABLE 5.1 – Global ns3 simulation parameters

Parameters	Values
Carrier Freq.	28 GHz
Bandwidth	200 MHz
Numerology	3
RLC Mode	Ack. (AM)
gNB Height	10 m
UE Height	1.5 m
RLC Buffer	10 MB
MSS	1440 B
Initial Window	10 MSS
RAN Throughput	850 Mbps
Propagation Model	3GPP Urban-Micro (UMI) ns-3 PropagationLossModel
3GPP Channel Scenario	UMI-Street Canyon

(i.e., supporting various numerologies) in the RAN. With that in mind and in order to evaluate the performance of RAPID, we implement all the functional modules illustrated in Figure 5.1 in a proxy device collocated with a mmWave base station in ns-3. The proxy is connected to the core network via a 10 Gbps link in order to simulate a real-world fiber backhaul. Since Cubic and BBR are the dominant CCAs today [81], we simulate two TCP sources based on these algorithms, as illustrated in Figure 5.2. Other simulation parameters are detailed in Tables 5.3 and 5.2.

TABLE 5.2 – Mobility and blockage parameters

Parameters	Values
User speed	1 m/s
Propagation Model	mmWave3GPP Buildings PropagationLossModel
Number of buildings	8
Building sizes†	Av. Jean Médecin, Nice, France

5.3.2 MEC scenarios

In this thesis, we focus on evaluating the performance of RAPID in mobile edge settings since it is more suitable for throughput-intensive ultra-low latency applications (e.g., AR/VR, tactile Internet, etc.) that require both high throughput (e.g., from 100 Mbps to a few Gbps) and low latency (e.g., from 1 to 10 ms) [73]. Furthermore, although several studies have been conducted on the evaluation of recent CCAs in 4G and 5G mmWave networks, to the best of our knowledge, there is no significant work on their evaluation in mobile edge and very low latency settings.

†. Google Earth™ is used to estimate the size of buildings.

Therefore, in our different experiments, we consider only end-to-end RTT in the range of 1 to 10 ms.

In order to evaluate the performance of RAPID in different RTT and flow configurations, we consider a main single user scenario consisting of one UE receiving simultaneously several flows from different end servers. Depending on the characteristics of the concurrent flows (i.e., sheer download or slow/interactive downloads), such a basic scenario can highlight both the self-inflicted bufferbloat issue introduced by loss-based CCAs and the unfairness in terms of bandwidth when different CCAs compete in deep buffer environments. Based on our main scenario, we define two sub-scenarios that highlight the impact of RAPID's flow-level bandwidth allocation scheme on the global achievable performance in terms of goodput and delay :

Fast eMBB downloads : ‡This first scenario aims to showcase the efficiency of RAPID in mitigating the interference between different CCAs (i.e., Cubic and BBR). To that end, the UE downloads simultaneously 200 MB from two TCP servers, one using Cubic and the other BBR, under both LOS and NLOS conditions. The obtained goodput and the delay variations are observed for each flow in various end-to-end RTT configurations.

Mixed Fast and App-limited/Slow downloads : This scenario highlights the negative effects of self-inflicted bufferbloat on app-limited flows (i.e., slow flows). The UE receives simultaneously a continuous flow (e.g., streaming) that uses Cubic and a slow flow (e.g., web browsing) that uses BBR. The slow flow is paced at 16 Mbps in order to simulate a continuous web browsing activity with 2 MB page sizes and a Page Load Time (PLT) of 1 second, which corresponds to the limit of user's real-time perception [85]. We then evaluate the effect of bufferbloat on web browsing for different RTT values and radio conditions by monitoring the bandwidth share of the slow flow, which should remain around 16 Mbps for a reasonable real-time perception.

Note that the NLOS conditions during our experiments are simulated by reproducing the positions, sizes, and heights of the buildings in Jean Medecin Avenue (a famous busy avenue in Nice, France).

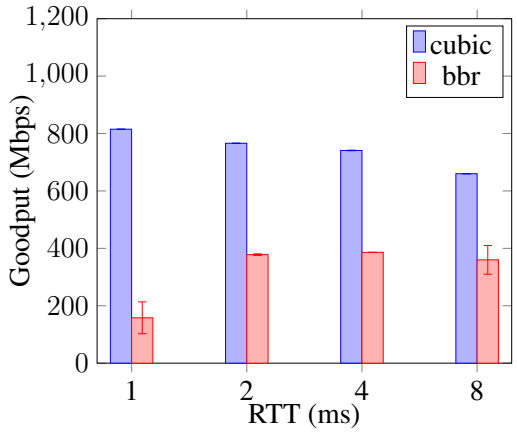
Also, it is worth noting that the reason we decided to stick with a single user is simply because evaluating RAPID in a multiple-user scenario would not bring any additional information about its performance, since the users are already isolated from each other by the base station. Instead, such a scenario would just show how efficient is the base station in distributing the available radio resources among the users.

5.3.3 Simulation results

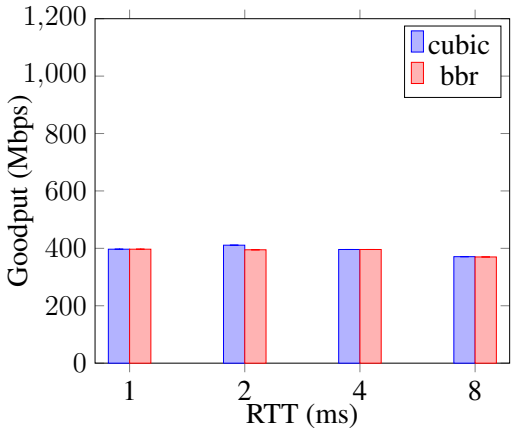
Figures 5.3(a), 5.3(c), 5.3(e), 5.3(g) show a comparison of both goodput and end-to-end latency when RAPID is not used in the fast eMBB downloads scenario. At first, it can be seen that Cubic grabs more bandwidth than BBR in all the evaluated configurations regardless of the radio conditions. This outcome was indeed expected since several studies show that loss-based CCAs outperform BBR in terms of goodput in deep buffer environments [24, 56, 102]. Although most of these studies claim that the degree of fairness in this situation only depends on the bottleneck buffer size, our results show that the end-to-end RTT value greatly affects the bandwidth shares of the two CCAs in mobile edge settings. As illustrated in Figures 5.3(a) and 5.3(c), the bandwidth share of BBR appears to increase as the end-to-end RTT increases, while Cubic follows an opposite pattern. This phenomenon is simply due to the Hystart and startup phases of Cubic and BBR, respectively. Basically, the two CCAs always start with an exponential growth ; however, in

‡. The average web page size on the Internet was around 1.5 MB [68] in 2013.

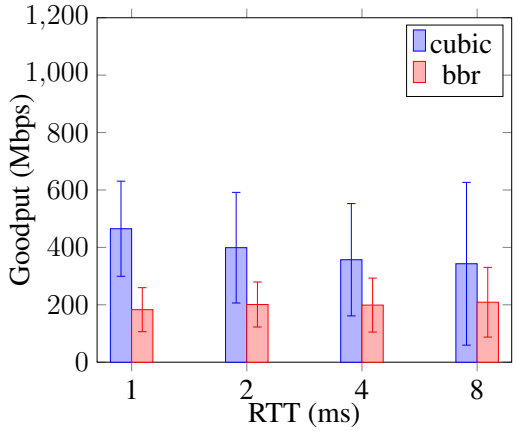
case of very small RTT (e.g., 1 ms), the BDP of the link is quickly reached. For instance, with the following parameters : initial congestion window = 10 MSS ; RAN throughput = 850 Mbps and RTT = 1 ms, the bytes in flight for each flow exceed the BDP of the link just after 4 RTTs (i.e., more than 2 BDPs of inflight data is maintained). However, since BBR's bottleneck bandwidth estimation window is around six to ten RTTs by default [24], it greatly underestimates the available bandwidth due to the large accumulation of data in the RLC buffer introduced by Cubic (more than 5 BDPs of inflight data at 10 RTTs). On the other hand, as the end-to-end RTT increases, it takes more RTTs to reach the BDP of the link, which allows BBR to get a better estimate of the bottleneck bandwidth (i.e., before Cubic introduces large buffering) but causes Cubic to leave the exponential increase phase prematurely. In fact, Cubic's Hystart exits the slow start phase after a certain increase in RTT [53], meanwhile, BBR's startup phase continues until it introduces more than two BDPs of inflight data [32]. Due to RTT increase, if BBR does not observe a lower minimum RTT (RTTmin) during a 10 seconds window, it enters a phase called probeRTT. In this phase, the number of bytes in flight is reduced to 4 Maximum Segment Size (MSS) at least for 200 ms in order to capture the new RTTmin. During this period, Cubic continues to grab more bandwidth as BBR decreases its sending rate, which explains why loss-based CCAs eventually outperform BBR in deep buffer environments.



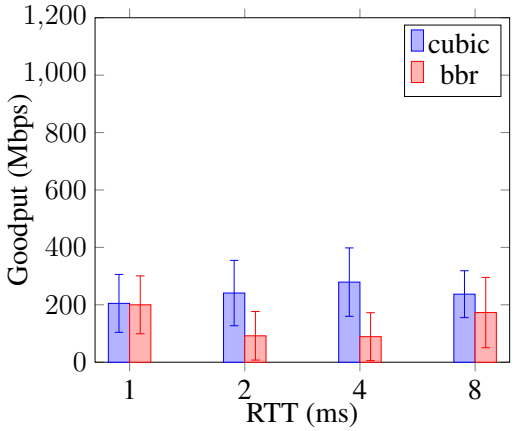
(a) Goodput in LOS without RAPID



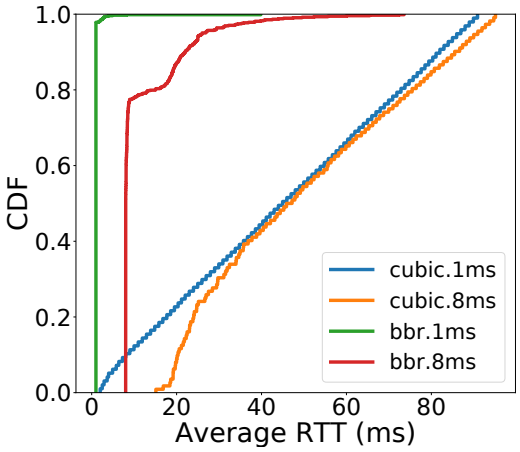
(b) Goodput in LOS with RAPID



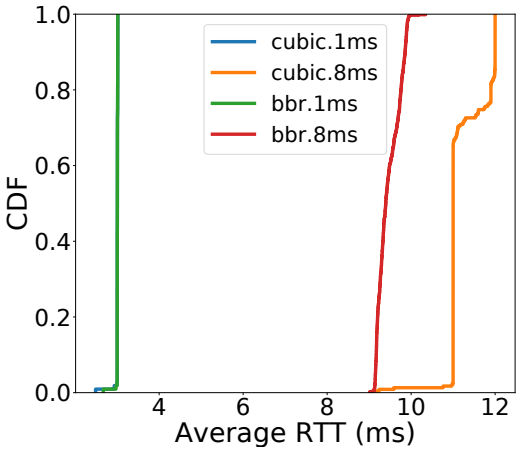
(c) Goodput in NLOS without RAPID



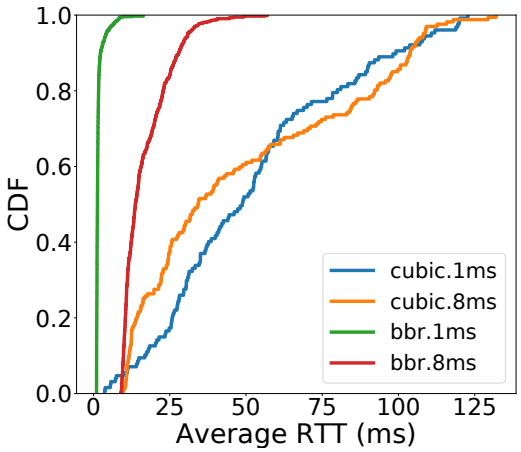
(d) Goodput in NLOS with RAPID



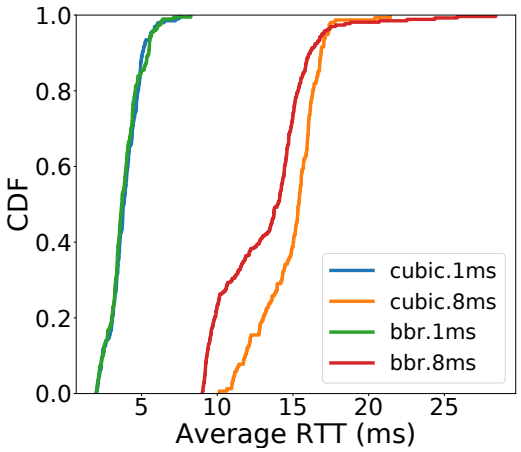
(e) RTT increase in LOS without RAPID



(f) RTT increase in LOS with RAPID



(g) RTT increase in NLOS without RAPID

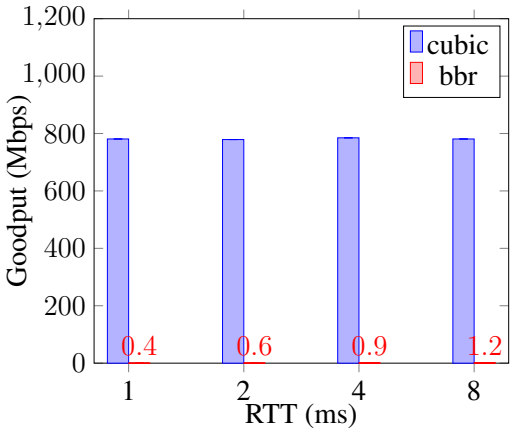


(h) RTT increase in NLOS with RAPID

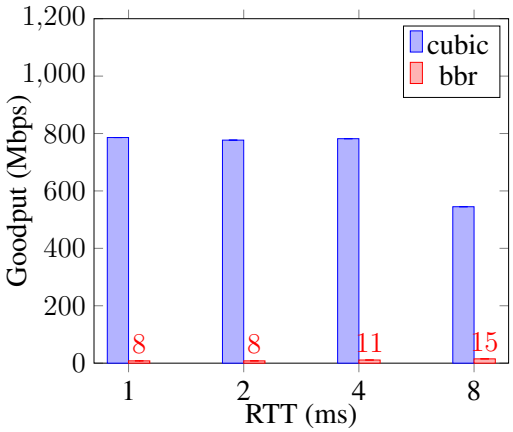
Figure 5.3 – Fast eMBB downloads scenario

However, for short to medium flows that finish before BBR's probeRTT phase takes place (i.e., for download duration around 10 s), as we increase the end-to-end RTT, BBR bandwidth share increases (even in very deep buffer environments) while Cubic share decreases due to the Hystart behavior. It can be seen from Figures 5.3(e) and 5.3(g) that excessive buffering causes a significant increase in RTT (from 100 to more than 5000%) for both BBR and Cubic, especially in the case of NLOS conditions. Furthermore, it is important to note that if the bandwidth is shared equally between the two flows, they should complete around the same time since they are downloading the same amount of data (i.e., 200 MB). In other words, the combined download time should be equivalent to the time it takes to download 400 MB over a 850 Mbps link, which is normally around 4 seconds. However, Figure 5.3(a) clearly shows that the Cubic flow always completes earlier (since it exhibits higher goodput) while the BBR flow takes much longer. For instance, in the 1 ms RTT configuration, the BBR flow takes at least 8 seconds in order to download 200 MB, thereby doubling the combined download duration of the two flows (i.e., from 4 to 8 seconds). Our subsequent experiments demonstrate that RAPID avoids this issue, allowing for a combined download time of 4 seconds instead of 8 seconds.

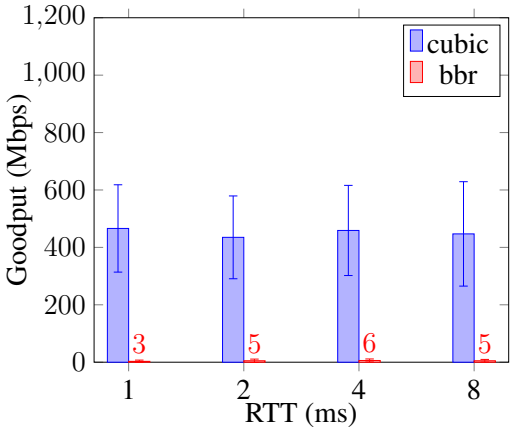
RAPID provides both flows with relatively fair bandwidth shares under LOS conditions, as illustrated in Figure 5.3(e) and the RTT increase due to Cubic overshooting becomes almost negligible (see Figure 5.3(f)) thanks to RAN-aware flow control. However, in case of NLOS conditions, although RAPID reduces the self-inflicted bufferbloat by a factor of 50 as illustrated in Figure 5.3(h), BBR is unable to fully exploit its allocated bandwidth share, as it is subject to throughput oscillations in case of fast variations in delay [73]. Figures 5.4(a) through 5.4(h) show the test results for the mixed fast/app-limited downloads scenario. In the cases where RAPID is not used (Figures 5.4(a), 5.4(c)), it can be seen that the app-limited flow is unable to obtain its desired bandwidth share (i.e., 16 Mbps) due to the fast download flow that overshoots more than the BDP. As a result, the slow flow is limited to less than 1 Mbps under LOS conditions and around 5 Mbps under NLOS. The increase in BBR goodput in case of NLOS is simply due to Cubic's sending rate decrease caused by packet losses. In fact, even after Cubic halves its congestion window, the goodput achieved by the slow flow is still 3 times less than the desired goodput.



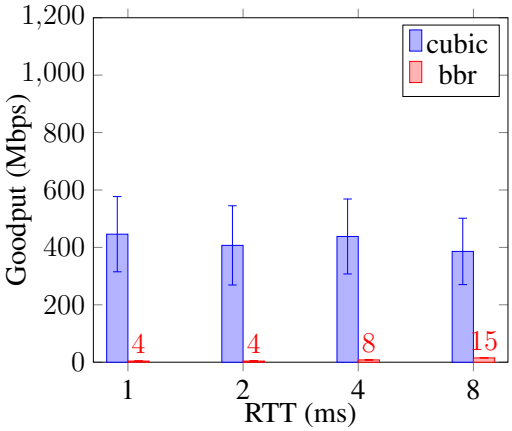
(a) Goodput in LOS without RAPID



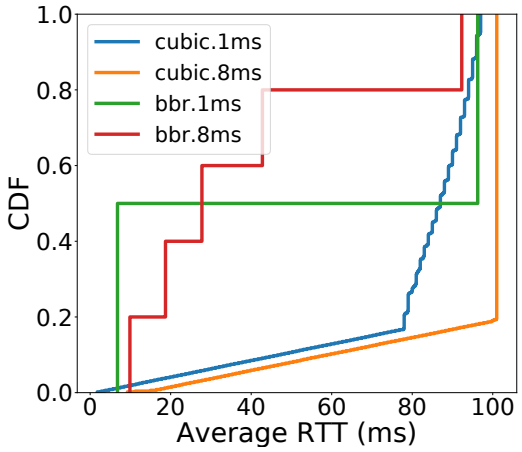
(b) Goodput in LOS with RAPID



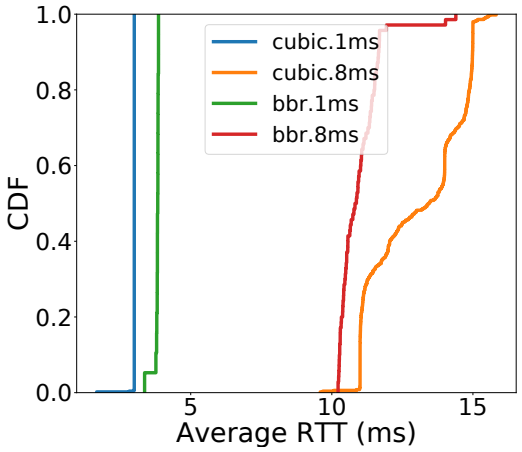
(c) Goodput in NLOS without RAPID



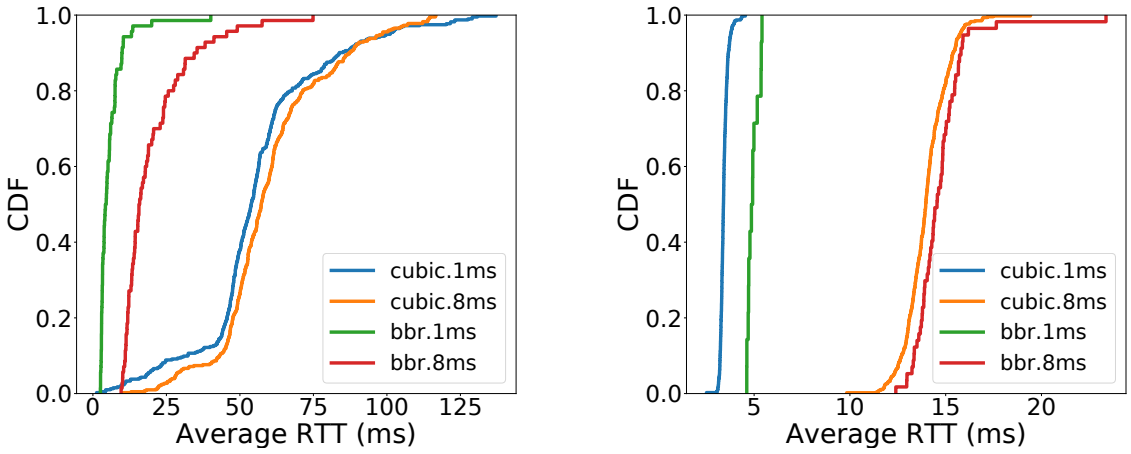
(d) Goodput in NLOS with RAPID



(e) RTT increase in LOS without RAPID



(f) RTT increase in LOS with RAPID



(g) RTT increase in NLOS without RAPID

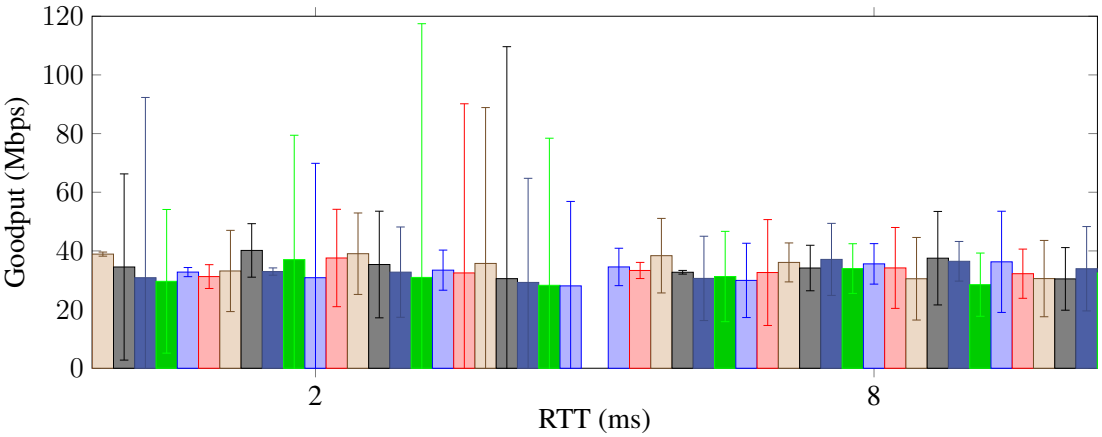
(h) RTT increase in NLOS with RAPID

Figure 5.4 – Mixed Fast and App-limited/Slow downloads scenario

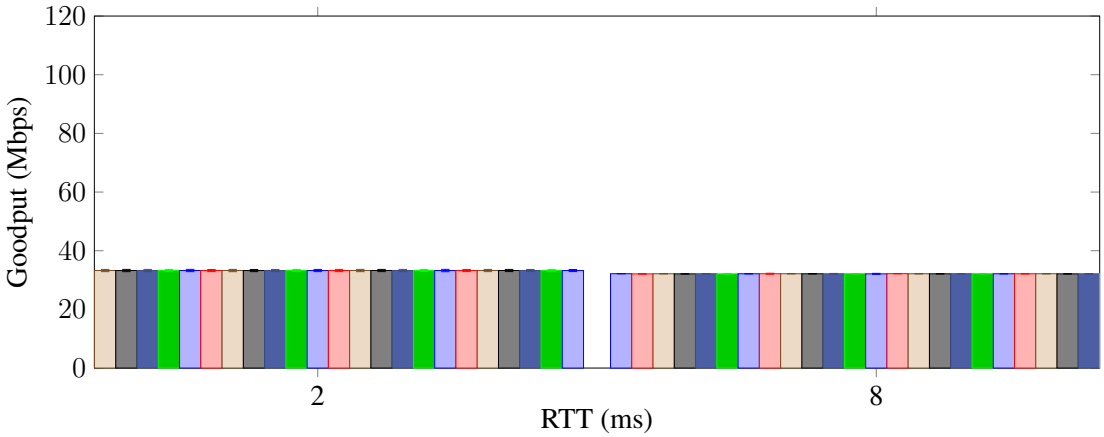
As a result, the PLT for a 2 MB web page is around 3 to 16 seconds. Moreover, as depicted in Figure 5.4(e), over an 80-factor increase in the end-to-end RTT can be observed for the slow flow in some cases. Such a large delay increase is unacceptable both for traditional web browsing and throughput-intensive ultra-low latency applications.
On the other hand, in the cases where RAPID is used (Figures 5.4(b) and 5.4(d)), the app-limited flow is allocated on average 11 Mbps in LOS and 8 Mbps in NLOS. Furthermore, unlike in the first scenario where RAPID equally shares the available bandwidth between the two download flows, here, the slow flow is automatically detected thanks to Equation 5.8 and is allocated a Receive Window proportional to its average data rate using Equation 5.10. As shown in Figures 5.4(b) and 5.4(d), this mechanism allows the fast download flow to exploit all the available bandwidth not used by the app-limited flow while remaining bounded by the actual RAN capacity. This not only improves the link utilization but also reduces the RTT increase for both flows by a factor of 10 to 50. Overall, thanks to RAPID flow categorization and bandwidth allocation schemes, the PLT is reduced by 90% (i.e., from 16 s to 1.5 s).

Besides evaluating RAPID's behavior under basic network conditions, we also consider other parameters that may affect its performance in real-world deployments :

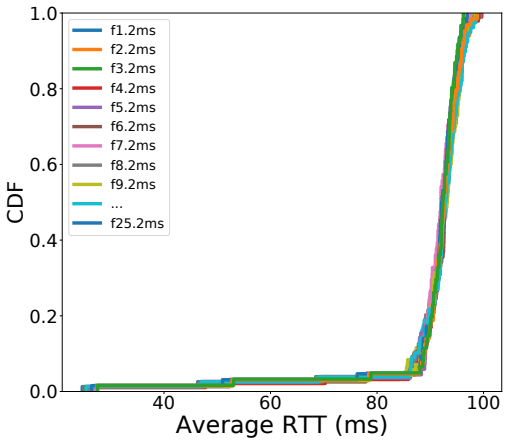
Scalability : In commercial mobile networks, there is no limitation on the number of flows a user can maintain in parallel. Therefore, in order to be efficient in such networks, RAPID must exhibit good performances regardless of the number of active flows per user. Figure 5.5 illustrates this behavior by showing the bandwidth allocation and delay distribution of 25 Cubic flows progressing in parallel in the same UE for both 2 ms and 8 ms end-to-end RTT configurations. From this figure, it can be seen that when RAPID is not used, a large degree of variability is observed between the goodput of the Cubic flows. This is due to packet losses (from an RLC buffer overflow) occurring at different moment at each new run. On the other hand, with RAPID, each flow gets the same goodput and the large variability is not observed anymore, simply because, by making sure the available radio capacity is never exceeded, RAPID completely prevents packet losses that arise from an RLC buffer overflow.



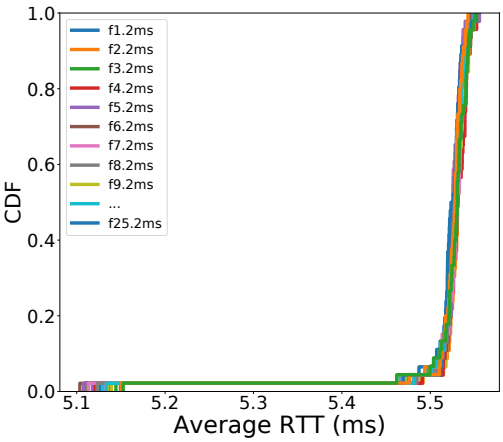
(a) Goodput distribution of 25 Cubic flows without RAPID



(b) Goodput distribution of 25 Cubic flows using RAPID

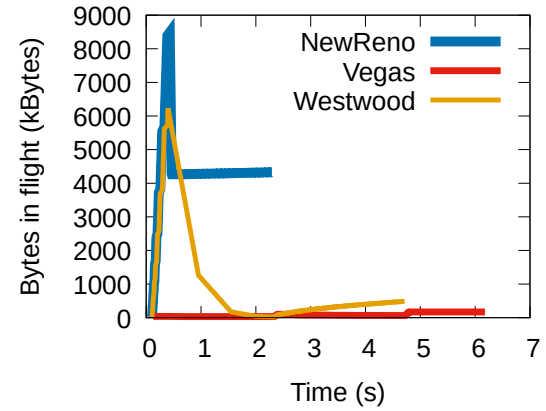


(c) Corresponding RTT increase without RAPID

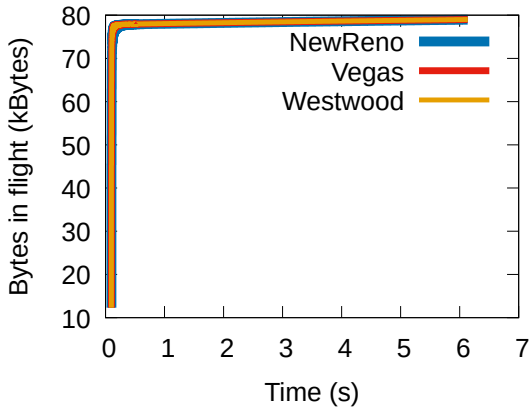


(d) Corresponding RTT increase with RAPID

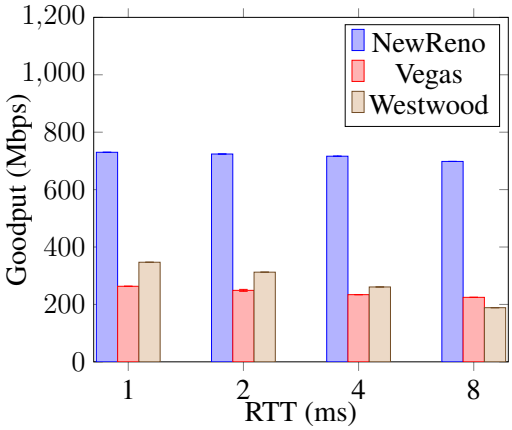
Figure 5.5 – Fast-download with 25 Cubic flows progressing in parallel in the same UE



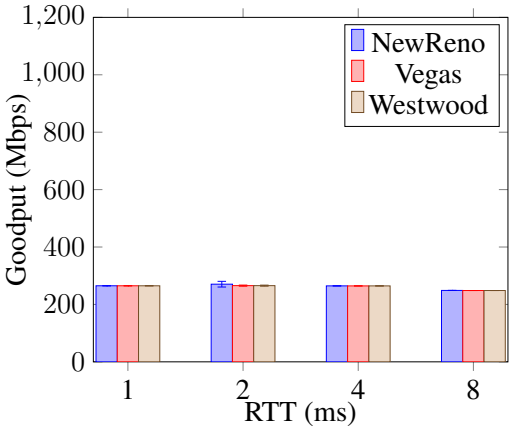
(a) Bytes in flight without RAPID



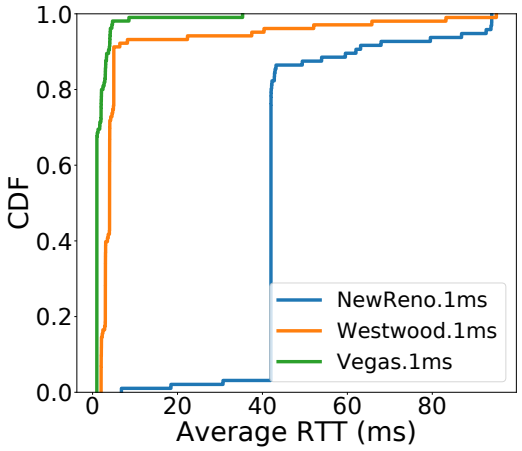
(b) Bytes in flight with RAPID



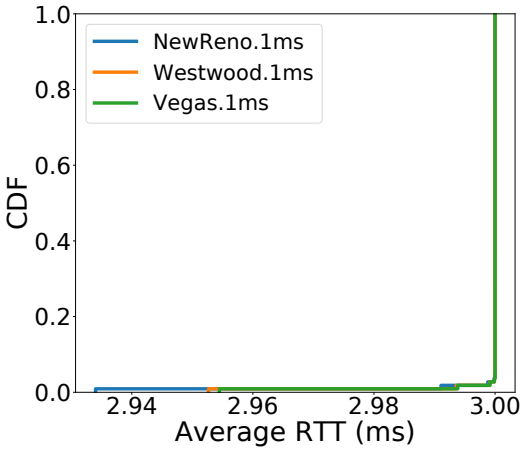
(c) Goodput without RAPID



(d) Goodput with RAPID



(e) RTT increase without RAPID



(f) RTT increase with RAPID

Figure 5.6 – Concurrent NewReno, Vegas and Westwood flows

Packet loss : In most cases and as Equation 5.7 assumes, packet loss in the backhaul network is generally negligible. This owes to the fact that, in commercial deployments, backhaul link dimensioning is done in such a way that the peak data rate or at least the average data rate of the cell is supported [80]. After all, if the backhaul link limits the cell data rate, there would be no point for the operator to invest in large frequency spectrum. On the other hand, packet loss may still occur in the RAN segment because of bad radio conditions or buffer overflow. The latter is avoided when using the RAPID's flow control mechanism, while the former is handled by RAPID's simplified CCA which always adapts its sending rate to the available RAN capacity regardless of random losses.

In this thesis, we mainly focus on scenarios involving Cubic and BBR as they are currently the two main CCAs in use on the Internet. Nevertheless, we have also evaluated RAPID with other well-known CCAs such as NewReno, Vegas [21], Yeah [15] and Westwood [79] that still control a large portion of today's Internet traffic. The results of these additional experiments are shown in Figure 5.6 where we evaluate a simple Fast-Download scenario in LOS condition involving NewReno, Vegas and Westwood in a 1 ms RTT configuration. As expected, our experiments show that RAPID allows delay-based TCP variants like Vegas (known for their very poor performances when competing with loss-based CCAs) to achieve similar goodput as NewReno or Westwood, while avoiding high delay increase. Figures 5.6(a) and 5.6(b) show, for a single run, the bytes in flight of the 3 flows as they progress in parallel on the same UE. It can be seen from Figure 5.6(d) that RAPID equally shares the total bandwidth (i.e., 850 Mbps) between the 3 flows. All the flows get a similar completion time (around 6 s) as illustrated in Figure 5.6(b) and the overall delay increase is reduced by a factor of 10 to 90, see Figures 5.6(e) and 5.6(f). As illustrated in Figure 5.6(a), without RAPID, NewReno gets by default the lowest completion time (around $2s$) because of its aggressive congestion growth function. In other words, it always maintains a high number of bytes in flight which negatively impacts the other competing flows, and in particular Vegas, as the latter is delay-based. As a result, NewReno always grabs a larger share of the total bandwidth (see Figure 5.6(c)) at the cost of over 70 times increase in delay (i.e. from 1 ms to about 80 ms) and with a high degree of unfairness to the other flows. Thanks to RAPID, a certain degree of fairness is maintained between the competing flows. Despite their different designs, all the 3 CCAs are imposed a similar number of bytes in flight, which allows them to obtain the same share and finish approximately at the same time, as illustrated in Figures 5.6(b) and 5.6(d). Note that all the scenarios and results shown in this section can be reproduced with ns-3 using the codes and scripts that we have made publicly available [117].

5.3.4 Bandwidth overhead

To work properly, RAPID requires an out-of-band control channel in the backhaul as shown in Figure 5.1. This control channel is necessary in order to allow the continuous retrieval of radio information, therefore, its requirements in terms of bandwidth depend both on the size of the retrieved radio information and the configured polling period. In our simulations, we use a polling period at subframe granularity (i.e., radio information is fetched every ms) and the size of the retrieved information per UE including TCP and IP headers is around 170 bytes. In other words, each UE introduces an overhead of 1 to 1.5 Mbps in the control channel, which is close to the FlexRAN overhead in real-world deployments [45]. We believe that such an overhead that accounts only for 0.13% of the achievable throughput (i.e., 850 Mbps in our case) and $10^{-8}\%$ of a typical 10 Gbps backhaul, is negligible compared to the resulting 150% and 3600% BBR's goodput increase in

fast-download and web browsing scenarios, respectively (see Figures 5.3(b) and 5.4(b)) as well as the overall end-to-end delay reduction in the range of 80 to 97%, as shown in Figures 5.3(h) and 5.4(h).

5.3.5 Discussion

Our simulation results demonstrate that RAPID significantly mitigates the self-inflicted bufferbloat issue in mobile networks, while maintaining near-optimal link utilization (Figures 5.3(b), 5.4(b)). The idea is to impose a certain level of fairness on the concurrent flows depending on their nature and on the available radio bandwidth. However, even if all the flows are assigned the same Receive Window, their performance in terms of goodput and link utilization still depends on the design of their respective CCA. Indeed, depending on their growth functions, different CCAs will reach a given number of bytes in flight at different times. Therefore, CCAs with faster growth functions will yield better link utilization. In addition to that, it is worth noting that the upcoming 5G and 6G networks are expected to deliver throughput in the range of several gigabits per second. So, with the design and growth functions of the current CCAs, it would be virtually impossible to reach such high throughput in a couple of RTTs. Therefore, it becomes necessary to rethink congestion control approaches in order to maintain high link utilization with minimum latency, which is very challenging as indicated in [109], mostly because of the highly intermittent nature of mmWave capacity.

In such a situation, we believe that providing flow-level RTT control and bufferbloat mitigation with proxy-based solutions such as RAPID will significantly simplify the design of future CCAs by allowing them to focus only on fast convergence.

5.4 Evaluating RAPID with OpenAirInterface

In this section, we first describe the design and implementation of our proxy as a Linux kernel module based on well-known open-source projects. Then, we showcase its efficiency when integrated into a 4G OAI infrastructure by conducting various experiments. Last but not least, we discuss the observed limitations and propose some workarounds.

5.4.1 Design and implementation

All the software components used in the implementation of RAPID as well as the way they interact with each other are illustrated in Figure 5.7. Basically, our Linux implementation is built on top of two open-source projects, namely pepsal [23] and FlexRAN [45]. Pepsal is an open-source Performance Enhancing Proxy (PEP) that splits incoming TCP connections, as such, it naturally replaces RAPID's TCP split functional module (illustrated in Figure 5.1). FlexRAN on the other hand brings the Radio Network Information Service (RNIS), which is exploited via Advanced Message Queuing Protocol (AMQP) so that the pepsal process can periodically receive real-time radio information. The latter information is then cleaned and sent via ioctl to the RAPID kernel module, which enforces the per-flow bandwidth allocation mechanism on the intercepted TCP connections.

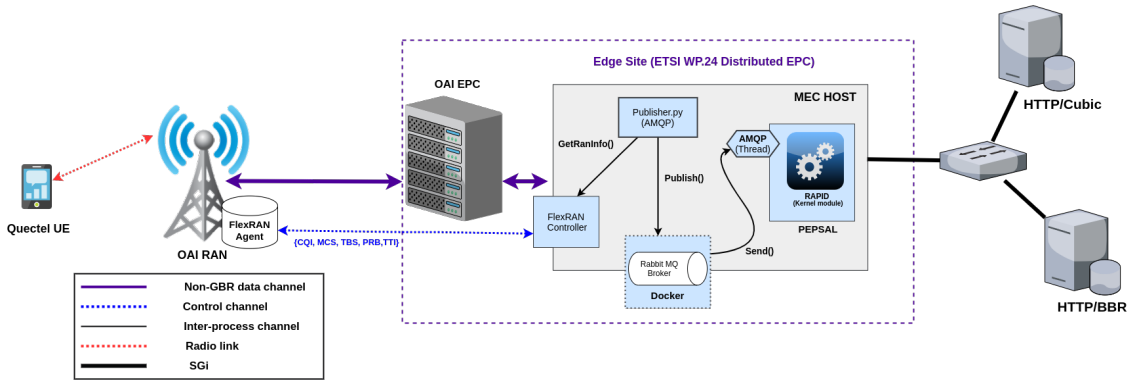


Figure 5.7 – RAPID OAI experimentation testbed

5.4.2 Experimentation and results

The experimentation setup illustrated in Figure 5.7 shows how we deploy RAPID along with an OAI 4G infrastructure. Also, it is important to note that this deployment method is by default compliant with any commercial 4G/5G networks. While our architecture is based on the "ETSI WP.24 Distributed EPC approach" [49], where the Evolved Packet Core (EPC) is located together with the MEC host at the edge site, it is worth noting that RAPID can also be evaluated following the "Bump in the wire approach" [49], in which the MEC platform is located between the base station and the mobile core. Table 5.3 describes the characteristics of our experimentation setup in the R2lab anechoic chamber*. With these parameters, we assess the following key aspects of RAPID using *wget* and *iperf* generated traffic :

Demand awareness and RTT reduction : We validate the efficiency of these features by reproducing the two scenarios presented in Section 5.3 under LOS conditions. The Fast eMBB Download is reproduced by launching two *wget* traffic, each simultaneously downloading a 60 MB file from two different HTTP servers that respectively use Cubic and BBR as CCAs. Similarly, the second scenario is reproduced by running two *wget* traffic in parallel using Cubic, followed later by a short SSH connection under BBR which displays a 500 kB text file using the Linux *cat* command. Figures 5.8 and 5.9 show the results of these experiments for 10 successive runs with 95% confidence interval.

In the first scenario, we observe that RAPID provides as expected a high level of fairness between Cubic and BBR despite the respective differences of the two CCAs. In the normal OAI deployment (i.e., without RAPID), Cubic grabs almost all the available bandwidth aggressively while BBR exploits only the remaining 25% (at least until the Cubic flow ends) as illustrated in Figure 5.8(a). This causes the average RTT to grow from 50ms to over 400 ms. Such a high increase in RTT not only penalizes BBR in terms of goodput, but also in terms of delay since the BBR flow exhibits average RTT values around 300 ms in the 80th percentile (see Figure 5.8(b)). On the other hand, when RAPID is deployed along with OAI, it can be observed from Figures 5.8(a) and 5.8(c) that Cubic and BBR achieve an average goodput of 8 and 7.8 Mbps, respectively, while maintaining RTTs below 100 ms in the 90th percentile. These results show that RAPID can allow in sheer download scenarios, over 4 times or 75% RTT reduction in a real-world 4G network

*. R2lab Testbed : http://r2lab.inria.fr

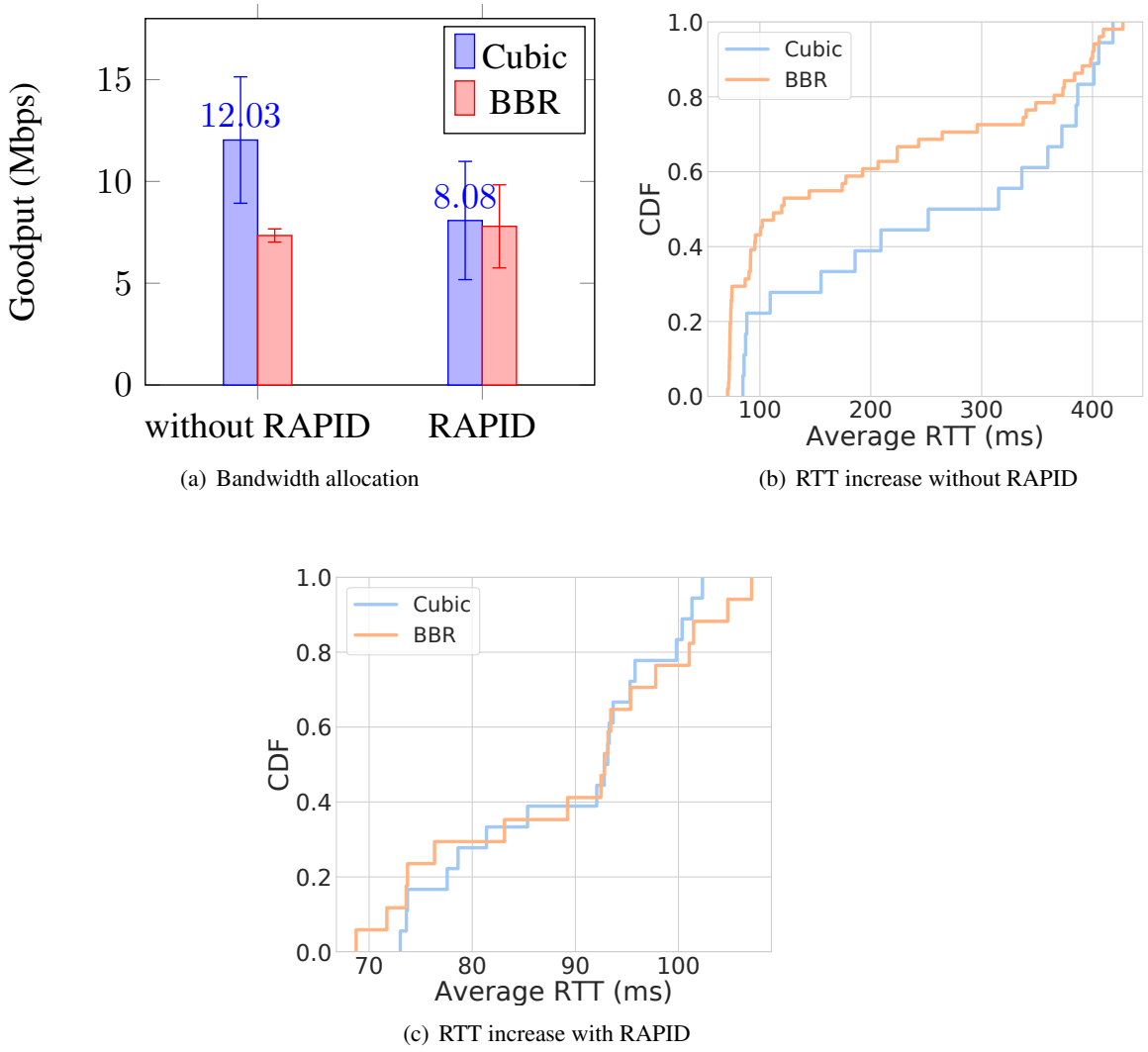
while maintaining a fair bandwidth allocation regardless of the differences between the concurrent CCAs.

TABLE 5.3 – OpenAir experimentation parameters

Parameters	Values
LTE mode	FDD
LTE bandwidth	5 MHz
Number of PRBs	25
RLC AM/UM buffer size	1 MB
UE	5G Quectel RM500Q-GL
End-to-End RTT	50 ms
Max. RAN Capacity	16 Mbps
Backhaul/s1 Bandwidth	1 Gbps

In the second scenario, from the results illustrated in Figure 5.9, we observe that the short SSH traffic takes on average 7 seconds in the standard deployment while it only takes 1 second with RAPID. In other words, RAPID enables 85% or 7 times reduction in the short flow completion time. This is in fact possible thanks to the low buffer occupancy resulting from RAPID's bandwidth allocation scheme and the demand awareness feature, which allows RAPID to allocate enough bandwidth to the SSH flow during its lifetime. This experiment demonstrates that RAPID is effective in real-life scenarios where the user maintains short and long flows at the same time. For instance, a user may want to surf on social media while a heavy software update is progressing in the background.

Impact of background UDP traffic : RAPID has been designed to only intercept TCP traffic ; therefore, protocols based on UDP such as QUIC or RTP are not intercepted and can affect RAPID's per-flow bandwidth allocation scheme. As described in Section 4.3, RAPID estimates the demand of a flow by using successive categorization periods upon which the average arrival rate of incoming packets is compared to the expected throughput (i.e., estimated radio bandwidth divided by the number of connections). It is important to detail the three reasons that can prevent the flow from reaching this expected throughput : (1) the flow has a low demand (e.g., Web browsing flow) ; (2) the capacity of the backhaul is less than the current radio capacity allocated to the UE ; (3) one or several concurrent data traffic (most likely unintercepted) exceed the capacity of the backhaul link. Supposing we are using a sheer download TCP flow, in this case we can eliminate (1) but also (2) since this condition is never met in commercial networks [80]. Likewise, the likelihood of observing (3) is close to zero in case of Constant-Bit-Rate (CBR) UDP flows such as Voice over IP (VoIP) traffic since their requirements in terms of bandwidth are very low (e.g., from 64 to few hundreds of kbps [18]). Such flows would even benefit from a significant decrease in delay and jitters thanks to the very low buffer occupancy enabled by RAPID. However, UDP flows that mimic the behavior of TCP such as QUIC could potentially overshoot the backhaul link depending on their CCAs. In such a situation, RAPID would continue to maintain a radio BDP worth of bytes in flight as long as the concurrent QUIC flow does not exceed the BDP of the backhaul. Once this occurs, RAPID would reduce the bandwidth allocated to the TCP flow (at the end of each categorization period), since it would no longer be able to reach the expected arrival rate. RAPID would eventually increase the bandwidth allocation of the TCP flow when the QUIC flow backs off due to a buffer-overflow or an RTO expiration. Figure 5.10 illustrates this mechanism in case of two



(a) Bandwidth allocation

(b) RTT increase without RAPID

(c) RTT increase with RAPID

Figure 5.8 – Goodput and RTT increase in OAI-4G with and without RAPID

competing TCP and QUIC flows using both a loss-based CCA. As shown in this figure, RAPID forces the TCP flow to operate around the radio BDP after the backhaul BDP is exceeded. Such a behavior allows the TCP flow to grab its fair share on average at the cost of delay increase, even when competing with aggressive long-lasting QUIC flows. Furthermore, it is worth noting that the latter explanation holds true only for long-lasting QUIC flows using aggressive CCAs. In cases where the competing QUIC flows use a less aggressive or delay-aware CCA such as BBR, RAPID allows the intercepted TCP flow to outperform QUIC both in terms of goodput and delay. In fact, this second scenario is the most likely to occur on the Internet since Google plans to use BBR on all its QUIC and TCP traffic [25].

Thus, in order to validate the aforementioned behaviors, we evaluate RAPID's performance under constant bit rate UDP traffic and also under QUIC traffic controlled by loss-based (i.e., Cubic) and model-based (i.e., BBR). For the CBR UDP scenarios we rely on *iperf* in order to

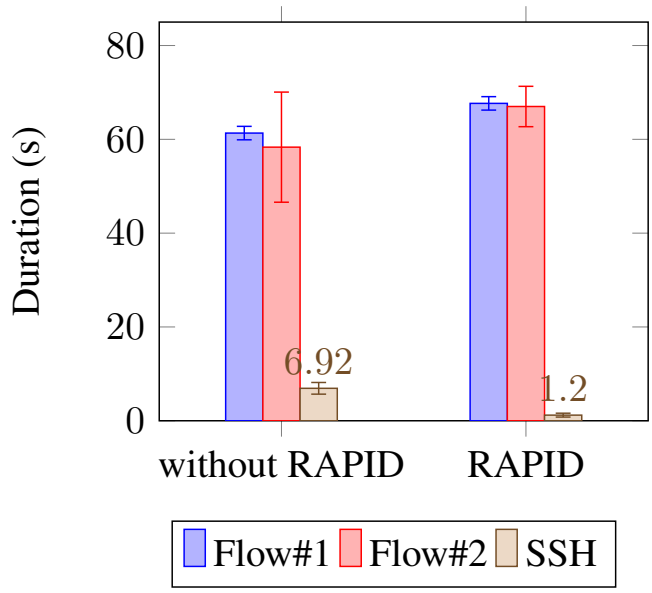


Figure 5.9 – Short flow duration during fast download

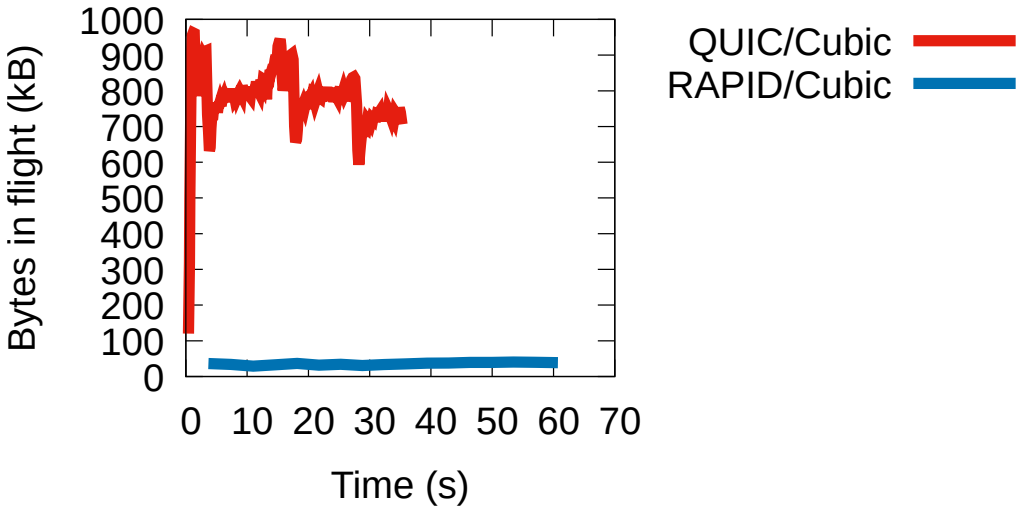
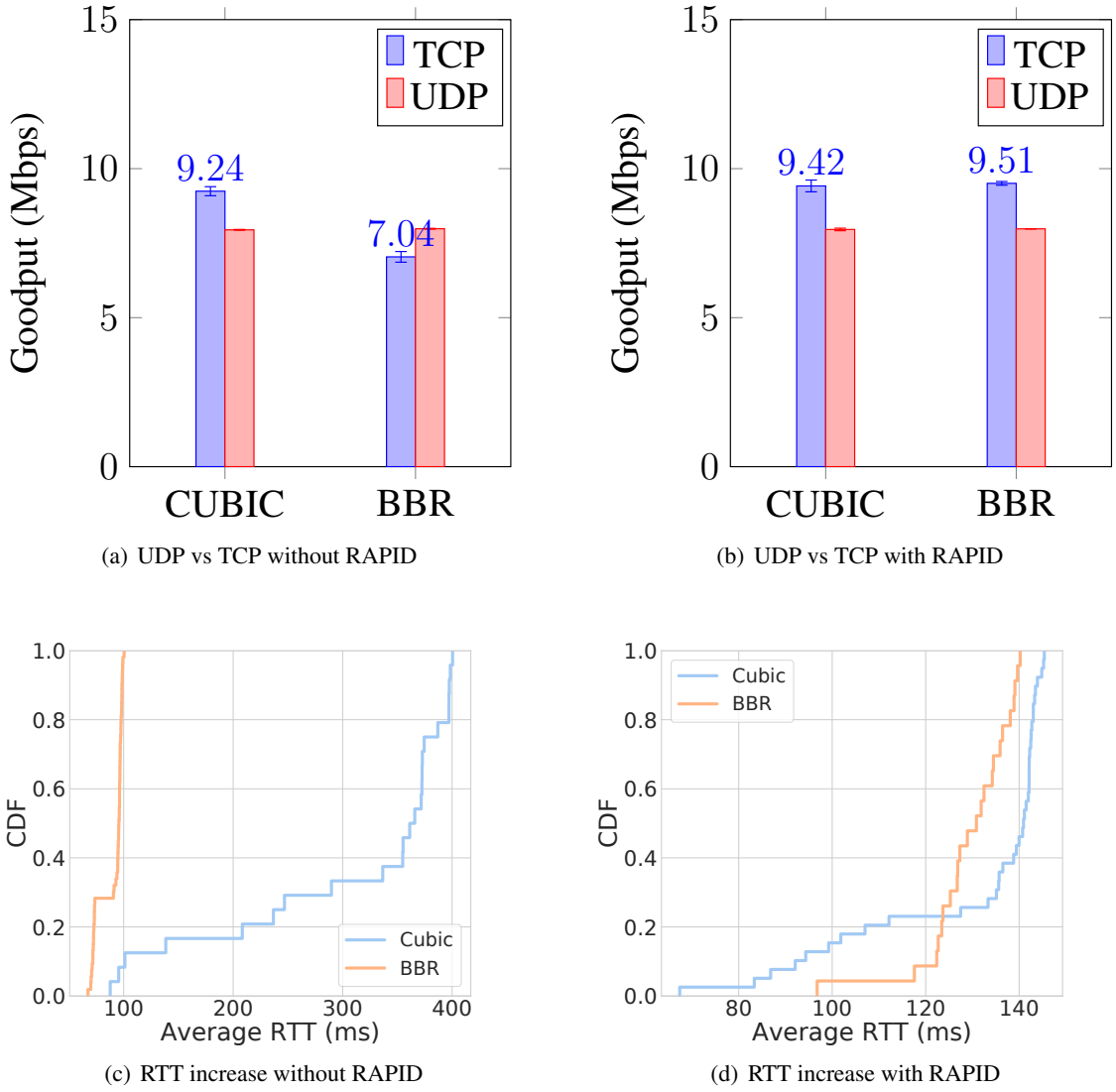


Figure 5.10 – RAPID/Cubic competing with QUIC/Cubic

generate an 8 Mbps UDP traffic competing with a *wget*-generated TCP flow and then we run another experiment which involves a 150 kbps UDP traffic also competing with a *wget*-generated TCP flow. We then change the CCA used by the TCP flow in order to observe the impact of the CBR flow on Cubic and BBR. For the scenarios involving QUIC, we rely on picoquic in order to generate QUIC flows that use Cubic and BBR. The goal of these experiments is to show that RAPID performs well both under aggressive and low bit-rate UDP traffic. This allows CBR traffic such as VoIP or streaming flows to run successfully alongside any TCP flow without experiencing a significant increase in delay and jitter. In fact, VoIP flows are considered of best quality only if the one-way latencies are below 130 ms [69]. Our experiments show that this threshold is exceeded

in the wild when an aggressive TCP download is progressing in parallel. More specifically, we observe the following phenomenons when a background UDP traffic is introduced.



(a) UDP vs TCP without RAPID

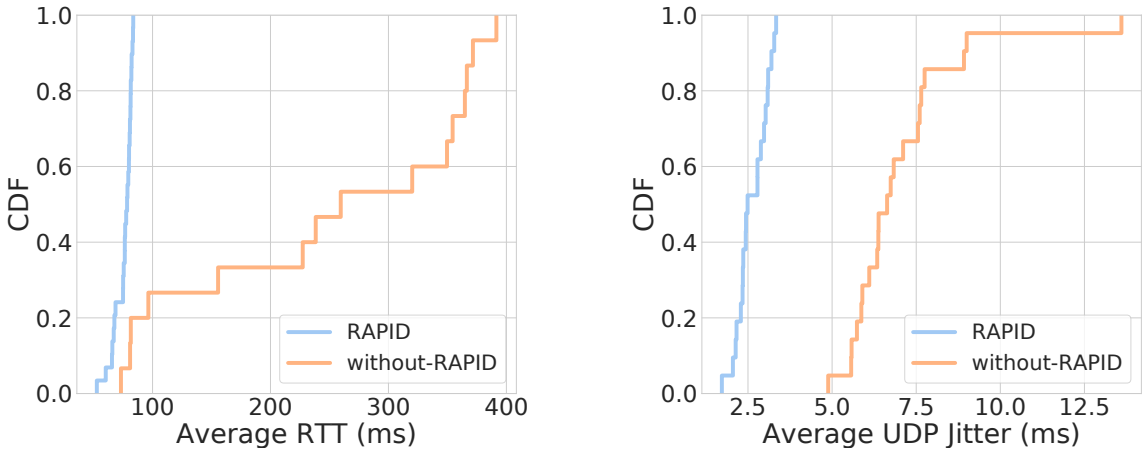
(b) UDP vs TCP with RAPID

(c) RTT increase without RAPID

(d) RTT increase with RAPID

Figure 5.11 – Goodput and RTT in OAI-4G under background UDP traffic

First, for the 8 Mbps CBR traffic, we observe several performance issues in the standard OAI deployment (strongly driven by the CCA in use). As illustrated in Figure 5.11(a), when RAPID is not used, we observe that the TCP flow exceeds its 8 Mbps fair share when competing with an 8 Mbps UDP flow, which, in case of a loss-based CCA, indicates an aggressive growth behavior. This situation causes a tremendous RTT increase, from 50 to over 350 ms (as shown in Figure 5.11(c)), which naturally hinders the quality of any concurrent VoIP traffic. On the other hand, for the same experiment, BBR shows a relatively lower performance in terms of goodput. It can be seen from Figure 5.11(a) that the BBR flow cannot exceed 7 Mbps on average even though the UDP flow is limited at 8 Mbps. In other words, BBR is not able to exploit the remaining 1 Mbps not



(a) RTT increase for the TCP flow

(b) Jitter increase for the UDP flow

Figure 5.12 – Average RTT and UDP Jitter under 150 kbps CBR

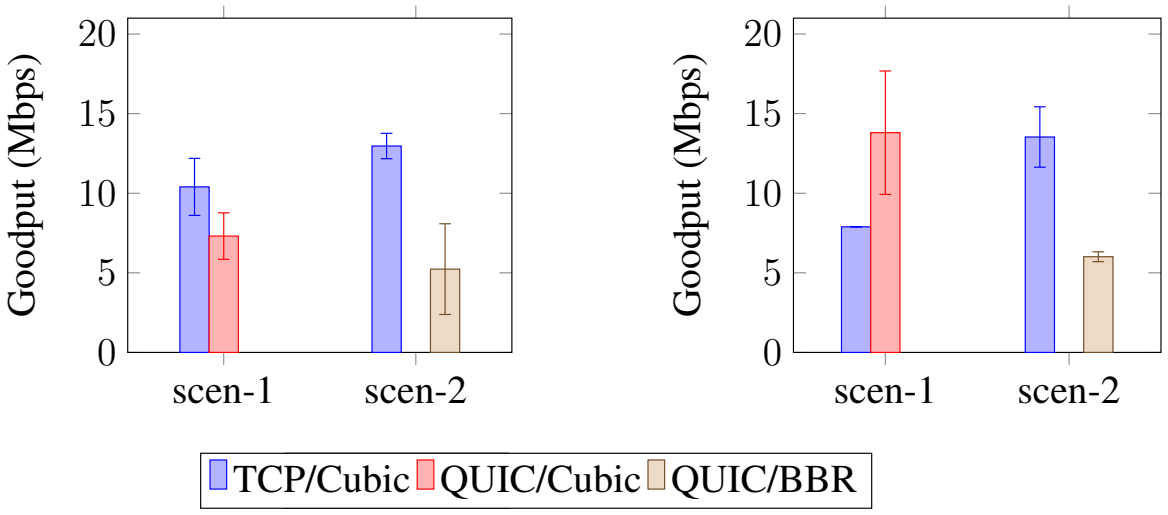
used by the UDP flow. While this behavior prevents delay increase as shown in Figure 5.11(c), it also enforces low link utilization since over 12% of the remaining bandwidth is not exploited. This low link utilization issue is actually due to incorrect BBR measurements of RTTmin in the OAI environment. Since BBR is prone to "throughput collapse" when the delay variation is high or when the minimum RTT estimate is too small [73], the observed low throughput was indeed expected from the fast-varying radio environment of OAI.

However, after enabling RAPID, we observe significant improvements for both CCAs. As expected, RAPID allows Cubic and BBR to fully exploit the remaining bandwidth at the cost of a relatively small delay increase as shown in Figures 5.11(b) and 5.11(d). While Cubic benefits from over a five time decrease in delay, BBR exhibits both low delay and over 35% increase in goodput. Unlike the previous experiment, here BBR is not affected by the fast delay variations in the radio segment. In fact, thanks to the TCP split feature, the BBR control loop remains in the wired segment, i.e., between RAPID and the end-server which allows BBR to get correct RTTmin estimates. Furthermore, the congestion window in the radio segment, computed based on the estimated BDP (from on Equation 5.10 or 5.13) prevents large buffering at the base station. These results demonstrate that RAPID can share the available bandwidth with a UDP traffic constantly exploiting half of the bandwidth without introducing a considerable delay increase.

Similarly, the results obtained from the 150 kbps CBR experiment, illustrated in Figures 5.12 and 5.12, show that RAPID enables a significant decrease in delay and jitter for time-sensitive flows. In addition to allowing the UDP flow to reach its required goodput (i.e., 150 kbps), over a six times jitter reduction and a 5 times delay reduction are observed. We owe it to the low buffer occupancy enforced by RAPID on the intercepted flow, thus preventing the aggressive TCP download from degrading the performance of the delay-sensitive UDP flow, which can be seen here as a basic VoIP flow.

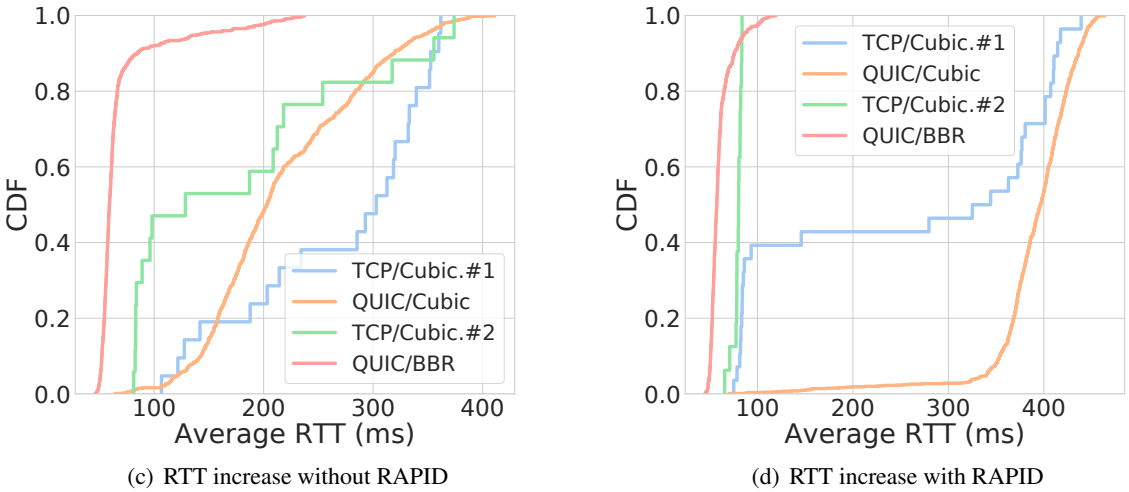
The experiments with background QUIC traffic show that RAPID's intercepted TCP flow grabs at least its fair share of the bandwidth in the best case scenario (i.e., when the QUIC flow is not aggressive), and manage to exploit at least 30% of the bandwidth in the worst case scena-

rio (i.e., when the QUIC flow is too aggressive). On the other hand, as shown in Figures 5.13(a) and 5.13(b), the QUIC flow exhibits overall poor performance when it uses BBR, either with or without RAPID.This performance penalty was expected given BBR's default behavior in response to delay increase or decrease in the network delivery rate. In fact, from a design point of view, the main goal of BBR is to operate near the Kleinrock operating point, i.e., making sure the sending rate matches the bottleneck delivery rate and the RTT stays around the minimum RTT value. As a result, when a BBR flow shares the bottleneck with another flow that has already filled the bottleneck pipe, the BBR flow observes a delay increase. Therefore, it decreases its sending rate as an attempt to match the reduced delivery rate and to rediscover the minimum RTT value.



(a) QUIC vs TCP without RAPID

(b) QUIC vs TCP with RAPID



(c) RTT increase without RAPID

(d) RTT increase with RAPID

Figure 5.13 – Goodput and RTT under Cubic/BBR QUIC traffic

Basically, in such a situation, the goodput that can be achieved by the BBR flow depends on how much delay or degree of buffering the concurrent flow can introduce. This phenomenon is

well reflected in Figures 5.13(a) and 5.13(b), from which it can be seen that the goodput achieved by the QUIC/BBR flow when RAPID is used to limit the buffering introduced by the TCP flow is slightly higher than the goodput it achieves when RAPID is not used, i.e., when the TCP flow introduces much more buffering. In any case, it is important to note that the BBR flow does not adopt such a behavior when it is intercepted by RAPID. It rather achieves roughly the same goodput as Cubic (see Figure 5.8(a)), because RAPID always makes sure the bottleneck capacity is adequately distributed between all the intercepted flows. However, in this scenario, RAPID is completely unaware of the QUIC flow. As a result, RAPID enforces the intercepted TCP flow to always operate at the radio BDP, regardless of the CCA used by the QUIC flow. Because of this, as illustrated in Figure 5.13(b), the intercepted TCP flow gets lower goodput when the QUIC flow exceeds the radio capacity (i.e., in case of QUIC/Cubic) and gets higher goodput when the QUIC flow sends less than the radio BDP (i.e., in case of QUIC/BBR). On the other hand, in terms of overall delay increase, we observe that the intercepted TCP flow starts experiencing considerable delay increase as soon as the QUIC flow exceeds the bandwidth of the RAN. The delay continues to grow as the QUIC flow increases the amount of byte in flight. In other words, the overall delay increase becomes less significant as soon as the QUIC flow stops. This is well illustrated in Figures 5.13(c) and 5.13(d), where a significant decrease in delay is observed at lower percentiles (e.g., 40th) for the intercepted TCP flow competing with QUIC/Cubic (i.e., for TCP/Cubic.#2). It is important to note that RAPID always tries to minimize its delay increase, even without being explicitly aware of the background QUIC traffic. This behavior is illustrated in Figure 5.10, where it can be observed that the amount of bytes in flight for the intercepted TCP flow decreases as the QUIC flow increases its sending rate. Overall, these results demonstrate that RAPID does not penalize QUIC flows (although they are not intercepted), but rather increases the performance of competing TCP flows.

5.4.3 Lessons learned from real-world experimentation

Despite the fact that RAPID allows an overall better performance with respect to standard TCP (i.e., end-to-end TCP), it is worth noting that it presents a minor limitation due to the inherent characteristics of real-world radio environments. More specifically, unlike in simulated environments (such as ns-3), where every single characteristic of the RAN is controlled, real-world cellular RAN are characterized by random and fast delay variations, which make tricky the measurement of the minimum RTT between the mobile and the proxy. In fact, we suspect that RTTmin is underestimated for most flows due to either low traffic load at the beginning of the connection (which is a well-known characteristic of TCP Slow Start algorithms), or because of the periodicity of the Scheduling Request (SR), which tells how often the base station allocates uplink resources to the UE. Consequently, since we use RTTmin in the computation of the flow BDP in the radio segment, it follows that the related flows suffer from low link utilization. This problem has already been reported in [92] which proposes a client-side RTTmin estimation method that takes the SR periodicity into account.

In our case, since we do not want to involve the client, we mitigate this issue differently. In fact, we use a smoothed RTTmin value that takes into account the minimum RTT estimated during each categorization period. With this method, the very low RTTmin values observed at the beginning of the connection are discarded and the RTTmin estimate gets better as the connection continues. We also set a relatively high value for the initial congestion window on the RAN segment so that RAPID can quickly reach the highest allowed CWND value (i.e., the estimation of the RAN BDP)

and directly deliver all the packets received from the original server. However, even though the smoothed RTTmin mechanism significantly improves the link utilization, we can still observe a slight bandwidth wastage as shown in Figure 5.8(a), which approximately accounts for 1% of the overall capacity (i.e., around 150 kbps).

5.5 Summary

In this chapter, we have highlighted the limits of the per-user queue isolation technique in mitigating the self-inflicted bufferbloat issue, especially in the case of multiple flows per user and intermittent mmWave links. We have also explored and pointed out the limitations of existing bufferbloat mitigation approaches. More specifically, we have shown that most of them either just delay the occurrence of bufferbloat or do not work properly in case of parallel flows. We have discussed their cost and requirements from a real-world deployment perspective and showed that they either require client-side modifications or changes in the standardized 4G/5G stack. As a response, we have proposed and presented our approach named RAPID, which is completely transparent to the client and the end server. We have demonstrated that this approach drastically reduces the delay increase while preserving good link utilization, regardless of the radio conditions or the CCAs in use at the end servers. We showed how it exploits up-to-date RAN statistics from MEC RNI service as well as specific TCP insights and capabilities in order to analyze the behavior of the competing flows and prevent them from exceeding the RAN capacity.

Through ns-3 simulations and real-world 4G OAI experiments, we demonstrated that RAPID not only offers significant performance improvement in cellular networks with MEC, but also mitigates BBR's limitations in fast-varying radio environments, and this without requiring any modifications or software patches at the server. We also demonstrated that our solution does not penalize QUIC flows, although they represent only 7% of today's Internet traffic (according to Google [75]). Through our results, we showed that RAPID enables traditional loss-based CCAs such as Cubic or Reno, which are known to introduce high delay increase in deep buffer environments, to outperform BBR in terms of goodput while maintaining comparable and even lower delays in certain scenarios.

To the best of our knowledge, RAPID is the first attempt to mitigate self-inflicted bufferbloat and interference between CCAs at TCP level.

CHAPTER 6

Conclusion and Perspectives

This chapter concludes and ends this thesis. It summarizes the ideas and contributions exposed throughout the previous chapters, presents our consolidated point of view with respect to the obtained results and discusses various perspectives for future work .

6.1 Conclusion

Addressing TCP performance issues in cellular networks is still a hot research topic, despite the fact that a wide range of solutions have been proposed over the last decades. One reason that could explain this phenomenon is the difficulties related to the real-world adoption of the proposed solutions. As shown throughout this thesis, most of these solutions either require drastic changes in the cellular stack (in the UE and/or in the base station) or work only in a few scenarios. However, in this thesis we have demonstrated that most transport layer issues can be mitigated without requiring any modifications in the end-user's device or in the 4G/5G stack. The only requirement being the adoption of the ETSI MEC framework or any other edge computing framework that can make RAN information accessible to external applications. In our journey to demonstrate that, we addressed three major transport layer issues :

- The uplink utilization issue hinders TCP performance in terms of throughput and delay and can cause on-device bufferbloat. We addressed this issue in Chapter 3 by proposing SIGMA, a MEC-aware uplink-oriented CCA that exploits radio information on the user's device in order to skip the traditional Slow Start procedure and operate around the Kleinrock's optimal operating point. We demonstrated through mathematical models and ns-3 simulations that this solution offers a better trade-off between goodput and delay than the existing CCAs. We showed that the main limitation of SIGMA is the fact that it requires some changes in the user's device. However, we can argue that such a limitation is common to all uplink-oriented CCAs, since they all need to be deployed on the client, as opposed to downlink-oriented CCAs, which need to be installed on the server. Also, it should be noted that the uplink TCP throughput strongly depends on the CCA configured in the end-user's device, hence it becomes virtually impossible to improve this throughput, without modifying or changing the configured CCA.

- The random packet loss issue occurs in bad radio conditions, especially under RLC UM mode and hinders the achievable throughput of loss-based CCAs. This issue has been addressed in Chapter 4 by MELD, which replaces the default loss detection algorithm in the server by a RAN-aware loss discrimination algorithm. We showed that by adequately interpreting some specific radio information, retrieved through the RNI API, MELD can distinguish between pure congestive losses and losses that can safely be ignored. The goodput improvement that MELD enables for NewReno and Cubic has been demonstrated through controlled experiments in a 4G/LTE testbed based OpenAirInterface.

- The self-inflicted bufferbloat issue affects flows belonging to the same user. We addressed this issue in Chapter 5 by proposing RAPID, a RAN-aware Performance Enhancing Proxy, deployed on the MEC host (between the user and the end server) and that is able to transparently influence the behavior of the traversing TCP flows in such a way that their aggregated throughput does not exceed the available radio bandwidth. While describing RAPID's design, we insisted on the fact that its bandwidth allocation scheme is demand-aware, meaning that it influences the intercepted flows based on their respective demands, which are estimated based on packet arrival rates. We showed that this feature allows RAPID to enforce a demand-aware fairness principle at flow level, regardless of the CCAs in use. Through various evaluations in a 5G mmWave environment using ns-3 and in a real-world 4G environment based on OpenAirInterface, we showed that RAPID can reduce the delay increase by a factor of 10 to 50 in both LOS and NLOS radio conditions while preserving near optimal throughput.

In light of our different findings and results throughout this thesis, we draw the following major conclusions regarding transport-layer optimization in today's and future generations of cellular networks :

1) The large-scale adaption of ETSI MEC in commercial 4G/5G networks would not only enable URLLC use-cases, but would also open the way for a new range of transport-layer optimization solutions, that can be easily deployed without involving the end-user or modifying the 4G/5G stack ;
2) End-user's device manufacturers should consider using CCAs that are adapted to cellular networks, such as SIGMA in order to maximize the uplink utilization and avoid on-device bufferbloat. Otherwise, they should provide an API that allows application running on the user's device to passively interact with their proprietary 4G/5G modules in order to collect radio information ;
3) The content providers or enterprises that intend to deploy their applications on MEC platforms should be aware of the transport-layer issues that may arise and therefore should consider protecting their services/applications with adequate MEC-based solutions such as RAPID and/or MELD ;
4) Given the limited number of parallel DRBs and network slices per user, the mobile network operators should consider using MEC-based solutions such as RAPID in order to improve their Service Level Agreements (SLAs).
5) The traditional approaches used for congestion control up until now need to be rethought when the end-server is located in a MEC-enabled cellular networks, since some heavy and time-consuming mechanisms become completely unnecessary in such a context, as

demonstrated throughout this thesis.

Through this thesis we observed MEC from a transport-layer perspective and proposed some efficient and easy-to-deploy MEC-based transport-layer optimization solutions that would benefit network operators, content providers, and end-users. We made all these solutions open-source and publicly available to the community [115, 116, 117, 118]. We hope our work in this thesis inspires future transport-layer CCAs or more optimization solutions based on MEC.

6.2 Perspectives

Even though the solutions proposed in this thesis are promising, they can still be improved or adapted to suit some specific use-cases. With that in mind, we have identified a few areas that can constitute interesting research directions :

Improving the per-UE bandwidth estimation in RAPID : In RAPID's per-UE bandwidth estimation equation (Equation. 5.2), we compute the expected number of PRBs (i.e., the maximum bandwidth the UE can expect to get from the base station) by adding the current number of allocated PRBs to the number of unused/idle PRBs. The advantage of this method is that it is conservative as it does not try to directly claim the fair-share number of PRBs as it is done in SIGMA (see Equation 3.8), but rather considers the current number of allocated PRBs because it assumes that this allocation already corresponds to the fair-share number of PRBs. This assumption works well in LTE, because the traffic load created by a TCP flow upon its first burst of back-to-back packets (i.e., default IW of 10 segments) requires a downlink data rate of around 100 Mbps (which is more or less equivalent to the peak achievable data rate in LTE) to be sent within 1 TTI. Therefore, regardless of the number of connected users, we can assume that the TCP flows belonging to a given UE create enough load for the base station to allocate a number of PRBs that is at least equal to the fair share. However, when the peak achievable data rate of the cell is very high (for instance 20 Gbps) and some users in the cell are already exploiting most or all the bandwidth, a new user that arrives in the cell would not receive a fair-share number of PRBs by just introducing a load of 10 packets. In such a context, it would make more sense to explicitly claim the fair-share number of PRBs based on the total number of PRBs and the number of connected users as shown in Equation 3.8. With that in mind, it would be interesting to replace Equation 5.2 by Equation 3.8, then evaluate the accuracy of the per-UE bandwidth estimation in case of a very high radio bandwidth and also with various MAC scheduling algorithms.

Enhancing RAPID friendliness towards UDP and QUIC flows : In Chapter 5 we evaluated the efficiency of RAPID when a TCP flow competes with an unintercepted UDP or QUIC flow. From the overall results, we can consider that the additional delay increase introduced in case of QUIC/BBR (which is the most likely scenario when it comes to QUIC, since BBR is the de-facto CCA used by Google QUIC [25]) is negligible. However, in case of a CBR UDP flow, the introduced delay may become quite significant depending on the sending rate of the UDP flow. This additional delay can be reduced by prioritizing real-time UDP traffic over sheer download TCP traffic as done in SIGMA. This choice is based on the assumption that a normal user would be more affected by the performance of her real-time UDP traffic than by the speed of her background TCP download. This assumption holds true as long as all the uplink packets sent by the UE are generated locally. However, in cases where the UE plays the role of an Internet gateway,

prioritizing a UDP flow over a TCP flow can create a fairness issue, since the 2 flows might be coming from 2 different devices. We plan to further study this aspect in the future and consider a more dynamic and context-aware prioritization mechanism between TCP and UDP flows.

Considering wired bottleneck scenarios in the design of SIGMA : SIGMA was designed specifically for scenarios where the radio link constitutes the bottleneck. However, with further studies and experimentations, its design can be enhanced and generalized in such a way that it becomes also applicable to scenarios where the bottleneck location is not known.

Considering Centralized Congestion Control for edge servers : When RAPID is used, the intercepted TCP flows can never exceed the available RAN capacity since the flow control mechanism is intelligently controlled in such a way that the aggregated throughput never exceeds the available capacity. Technically speaking, RAPID can also be seen as a centralized congestion controller since it prevents geographically distributed senders from congesting the bottleneck. Therefore, since we know that RAPID already handles congestion control in a centralized way, it would be interesting to study whether it is necessary or counterproductive to still keep a CCA on the TCP servers that are deployed at the edge. For instance, by removing or ignoring congestion control, we know that the end-sender would be able to reach the indicated receive window (i.e., the available radio bandwidth) within just one RTT. We believe that this would significantly increase the goodput and the overall radio link utilization, especially in case of short-lived flows since no RTTs are wasted in probing (like in Max Start, see Section 3.2.3).

Considering RAPID deployment on 4G/5G modems/routers : The only requirement in order to implement RAPID is the ability to easily access RAN information. Therefore, MEC and RNIS are not required if the needed RAN information can be easily retrieved through an alternative means, for instance directly from the UE. However, in this thesis, we did not explore these alternative means since we wanted RAPID to be completely transparent and usable in general use cases. With that being said, for specific use cases that involve 4G/5G CPEs (Customer Premise Equipments) such as onboard WiFi (e.g., in buses/trains as shown in Figure 6.1) or industrial/enterprise 5G gateways, having RAPID directly implemented on the CPE would tremendously improve the Quality of Experience (QoE) of the served users. Such implementation should be considered by 4G/5G CPE manufacturers since they can easily retrieve the needed radio information from their proprietary 4G/5G firmware. This would be a very useful feature for bufferbloat mitigation, because a CPE generally serves several users. But from the 4G/5G network perspective, the flows from all these users are coming from the same UE (i.e., the CPE), so, they are naturally treated as parallel flows belonging to a single user. In other words, they generally share the same buffer. As a result, even if a single user (among the users served by the CPE) maintains a download or an aggressive flow, she will penalize all the other users. For instance, one user downloading a big movie via the on-board WiFi of a bus/train can hinder the QoE of all the users on this bus/train.

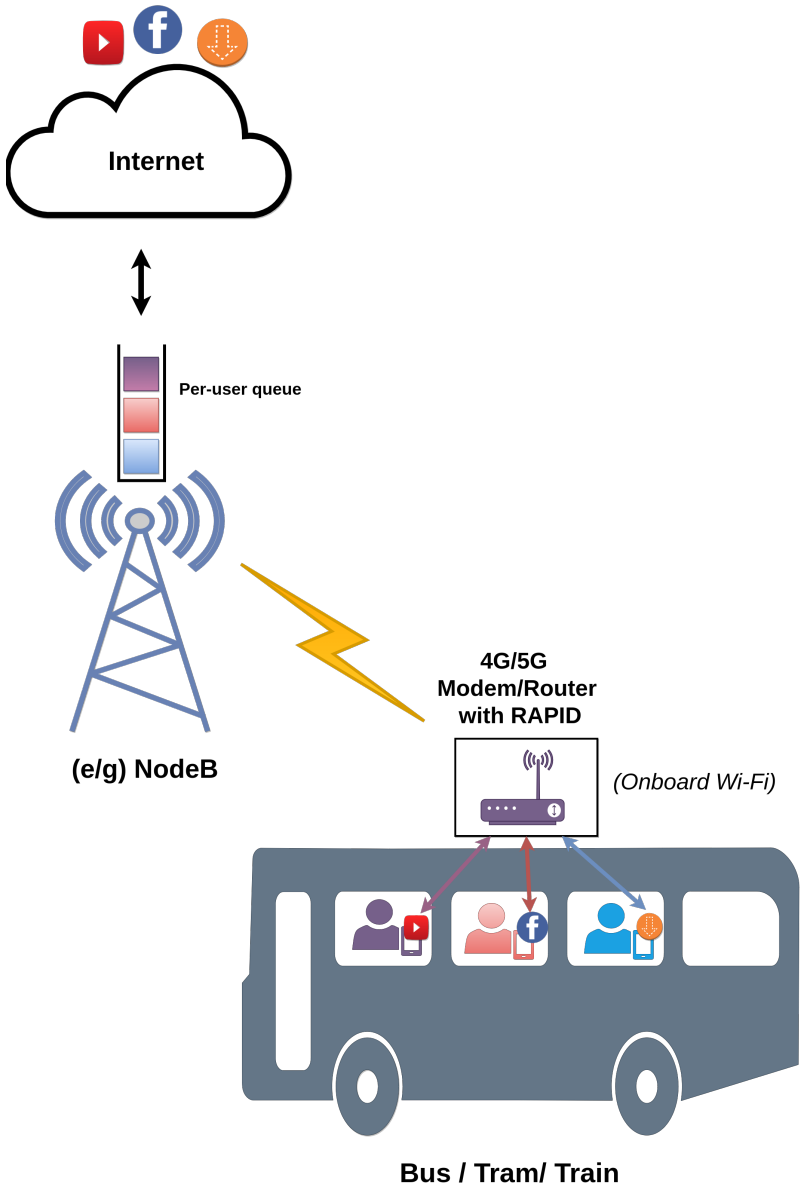


Figure 6.1 – RAPID integration into commercial 4G/5G modems for better on-board public WiFi experience : The 4G/5G modem is considered as a single UE, so the traffic from all the users on the bus shares the same RLC buffer at the base station – The use of RAPID alleviates the bufferbloat effect – It allows users with interactive flows to have a good experience even if other users are using sheer download flows.

References

[1] 3GPP. LTE Evolved Universal Terrestrial Radio Access (E-UTRA) ; Physical layer procedures. In *TS 36.213 version 14.2.0* (2017).

[2] 3GPP. 5g ; study on new radio (nr) access technology. *3rd Generation Partnership Project (3GPP), Technical Report (TR) 38.912 version 15.0.0 Release 15* (2018).

[3] 3GPP. LTE Evolved Universal Terrestrial Radio Access (E-UTRA) ; Physical layer procedures. In *TS 36.213 version 15.2.0* (2018).

[4] 3GPP. 5g ; nr ; base station (bs) radio transmission and reception. *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.104 version 16.4.0 Release 16* (2020).

[5] 3GPP. 5G ; NR ; Physical layer procedures for data. In *TS 38.214 version 16.2.0* (2020).

[6] 3GPP. Feasibility study for Further Advancements for E-UTRA (LTE-Advanced) (Rel. 16). In *3GPP TR 36.912 V16.0.0* (2020).

[7] 3GPP. Feasibility study for Further Advancements for E-UTRA (LTE-Advanced) (Rel. 16). In *3GPP TR 36.912 V16.0.0* (2020).

[8] 3GPP. Lte ; 5g ; evolved universal terrestrial radio access (e-utra) and nr ; service data adaptation protocol (sdap) specification. *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 37.324 version 16.2.0 Release 16* (2020).

[9] 3GPP. System architecture for the 5g system (5gs). *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.501 version 15.9.0 Release 15* (2020).

[10] ABBASLOO, S., XU, Y., CHAO, H. J., SHI, H., KOZAT, U. C., AND YE, Y. Toward optimal performance with network assisted {TCP} at mobile edge. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)* (2019).

[11] AGGARWAL, A., SAVAGE, S., AND ANDERSON, T. Understanding the performance of tcp pacing. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)* (2000), vol. 3, pp. 1157–1165 vol.3.

[12] ANDERSON, C., WOLFF, M., ET AL. The web is dead. long live the internet. *Wired Magazine 18*, 15 (2010), 1–12.

[13] ARORA SAGAR, ET AL. Exposing radio network information in a MEC-in-NFV environment : the RNISaaS concept. In *Proc. of NETSOFT* (2019).

[14] AZZINO, T., DRAGO, M., POLESE, M., ZANELLA, A., AND ZORZI, M. X-tcp : a cross layer approach for tcp uplink flows in mmwave networks. In *2017 16th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)* (2017), pp. 1–6.

[15] BAIOCCHI, A., ET AL. YeAH-TCP : yet another highspeed TCP. In *Proc. PFLDnet* (2007), vol. 7, pp. 37–42.

[16] BALASUBRAMANIAN, P., HUANG, Y., AND OLSON, M. Hystart++ : Modified slow start for tcp. *Internet-Draft draft-balasubramanian-tcpmhystartplusplus-03, Internet Engineering Task Force* (2020).

[17] BESHAY, J. D., ET AL. Link-coupled tcp for 5g networks. In *IWQoS 2017* (2017), pp. 1–6.

[18] BHANU, S. V., CHANDRASEKARAN, R., AND BALAKRISHNAN, V. Effective bandwidth utilization in ieee802. 11 for voip. *arXiv preprint arXiv :1005.0952* (2010).

[19] BLANTON, E., PAXSON, D. V., AND ALLMAN, M. TCP Congestion Control. RFC 5681, Sept. 2009.

[20] BORMAN, D., BRADEN, R. T., JACOBSON, V., AND SCHEFFENEGGER, R. TCP Extensions for High Performance. RFC 7323, Sept. 2014.

[21] BRAKMO, L. S., ET AL. TCP Vegas : New techniques for congestion detection and avoidance. In *ACM SIGCOMM* (1994), pp. 24–35.

[22] BRAKMO, L. S., AND PETERSON, L. L. Tcp vegas : End to end congestion avoidance on a global internet. *IEEE Journal on selected Areas in communications 13*, 8 (1995), 1465–1480.

[23] CAINI, C., FIRRINCIELI, R., AND LACAMERA, D. Pepsal : a performance enhancing proxy designed for tcp satellite connections. In *2006 IEEE 63rd Vehicular Technology Conference* (2006), vol. 6, pp. 2607–2611.

[24] CARDWELL, N., CHENG, Y., GUNN, C. S., YEGANEH, S. H., AND JACOBSON, V. Bbr : Congestion-based congestion control. *Commun. ACM 60*, 2 (jan 2017), 58–66.

[25] CARDWELL, N., CHENG, Y., GUNN, C. S., YEGANEH, S. H., SWETT, I., IYENGAR, J., VASILIEV, V., AND JACOBSON, V. Bbr congestion control : Ietf 100 update : Bbr in shallow buffers. In *Proc. IETF-100* (2017).

[26] CARDWELL, N., CHENG, Y., YEGANEH, S. H., AND JACOBSON, V. Bbr congestion control. *Working Draft, IETF Secretariat, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00* (2017).

[27] CARDWELL, N., CHENG, Y., YEGANEH, S. H., SWETT, I., AND JACOBSON, V. BBR Congestion Control. Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-02, Internet Engineering Task Force, Mar. 2022. Work in Progress.

[28] CHAE, M., AND KIM, J. What's so different about the mobile internet ? *Communications of the ACM 46*, 12 (2003), 240–247.

[29] CHANG, B.-J., ET AL. Cross-Layer-Based Adaptive TCP Algorithm in 4G Packet Service LTE-Advanced Relaying Communications. In *AISA - Vol. 1* (2013).

[30] CHEN, Y.-C., ET AL. Measuring cellular networks : Characterizing 3g, 4g, and path diversity. In *Conf. Int. Tech. Alliance* (2012).

[31] CHENG, Y., CARDWELL, N., DUKKIPAT, N., AND JHA, P. The rack-tlp loss detection algorithm for tcp. *RFC 8985. IETF* (2021).

[32] CHIARIOTTI, F., ZANELLA, A., KUCERA, S., AND CLAUSSEN, H. Bbr-s : A low-latency bbr modification for fast-varying connections. *IEEE Access 9* (2021), 76364–76378.

[33] DE LA OLIVA, A., HERNANDEZ, J. A., LARRABEITI, D., AND AZCORRA, A. An overview of the cpri specification and its application to c-ran-based lte scenarios. *IEEE Communications Magazine 54*, 2 (2016), 152–159.

[34] DIARRA, M., DABBOUS, W., ISMAIL, A., TETU, B., AND TURLETTI, T. Rapid : A ran-aware performance enhancing proxy for high throughput low delay flows in mec-enabled cellular networks. *Computer Networks 218* (2022), 109357.

[35] DIARRA, M., DABBOUS, W., ISMAIL, A., AND TURLETTI, T. Ran-aware proxy-based flow control for high throughput and low delay embb. In *Proceedings of the 24th International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems* (2021), pp. 41–50.

[36] DIARRA, M., ET AL. Cross-layer loss discrimination algorithms for MEC in 4G networks. In *Proc. of IEEE HPSR '21* (2021).

[37] DORLAN, P. L. *An introduction to computer networks*. Autoedición, 2016.

[38] DUKKIPATI, N., REFICE, T., CHENG, Y., CHU, J., HERBERT, T., AGARWAL, A., JAIN, A., AND SUTIN, N. An argument for increasing tcp's initial congestion window. *SIGCOMM Comput. Commun. Rev. 40*, 3 (jun 2010), 26–33.

[39] ETSI. Multi-access edge computing (mec); radio network information api. In *GS MEC 012 V2.1.1* (2019).

[40] ETSI. Multi-access edge computing (mec); framework and reference architecture, etsi gs mec 003 v3.1.10. *ETSI ISG* (2022).

[41] FAHMY, S., AND KARWA, T. P. Tcp congestion control : overview and survey of ongoing research.

[42] FILIPPOU, M. C., SABELLA, D., EMARA, M., PRABHAKARAN, S., SHI, Y., BIAN, B., AND RAO, A. Multi-access edge computing : A comparative analysis of 5g system deployments and service consumption locality variants. *IEEE Communications Standards Magazine 4*, 2 (2020), 32–39.

[43] FLOYD, S. Tcp and explicit congestion notification. *SIGCOMM Comput. Commun. Rev. 24*, 5 (oct 1994), 8–23.

[44] FLOYD, S., MAHDAVI, J., MATHIS, M., AND ROMANOW, D. A. TCP Selective Acknowledgment Options. RFC 2018, Oct. 1996.

[45] FOUKAS, X., ET AL. FlexRAN : A Flexible and Programmable Platform for Software-Defined Radio Access Networks. In *ACM CoNEXT* (2016).

[46] FU, C. P., ET AL. TCP Veno : TCP enhancement for transmission over wireless access networks. *IEEE JSAC 21*, 2 (2003).

[47] GETTYS, J. Bufferbloat : Dark buffers in the internet. *IEEE Internet Computing 15*, 3 (2011), 96–96.

[48] GIUST, F., VERIN, G., ANTEVSKI, K., CHOU, J., FANG, Y., FEATHERSTONE, W., FONTES, F., FRYDMAN, D., LI, A., MANZALINI, A., ET AL. Mec deployments in 4g and evolution towards 5g. *ETSI White paper 24*, 2018 (2018), 1–24.

[49] GIUST, F., VERIN, G., ANTEVSKI, K., CHOU, J., FANG, Y., FEATHERSTONE, W., FONTES, F., FRYDMAN, D., LI, A., MANZALINI, A., ET AL. Mec deployments in 4g and evolution towards 5g. *ETSI White paper 24*, 2018 (2018), 1–24.

[50] GOYAL, P., NARAYAN, A., CANGIALOSI, F., NARAYANA, S., ALIZADEH, M., AND BALAKRISHNAN, H. Elasticity detection : A building block for internet congestion control. In *Proceedings of the ACM SIGCOMM 2022 Conference* (New York, NY, USA, 2022), SIGCOMM '22, Association for Computing Machinery, p. 158–176.

[51] GUO, Y., QIAN, F., CHEN, Q. A., MAO, Z. M., AND SEN, S. Understanding on-device bufferbloat for cellular upload. In *Proceedings of the 2016 Internet Measurement*

Conference (New York, NY, USA, 2016), IMC '16, Association for Computing Machinery, p. 303–317.

[52] GURTOV, A., HENDERSON, T., FLOYD, S., AND NISHIDA, Y. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582, Apr. 2012.

[53] HA, S., AND RHEE, I. Taming the elephants : New tcp slow start. *Computer Networks 55*, 9 (2011), 2092–2110.

[54] HAILE, H., ET AL. End-to-end congestion control approaches for high throughput and low delay in 4G/5G cellular networks. *Computer Networks 186* (2021).

[55] HENDERSON, T. R., ET AL. Network simulations with the ns-3 simulator. In *ACM SIGCOMM demo* (2008), vol. 14.

[56] HOCK, M., ET AL. Experimental evaluation of BBR congestion control. In *IEEE ICNP* (2017), pp. 1–10.

[57] HONG, S., BRAND, J., CHOI, J. I., JAIN, M., MEHLMAN, J., KATTI, S., AND LEVIS, P. Applications of self-interference cancellation in 5g and beyond. *IEEE Communications Magazine 52*, 2 (2014), 114–121.

[58] HUANG, J., ET AL. An In-depth Study of LTE : Effect of Network Protocol and Application Behavior on Performance. *ACM SIGCOMM* (2013).

[59] HUANG, J., QIAN, F., GUO, Y., ZHOU, Y., XU, Q., MAO, Z. M., SEN, S., AND SPATSCHECK, O. An in-depth study of lte : Effect of network protocol and application behavior on performance. *SIGCOMM Comput. Commun. Rev. 43*, 4 (aug 2013), 363–374.

[60] IRAZABAL, M., LOPEZ-AGUILERA, E., AND DEMIRKOL, I. Active queue management as quality of service enabler for 5g networks. In *2019 European Conference on Networks and Communications (EuCNC)* (2019), pp. 421–426.

[61] IRAZABAL, M., LOPEZ-AGUILERA, E., DEMIRKOL, I., AND NIKAEIN, N. Dynamic buffer sizing and pacing as enablers of 5g low-latency services. *IEEE Transactions on Mobile Computing 21*, 3 (2022), 926–939.

[62] IRAZABAL BENGOA, M. Enhanced quality of service mechanisms for 5g networks.

[63] JAIN, A., ET AL. Mobile throughput guidance inband signaling protocol. pp. 1–16.

[64] JAIN, R., ET AL. A quantitative measure of fairness and discrimination for resource allocation in shared computer systems. *DEC Research Report TR-301* (1984).

[65] JAIN, S., AND RAINA, G. An experimental evaluation of cubic tcp in a small buffer regime. In *2011 National Conference on Communications (NCC)* (2011), pp. 1–5.

[66] JIANG, H., ET AL. Tackling bufferbloat in 3G/4G networks. In *ACM IMC* (2012), pp. 329–342.

[67] JIANG, H., LIU, Z., WANG, Y., LEE, K., AND RHEE, I. Understanding bufferbloat in cellular networks. In *Proceedings of the 2012 ACM SIGCOMM Workshop on Cellular Networks : Operations, Challenges, and Future Design* (New York, NY, USA, 2012), CellNet '12, Association for Computing Machinery, p. 1–6.

[68] JOHNSON, T., ET AL. Desktop and mobile web page comparison : characteristics, trends, and implications. *IEEE Communications Magazine 52*, 9 (2014).

[69] KASSIM, M., RAHMAN, R. A., AZIZ, M. A. A., IDRIS, A., AND YUSOF, M. I. Performance analysis of voip over 3g and 4g lte network. In *2017 International Conference on Electrical, Electronics and System Engineering (ICEESE)* (2017), pp. 37–41.

[70] KEKKI, S., FEATHERSTONE, W., FANG, Y., KUURE, P., LI, A., RANJAN, A., PURKAYASTHA, D., JIANGPING, F., FRYDMAN, D., VERIN, G., ET AL. Mec in 5g networks. *ETSI white paper 28*, 28 (2018), 1–28.

[71] KHIRALLAH, C., ET AL. Design of Bandwidth and Energy Efficient Video Broadcasting Services over LTE/LTE-A. In *IEEE WCNC* (April 2013).

[72] KIM, T., KIM, Y., LIN, Q., SUN, F., FU, J., KIM, Y., PAPASAKELLARIOU, A., JI, H., AND LEE, J. Evolution of power saving technologies for 5g new radio. *IEEE Access 8* (2020), 198912–198924.

[73] KUMAR, R., ET AL. TCP BBR for Ultra-Low Latency Networking : Challenges, Analysis, and Solutions. In *IFIP Networking* (2019), pp. 1–9.

[74] LANGLEY, A., ET AL. The QUIC Transport Protocol : Design and Internet-Scale Deployment. In *Proc. ACM SIGCOMM* (2017).

[75] LANGLEY, A., RIDDOCH, A., WILK, A., VICENTE, A., KRASIC, C., ZHANG, D., YANG, F., KOURANOV, F., SWETT, I., IYENGAR, J., ET AL. The quic transport protocol : Design and internet-scale deployment. In *Proceedings of the conference of the ACM special interest group on data communication* (2017), pp. 183–196.

[76] LEE, G., ET AL. Simulation study of TCP proxy in multi-connectivity enabled 5G mmWave network. In *ICTC 2019* (2019), pp. 865–869.

[77] LORINCZ, J., KLARIN, Z., AND OŽEGOVIĆ, J. A comprehensive overview of tcp congestion control in 5g networks : Research challenges and future perspectives. *Sensors 21*, 13 (2021), 4510.

[78] LU, F., DU, H., JAIN, A., VOELKER, G. M., SNOEREN, A. C., AND TERZIS, A. Cqic : Revisiting cross-layer congestion control for cellular networks. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications* (New York, NY, USA, 2015), HotMobile '15, Association for Computing Machinery, p. 45–50.

[79] MASCOLO, S., ET AL. TCP Westwood : Bandwidth Estimation for Enhanced Transport over Wireless Links. In *ACM MOBICsOM* (New York, NY, USA, 2001), p. 287–297.

[80] METSÄLÄ, E., AND SALMELIN, J. *Planning and Optimizing Mobile Backhaul for LTE*. 2015, pp. 129–237.

[81] MISHRA, A., ET AL. The great internet TCP congestion control census. *ACM POMACS 3*, 3 (2019), 1–24.

[82] MISHRA, A., SUN, X., JAIN, A., PANDE, S., JOSHI, R., AND LEONG, B. The great internet tcp congestion control census. *Proc. ACM Meas. Anal. Comput. Syst. 3*, 3 (dec 2019).

[83] MONGKOLLUKSAMEE, S., VISOOTTIVISETH, V., AND FUKUDA, K. Enhancing the performance of mobile traffic identification with communication patterns. In *2015 IEEE 39th Annual Computer Software and Applications Conference* (2015), vol. 2, pp. 336–345.

[84] MONIKANDAN, S., ET AL. A review of mac scheduling algorithms in lte system. *Int. J. Adv. Sci. Eng. Inf. Technol 3* (2017), 1056–1068.

[85] NAH, F. F.-H. A study on tolerable waiting time : how long are web users willing to wait ? *Behaviour & Information Technology 23*, 3 (2004), 153–163.

[86] NICHOLS, K., AND JACOBSON, V. Controlling queue delay. *Commun. ACM 55*, 7 (jul 2012), 42–50.

[87] NURMINEN, J. K. Parallel connections and their effect on the battery consumption of a mobile phone. In *IEEE CCNC 2010* (2010), pp. 1–5.

[88] OUEIS, J., AND STRINATI, E. C. Uplink traffic in future mobile networks : Pulling the alarm. In *International Conference on Cognitive Radio Oriented Wireless Networks* (2016), Springer, pp. 583–593.

[89] PANDE, A., AHUJA, V., SIVARAJ, R., BAIK, E., AND MOHAPATRA, P. Video delivery challenges and opportunities in 4g networks. *IEEE MultiMedia 20*, 3 (2013), 88–94.

[90] PARK, H.-S., LEE, J.-Y., AND KIM, B.-C. Tcp performance issues in lte networks. In *ICTC 2011* (2011), pp. 493–496.

[91] PARK, M., ET AL. Analyzing effect of loss differentiation algorithms on improving TCP performance. In *Proc of ICACT* (2010), vol. 1.

[92] PARK, S., LEE, J., KIM, J., LEE, J., HA, S., AND LEE, K. Exll : An extremely low-latency congestion control for mobile cellular networks. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies* (New York, NY, USA, 2018), CoNEXT '18, Association for Computing Machinery, p. 307–319.

[93] PIRAUX, M., ET AL. Observing the Evolution of QUIC Implementations. In *Workshop on the EPCI of QUIC* (New York, NY, USA, 2018).

[94] POLESE, M., ET AL. milliProxy : A TCP proxy architecture for 5G mmWave cellular systems. In *ACSSC 2017* (2017), pp. 951–957.

[95] RAMAKRISHNAN, K., FLOYD, S., AND BLACK, D. Rfc3168 : The addition of explicit congestion notification (ecn) to ip, 2001.

[96] ROBERT, R., ET AL. Behaviour of common TCP variants over LTE. In *GLOBECOM 2016* (2016), pp. 1–7.

[97] ROMAN, K., ET AL. An accurate approximation of resource request distributions in millimeter wave 3gpp new radio systems. In *Internet of Things, Smart Spaces, and Next Generation Networks and Systems* (2019).

[98] RÜTH, J., KUNZE, I., AND HOHLFELD, O. Tcp's initial window—deployment in the wild and its impact on performance. *IEEE Transactions on Network and Service Management 16*, 2 (2019), 389–402.

[99] SABELLA, D., ET AL. A Hierarchical MEC Architecture : Experimenting the RAVEN Use-Case. In *IEEE VTC Spring* (2018).

[100] SABELLA, D., SUKHOMLINOV, V., TRANG, L., KEKKI, S., PAGLIERANI, P., ROSSBACH, R., LI, X., FANG, Y., DRUTA, D., GIUST, F., ET AL. Developing software for multi-access edge computing. *ETSI white paper 20* (2019), 1–38.

[101] SAMARAWEERA, N. K. G. Non-congestion packet loss detection for TCP error recovery using wireless links. *IEE Proc. Com.* (1999).

[102] SONG, Y.-J., KIM, G.-H., AND CHO, Y.-Z. Bbr-cws : Improving the inter-protocol fairness of bbr. *Electronics 9*, 5 (2020), 862.

[103] SUNDARESAN, S., DE DONATO, W., FEAMSTER, N., TEIXEIRA, R., CRAWFORD, S., AND PESCAPÈ, A. Measuring home broadband performance. *Commun. ACM 55*, 11 (nov 2012), 100–109.

[104] WARE, R., ET AL. Beyond jain's fairness index : Setting the bar for the deployment of congestion control algorithms. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (New York, NY, USA, 2019), HotNets '19, Association for Computing Machinery, p. 17–24.

[105] WARE, R., MUKERJEE, M. K., SESHAN, S., AND SHERRY, J. Modeling bbr's interactions with loss-based congestion control. In *Proceedings of the internet measurement conference* (2019), pp. 137–143.

[106] WOLSING, K., ET AL. A performance perspective on web optimized protocol stacks : Tcp+tls+http/2 vs. quic. In *Proc. of ANR Workshop* (2019).

[107] XIE, Y., YI, F., AND JAMIESON, K. Pbe-cc : Congestion control via endpoint-centric, physical-layer bandwidth measurements. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (2020), pp. 451–464.

[108] ZHANG, M., ET AL. Transport layer performance in 5g mmwave cellular. In *2016 IEEE INFOCOM WKSHPS* (2016), pp. 730–735.

[109] ZHANG, M., ET AL. Will TCP work in mmWave 5G cellular networks ? *IEEE Communications Magazine 57*, 1 (2019), 65–71.

[110] ZHONG, Z., HAMCHAOUI, I., FERRIEUX, A., KHATOUN, R., AND SERHROUCHNI, A. Cdbe : A cooperative way to improve end-to-end congestion control in mobile network. In *2018 14th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)* (2018), pp. 216–223.

[111] ZHONG, Z., HAMCHAOUI, I., AND KHATOUN, R. Perils of using cqic in lte network and a quick fix with delayed ack. In *2018 15th IEEE Annual Consumer Communications & Networking Conference (CCNC)* (2018), IEEE, pp. 1–2.

Web Pages

[112] "Mobile internet usage worldwide," https://www.statista.com/topics/779/mobile-internet/#dossierKeyfigures.

[113] C. Huitema, "picoquic," 2018, Software. [Online]. Available : https://github.com/private-octopus/picoquic

[114] "Android statistics (2022)," https://www.businessofapps.com/data/android-statistics/, 2022.

[115] M. Diarra *et al.*, "SIGMA," 2022, Software. [Online]. Available : https://github.com/madi223/sigma

[116] ——, "MELD," 2020, Software. [Online]. Available : https://github.com/madi223/MELD

[117] ——, "RAPID," 2021. [Online]. Available : https://github.com/madi223/RAPID

[118] ——, "RAPID-Linux," 2022. [Online]. Available : https://github.com/madi223/RAPID-LINUX

List of figures

List of tables

Enhanced Transport-Layer Mechanisms for MEC-Assisted Cellular Networks

Résumé

La résolution des problèmes de la couche transport dans les réseaux cellulaires est toujours un sujet d'actualité, malgré le fait que de nombreuses solutions ont été proposées au cours des dernières décennies. Une des raisons qui pourrait expliquer ce phénomène, est la difficulté d'adoption des solutions proposées dans le monde réel car elles nécessitent soit des changements drastiques dans la pile protocolaire des réseaux cellulaires (au niveau de l'UE et/ou au niveau de la station de base), soit un effort de normalisation pour que les en-têtes TCP puissent inclure des informations radio. Une autre raison est le fait que la plupart de ces solutions ne sont efficaces que dans quelques scénarios et ne parviennent pas à resoudre les problèmes de couche transport considérés d'une manière générale ou à grande échelle. Inspiré par ces limitations et aussi par la tendance grandissante vers le traitement à la bordure des réseaux mobiles ou Multi-access Edge Computing (MEC), nous nous sommes donné comme objectif dans cette thèse de démontrer qu'un très grand nombre de problèmes de couche transport dans un réseau cellulaire avec nœuds de traitement en bordure du réseau (scénario MEC), peuvent être resolus sans nécessiter de modifications dans l'appareil de l'utilisateur final ou dans la pile protocolaire 4G/5G. À cette fin, nous proposons de nouvelles solutions d'optimisation de la couche de transport qui exploitent le service de collecte d'informations radio du MEC (nommé RNIS) et d'autres moyens fournis par l'environnement MEC pour améliorer certains mécanismes clés de la couche transport tels que, le mécanisme de demarrage lent ou slow start, le mécanisme d'évitement de congestion ou régime permanent, le mécanisme de détection des pertes, ou encore le mécanisme de contrôle de flux. Nous montrons l'efficacité de cette approche en proposant d'abord SIGMA, un algorithme de contrôle de congestion (CCA) orienté lien-montant (uplink) qui surpasse les performances de BBR et des CCAs existants en termes de débit et de délai ; puis nous proposons et implémentons MELD, un algorithme de discrimination de pertes capable d'améliorer jusqu'à 80% le débit des CCAs de type loss-based (i.e., les CCAs qui utilisent les pertes comme signaux de congestion) en cas de pertes aléatoires ; et enfin nous proposons RAPID, un proxy d'amélioration de performance (PEP) qui tient compte de l'environnement radio et qui permet de réduire les augmentations de délais de bout-en-bout de manière transparente par un facteur de 10 à 50 sans introduire une diminution de débit, et ceci, quel que soit le nombre ou le comportement des flux TCP partageant la file d'attente dédiée à l'utilisateur. Enfin, nous avons implementé les solutions MELD et RAPID sur Linux et avons validé leurs performances dans un réseau 4G basé sur OpenAirInterface et FlexRAN.

Mots-clés : Réseaux cellulaires, Multi-access Edge Computing, Algorithmes de Contrôle de Congestion, Problèmes de couche transport, Optimisation TCP, Service de collecte d'informations radio

Abstract

Addressing transport-layer issues in cellular networks is still a hot research topic, despite the fact that a wide range of solutions have been proposed over the last decades. One reason that could explain this phenomenon is the difficulties in the real-world adoption of the proposed solutions as they require either drastic changes in the cellular network protocol stack (in the UE and/or in the base station) or a standardization effort so that TCP headers can include radio information. Another reason is the fact that most of these solutions are effective only in a few scenarios and fail to mitigate the transport-layer issues considered on a global or large scale. Inspired by these limitations and the growing trend towards Mobile and Multi-Access Edge Computing (MEC), we set out in this thesis to demonstrate that several transport-layer issues in a MEC-enabled 4G/5G network can be mitigated without requiring any modifications in the end-user's device or in the 4G/5G stack. To this end, we propose novel transport-layer optimization solutions that leverage the MEC Radio Network Information Service (RNIS) and other MEC capabilities to improve some key traditional transport-layer mechanisms such as slow start, steady state behavior, loss detection and flow control mechanisms. We show the efficiency of this approach by first proposing SIGMA, an uplink-oriented Congestion Control Algorithms (CCA) that outperforms BBR and existing CCAs in terms of throughput and delay ; then we propose and implement MELD, a loss discrimination algorithm that can improve the throughput of loss-based CCAs by up to 80% in case of wireless/random losses ; and lastly we propose RAPID, a RAN-aware Performance Enhancing Proxy (PEP) capable of transparently reducing the increase in delay by a factor of 10 to 50 without lowering the throughput, regardless of the number or the behaviors of the TCP flows sharing the same per-user buffer. Finally, we implement MELD and RAPID in Linux and validate their performance in a real-world 4G network based on OpenAirInterface and FlexRAN.

Keywords : Cellular networks, Multi-access Edge Computing, Congestion Control Algorithms, Transport-layer issues, TCP optimization, Radio Network Information Service