Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Electrical, electronics and Communications Engineering
(36^{th}cycle)

Merging Passive and Active Measurements: Machine Learning for Network Monitoring

Gianluca Perna

******

Supervisor:
Prof. Michela Meo, Ph.D

Doctoral Examination Committee:
Pedro Casas, Ph.D, Austrian Institute of Technology, Vienna
Matteo Varvello, Ph.D, Nokia Bell Labs, New Jersey
Anna Brunström, Ph.D, Karlstad University, Sweden
Fulvio Giovanni Ottavio Risso, Ph.D., Politecnico di Torino, Turin
Daniele Apiletti, Ph.D, Politecnico di Torino, Turin

Politecnico di Torino
2024

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Gianluca Perna
2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Acknowledgements

First and foremost, I would like to begin this long list of acknowledgments by mentioning my tutor and mentor, Michela Meo. Your immense technical expertise, support over the years, advice, kindness, and calmness in facing every situation have been fundamental parts of my journey. I am grateful to have known you, and I feel extremely fortunate to have been able to follow this path with you. Equally important is Marco Mellia, a wonderful, charismatic, and engaging person; together, you both have never ceased to support and encourage young people. You and Michela are truly remarkable individuals.

To Pedro Casas and Matteo Varvello, I had the pleasure of meeting you around the world, listening to your presentations, and learning from your work. You are extremely knowledgeable individuals, and I am happy to have had you as reviewers of my work. Your dedication, competence, and commitment in evaluating my work have been invaluable contributions to its overall improvement. Your observations and advice have been crucial in elevating the quality of my research. Thank you wholeheartedly for your invaluable contribution and professionalism. Yet, thanks to Daniele Apiletti, Anna Brunström and Fulvio Risso, for agreeing to serve on my committee. I am honored to be able to share my work with experts of your caliber.

Martino Trevisan and Idilio Drago, perhaps you are the people with whom I have shared the most in these years, thanks also to the Cloud Management experience that brought us together. I believe it is now well known even by the walls of our offices how much esteem and affection I have for you both. You are individuals with extraordinary technical knowledge and dedication, beyond common, you have enriched me immensely both professionally and personally. There are no words to express the enormous gratitude I feel towards you both, thank you from the bottom of my heart.

Danilo Giordano and Luca Vassio, two pillars of SmartData, thank you for always being present, for being not only colleagues but also friends. For always having the right advice at the right moment. You have made my stay in SmartData better, you are valuable resources and people. A huge thank you also to professors Maurizio Matteo Munafò and Paolo Garza, we have worked so much together over the years and I am happy to have been able to share much of my academic journey with you.

Dena Markudova, my doctorate may not have started without you. Known for a thesis, then colleagues, and now in search of our place in the world. You have been my first colleague, first friend, first advisor; you have been a fundamental piece in my doctorate, your laughter has cheered us up in moments of discouragement. You are a precious friend, and above all, a talented researcher, I wish you all the best.

To all members of SmartData, Daniel, Francesca, Matteo, Luca G., Rodolfo, Tailai, PHIL, Luciano, Francesco, Giordano, Nikhil, and all those I have probably forgotten, you are excellent colleagues, great friends, poor trashball players with whom I have shared many adventures. Never lose the enthusiasm that unites you; you are a force of nature, and I wish you all the best!

To lifelong friends Federico, Alfredo, Giulio, Stefano, and Gianluca Z., grown up together in a small provincial town, now adults, many of you fathers. I can tell you that the time spent with you was magnificent, you have brightened my days, we laughed, played, and joked together, and yet, we got angry together, I would repeat every single moment spent with you. I hope the future holds only the best for all of you.

To friends found by chance at university and never separated, Alessio, Federico, Gianmarco, and Riccardo, for you, neither words nor praise are needed, there is only one letter to describe the immense bond that binds us: J.

To my family and to those watching over us from above, life has not always been kind, but you have given me the strength to reach this milestone. There is no sufficient way to thank the family; you have done so much, perhaps too much. I am proud of the teachings I have received from you, and I am proud of the people you are, I could not have imagined a better family. Thank you very much.

Finally, the most important thanks go to Chiara, my girlfriend, the person with whom I have shared everything; you are the cornerstone of my life. You have always been by my side, supporting me in all my choices, bearing with me, listening to

me, and helping me. Infinite thanks would not be enough for you. You are among the best people I have ever met, always kind and helpful to everyone; people like you make this planet a better place. I am proud of the person you are; you are truly special, may life only give us the best.

Abstract

In an era where the internet permeates every facet of life, from essential services to daily entertainment, the need for a deep understanding and efficient management of its infrastructure is undeniable. This thesis delves into how the fusion of Machine Learning (ML) with active and passive network monitoring techniques can significantly enhance the robustness and reliability of network infrastructures. By focusing on distinct but interconnected domains such as Cloud Gaming, the HTTP/3 protocol, Geostationary Earth Orbit (GEO) Satellite Communication (SatCom), and Real-Time Communication (RTC) traffic, we offer insights into optimizing network performance for a variety of internet applications.

Our research begins with an in-depth analysis of Cloud Gaming platforms, including Google Stadia, NVIDIA GeForce Now, and PlayStation Now. We examine their unique network requirements, protocol usage, and the challenges of delivering high-quality gaming experiences also over mobile networks.

Turning our attention to the HTTP/3 protocol, we evaluate its adoption trends and operational benefits. Our findings highlight HTTP/3's role in enhancing web browsing experiences through reduced latency and improved efficiency, particularly in mobile environments. However, we also identify the protocol's varied performance across different hosting setups and its limitations in scenarios with high packet loss, underscoring the complexity of its deployment.

A further advancement in our research is the creation of *Retina*, software tool tailored to simplify the feature extraction process for analyzing SRTP traffic, which is widely used in videoconferencing applications. *Retina* streamlines the extraction of comprehensive features from SRTP streams, laying the groundwork for the creation of a Machine Learning (ML) model. We showcase consequently the application of this ML model, which excels at predicting the type of multimedia content transmitted within an SRTP flow. Furthermore, we explore the potential to adapt this model

to similar scenarios by employing transfer learning techniques, demonstrating its versatility and applicability.

To wrap up our research, we delve into the Satellite Communication (SatCom) domain, where we are granted the unique opportunity to analyze the entirety of traffic managed by a SatCom provider. Our initial efforts focus on conducting an extensive measurement and performance campaign, during which we examine the dynamics of traffic over different months. This comprehensive analysis provides insight into how traffic patterns evolve over time in different countries. Following this, we develop a system aimed at estimating the Web Quality of Experience (QoE) for SatCom operators, leveraging both active and passive measurements. This innovative system confronts the complexities introduced by Performance Enhancing Proxies (PEPs) and the ever-changing nature of web content. Utilizing Machine Learning (ML) algorithms to process the extracted features, we establish correlations between network characteristics and key QoE metrics, such as SpeedIndex and OnLoad. This approach highlights the critical role of ongoing adaptation in ML models to maintain consistent performance in the face of network variability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the modern era, it is hard to imagine life without the Internet. More than half of the global population is connected[1]. Digital networks, much like the human nervous system, play a crucial role in our communication, work, and entertainment, enabling the constant flow of data between individuals, systems, and applications.

However, extensive networks necessitate astute management and active surveillance to ensure their smooth and reliable operation. The importance of meticulous network monitoring cannot be underestimated, as it provides a critical window into the health and performance of the networks themselves. This raises the question: how are these invaluable insights obtained? And this is where active and passive measurements combined with Machine Learning models find application.

Active measurements involve generating specific test signals, such as data packets or more in general guided experiments, and closely observing/measuring their responses. This proactive approach is similar to a doctor evaluating the body's reaction to medication. Passive measurements, on the other hand, rely on observing traffic without actively interfering. This approach is similar to diagnosing a clinical picture based on visible symptoms and signs. Through both active and passive monitoring, we can detect and mitigate congestion, preempt service disruptions, and optimize resource allocation. Yet, in an increasingly complex and interconnected world, the volume of data produced by networks often surpasses our human analysis capacity.

[1] As of October 2023, there were 5.3 billion internet users worldwide, which amounted to 65.7% of the global population.

This is where artificial intelligence, particularly Machine Learning, comes into play. Applying Machine Learning algorithms to network data analysis can unveil hidden patterns and trends, offering deeper insights into user Quality of Service and Experience (QoS-QoE) and network performance. Through predictive models, we can anticipate anomalies and emerging issues, proactively acting to ensure more reliable and high-performing networks.

This thesis aims to show how the combination of active and passive measurements with Machine Learning contributes to developing models for rapid network analysis and monitoring. This work stems from a collaboration with two industry leaders, *Cisco Systems Inc.* and *Eutelsat*. We explore various scenarios and address different issues within the broader context of network monitoring. Working synergistically with these companies, we focused on developing new models for active network traffic monitoring based on Machine Learning.

Our initial focus was on video conferencing applications, in particular, the aim was to establish a comprehensive framework for assessing the Quality of Experience (QoE) of users involved in Real-Time Communication (RTC) applications. By analyzing extensive amounts of RTP traffic, we developed a Machine Learning model capable of categorizing the content of encrypted RTP streams into various media types. Consequently, the project resulted in the publication of several papers, some of which will be discussed in this thesis, particularly in Chapters 6 through 7.

Subsequently, our focus shifted towards other projects, with an emphasis on the intricacies of satellite operations. Here, the utilization of both active and passive network measurements assumed a more complex role. While still dealing with network statistics, the satellite environment, characterized by its high latency, significantly complicated all analyses. In this context, our aim was to develop a model capable of assessing user navigation quality using web browser-generated traffic. Despite previous efforts in this area, it became evident that the satellite domain had remained relatively unexplored. Our research endeavors sought to address this gap, presenting an initial exploration into this network scenario as detailed in Chapters 5-8.

1.1 Thesis Outline

In this section, we outline the thesis structure and the main contributions of each chapter.

In Chapter 2, we discuss all the essential technical knowledge required to engage with the content of this thesis. This includes an explanation of the differences between active and passive measurements, along with an exploration of network protocols such as RTP and HTTP/3. We also delve into the rationale and motivations driving this thesis, as well as its unique contributions.

In Chapter 3, we discuss the work concerning network monitoring and measurements for Cloud Gaming applications. In recent years, these applications have gained significant popularity, facilitated by the widespread availability of broadband connections, allowing a wide audience to make use of such services. We analyze the type of protocol employed by the main providers, as well as the minimum and maximum requirements needed to fully enjoy these services.

In Chapter 4, we delve into the comprehensive role of the new Hypertext Transfer Protocol 3 (HTTP/3). HTTP/3, along with its predecessors HTTP/2 and HTTP/1, is used to access the majority of services on the Internet, ranging from websites to social networks and collaborative platforms. We present an extensive measurement study of HTTP/3 adoption and performance to quantify the advantages ushered in by HTTP/3 in various scenarios, including browsing, mobile usage, and adaptive video streaming.

In Chapter 5, we present the first large-scale passive characterization of a global GEO Satellite Communication (SatCom) Internet access solution. Through passive instrumentation of the satellite ground station, we observe traffic from tens of thousands of customers in more than 20 countries in Europe and Africa. This allows us to characterize different Internet usage habits in various scenarios and to observe the impact of SatCom technology on performance.

Chapter 6 introduces *Retina*, an open-source tool designed for the versatile analysis of Real-Time Communication (RTC) traffic. By providing raw captures of RTC traffic, *Retina* generates a range of logs and graphs based on user-defined parameters. Among these is a log detailing traffic statistics on a per-second basis, which serves as input for the Machine Learning algorithms discussed in Chapter 7.

Chapter 7 introduces a Machine Learning classifier that leverages statistical attributes derived from the traffic data generated by *Retina* (Chapter 6). This classifier is designed to distinguish RTP streams based on the type of media content they transport, such as audio, video, screen sharing, and more. It operates on one-second traffic segments as input, and in addition to standard media, it can also identify various video qualities and error correction streams.

Chapter 8 delves into the nature of connections in a SatCom scenario already presented in Chapter 5, with a particular focus on assessing Quality of Experience (QoE) in browsing through Machine Learning in the satellite scenario. This investigation involves the utilization of three months' worth of both active and passive data. The active probe is positioned upstream of the satellite apparatus, before the ground receiving antenna. Passive data is collected on the segment connecting the operator's DataCenter to the Internet using the Tstat software. The goal is to develop a Machine Learning model capable of predicting QoE metrics in browsing, such as OnLoad or SpeedIndex, with the aim of bridging the worlds of active and passive measurements thanks to Machine Learning. Additionally, we examine the critical aspects of the model, such as its long-term adaptability, and present solutions to address these challenges.

1.2 List of Publications

In this section, we outline the papers published during the PhD, dividing them into two subsets: those relevant for the thesis and additional works.

Publications described in the thesis:

1. **A Network Analysis on Cloud Gaming: Stadia, GeForce Now and PSNow**, Andrea Di Domenico; Gianluca Perna; Martino Trevisan; Luca Vassio; Danilo Giordano; (2021) In: Network*Chapter 3: journal version*

2. **A first look at HTTP/3 adoption and performance**, Gianluca, Perna; Martino, Trevisan; Danilo, Giordano; Idilio, Drago; (2022) In: Computer Communications *Chapter 4: journal version*

3. **When Satellite is All You Have: Watching the Internet from 550 Ms**, Daniel Perdices; Gianluca Perna; Martino Trevisan; Danilo Giordano; Marco Mellia; (2022) In: Proceedings of the 22nd ACM Internet Measurement Conference (IMC) *Chapter 5: conference version*

4. **Retina: An open-source tool for flexible analysis of RTC traffic**, Perna, Gianluca; Markudova, Dena; Trevisan, Martino; Garza, Paolo; Meo, Michela; Munafò, Maurizio, (2022) In: Computer Networks 202 - *Chapter 6*

5. **Online Classification of RTC Traffic**, Perna, Gianluca; Markudova, Dena; Trevisan, Martino; Garza, Paolo; Meo, Michela; Munafò, Maurizio; Carofiglio, Giovanna, (2020) In: 2020 IEEE 18th Annual Consumer Communications and Networking Conference (CCNC) - *Chapter 7: conference version*

6. **Real-Time Classification of Real-Time Communications**, Perna, Gianluca; Markudova, Dena; Trevisan, Martino; Garza, Paolo; Meo, Michela; Munafo, Maurizio; Carofiglio, Giovanna, (2022) In: IEEE Transactions on Network and Service Management - *Chapter 7: journal version*

7. **Monitoring Web QoE in Satellite Networks from Passive Measurements**, Gianluca Perna; Martino Trevisan; Danilo Giordano; Marco Mellia; Daniel Perdices; (2023) In: 2023 IEEE Consumer Communications and Networking Conference 2023 - Conference (CCNC) *Chapter 8: conference version*

Other published works:

1. **Where did my packet go? Real-time prediction of losses in networks**, Tailai Song, Dena Markudova, Gianluca Perna, Michela Meo; In: IEEE 2023 International Conference on Communications (ICC), conference version

2. **BitFormer: Transformer-based Neural Network for Bitrate Prediction in Real-Time Communications**, Tailai Song; Gianluca Perna; Michela Meo; Paolo Garza; Maurizio Matteo Munafò; In: IEEE 2023 Consumer Communications and Networking Conference (CCNC), conference version

3. **Realistic Testing of RTC Applications under Mobile Networks**, Gianluca, Perna; Martino, Trevisan; Danilo, Giordano; (2020) In: Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)

4. **Packet Loss in Real-Time Communications: Can ML Tame its Unpredictable Nature?**, Tailai Song; Gianluca Perna; Paolo Garza; Michela Meo; Maurizio Matteo Munafò; (Submitted to review stage), journal version

My career in the Networking field has been enriched by valuable collaborations with key industry players, including globally renowned companies like Cisco. These partnerships have provided me with a unique opportunity to share and contribute to significant research projects. Throughout these experiences, I have had the privilege to work with exceptional colleagues, sharing a common vision and implementing a synergistic approach to teamwork. In my consistent dedication, I have always played a pivotal role, successfully balancing technical and theoretical aspects, thereby ensuring a meaningful contribution to the achieved results and the resulting publications.

1.3 Open Source - Code and Dataset

In line with our commitment to open science, a significant portion of our work, including datasets and tools, is publicly available. Throughout the chapters, references for accessing this material are provided, and a summarized version is presented in Table 1.1.

Collecting data in research projects is complex, requiring significant time, automation, and meticulous processing and storage. Thanks to the open-sourcing of code and datasets, sometimes this step can be accelerated, offering several tangible benefits. Firstly, it allows fellow researchers to replicate and verify findings, thereby accelerating the pace of scientific discovery. Additionally, it facilitates the reuse of existing resources, saving time and resources that would otherwise be spent reinventing the wheel. Moreover, open access to code and datasets promotes educational opportunities, enabling students and aspiring researchers to learn from real-world examples and contribute to ongoing projects. As data emerges as the driving force of the century, accessibility further empowers technology development.

In conclusion, while acknowledging the occasional constraints imposed by proprietary considerations, researchers must remain steadfast in their commitment to open science. By making conscientious efforts to publish code and datasets, we contribute to a more inclusive, collaborative, and impactful scientific ecosystem, ultimately advancing the frontiers of knowledge for the betterment of society.

Chapter	**Open Resource Link**
3: Cloud Gaming	https://zenodo.org/records/5509243
4: HTTP/3 Benchmark	https://github.com/SmartData-Polito/h3-benchmark
5: SatCom Characterization	https://github.com/SmartData-Polito/errant
6: Retina	https://github.com/GianlucaPoliTo/Retina
6: Retina	https://hub.docker.com/r/gianlucapolito/retina
7: RTC Classification	https://smartdata.polito.it/rtc-classification/

Table 1.1 Open source resources of the thesis

Chapter 2

Background and Motivation

This chapter provides an in-depth examination of the literature regarding network analysis, active and passive measurements, and an inspection of detailed protocols such as RTP and HTTP/3. We will then see how machine learning can be employed to design a traffic monitoring or classification system, based on the data collected. We will describe the best-practices for setting up a testbed, focusing on the most widely used technologies now; next, we will analyze the benefits and critical issues in a data collection campaign.

2.1 Active and Passive Measurements

Since ancient times measuring quantities has been a human need, we measure weight, length forces, in fact we measure anything that can be measured, and the Internet is no exception. The Internet is a network of networks, each macro network is an Autonomous System (AS), and these are interconnected with each other and talk to each other, with a complex routing protocol called Border Gateway Protocol (BGP)[1]. The Internet, as we know it today, was not initially envisioned in its current form. Its success and adoption unfolded gradually, thanks to the development of software, protocols, and standards. Similar to urban infrastructure, long-term perspectives were often lacking in its early stages. This lack of foresight is evident in events like the unexpected exhaustion of IPv4 addresses, which few had foreseen. The Internet's evolution has been marked by rapid and sometimes chaotic growth. In

[1] BGP was developed at Stanford University, in the United States, in 1989.

response, organizations like the Internet Engineering Task Force (IETF) emerged to standardize its use. They navigate complexities, such as adapting new protocols to existing infrastructure, exemplified by the transition to IPv6.

Understanding the Internet's behavior can be challenging, which is why monitoring its performance and stability is of vital importance. What makes the Internet unique is its dynamic and ever-evolving nature. Measurements provide the necessary framework to understand how data travels through this intricate maze of connections. They allow us to assess congestion, latency, and the reliability of connections in real-time, crucial elements to ensure a seamless and uninterrupted user experience. Furthermore, anomaly detection is essential for identifying out-of-the-ordinary behavior, which could indicate security issues or malfunctions in the network. Thanks to these measurements, it is possible to intervene promptly, preventing potential crises or more severe problems. In addition to ensuring optimal operation, measurements provide valuable insights into how users utilize the network. Every interaction, every click, every download is captured and analyzed. This approach, known as user profiling, plays a crucial role in understanding what attracts users, which services are in higher demand, and how to enhance the offering. Ultimately, this information not only enables the provision of more targeted services but also helps businesses develop more effective marketing strategies, generating a positive impact on revenue and customer satisfaction. Furthermore, we cannot underestimate the importance of measurements in the realm of online security. The network is constantly exposed to external threats, such as cyberattacks and fraud attempts. Measurements allow for the rapid identification of these events, implementing the necessary countermeasurements to safeguard data integrity and user privacy. Without this level of vigilance and proactive response, the Internet would be much more vulnerable to attacks and intrusions. Consider also the significant impact of energy consumption on Internet operations. In an era where sustainability and green initiatives take center stage, monitoring becomes crucial in this context as well. It allows us to identify areas of resource wastage and enhance overall efficiency. The Internet, a voracious consumer of energy and data[2], accounts for approximately 2.5% of global electricity consumption. Projections indicate that this demand is set to double by 2030, underscoring the substantial resources required to maintain its operations.

[2]According to a recent study regarding Energy Consumption of the Internet this accounts for about 2.5% of all global electricity consumed. In addition, it is estimated that the energy demand of the Internet will double by 2030

After this introduction to the Internet measurement, we can now address another important question: *what is the difference between active and passive measurements, and what are they used for?*

Active measurements involve direct action by the experimenter on the network. For example, performing a ping to assess the RTT to a server or using a speedtest to determine the available upload and download bandwidth. In short, active measurements include all actions taken by the experimenter with the goal of obtaining value estimates. Passive measurements, in contrast, operate in the reverse way. This technique involves capturing and analyzing live network traffic, or traffic statistics, at a specific point in the network without interact directly with it.

Going deeper, active monitoring can and should be employed to provide a 360-degree, real-time view of service and network (QoS) conditions. Through proactive use, these tools can provide us with early warning of performance degradation, potentially even before the customer is aware of it. In addition, by having this level of visibility into most, if not all, services in the network, automated troubleshooting tools can quickly triage impacted services and identify hotspots or common cause elements to ensure that the most critical problems are addressed first. A practical example would be the use of a monitoring tool in a datacenter such as Zabbix[3], which among other functions offers the ability to perform ping tests at regular instants of time, on all machines in the datacenter, to check their functionality.

Passive monitoring serves a multifaceted role, exemplifying its versatility in both cybersecurity and operational realms. This methodology plays a pivotal role in post-event analysis by effectively identifying root causes and malicious traffic instances. Leveraging historical traffic profiles, it discerns anomalies such as Distributed Denial of Service (DDoS) attacks or high retransmission activity, facilitating prompt response strategies. Additionally, passive monitoring offers invaluable insights into customer usage patterns and application performance, enabling carriers to directly monitor Quality of Experience (QoE). This granular data empowers the development of tailored service packages, informs strategic network and system upgrades, and uncovers new service opportunities. For instance, within content distribution networks like Netflix, passive monitoring discerns popular content by geographic region, thereby guiding informed data placement strategies to optimize user experience [1] [2].

[3] https://www.zabbix.com/

In conclusion, it is crucial to understand that both active and passive monitoring play a critical role in network infrastructure analysis and data analysis, but above all that the two types of monitoring are a powerful tool when used together.

2.2 Real-Time Transport Protocol

To facilitate the reading of the thesis, in this section we provide an overview of the most common protocols used in Real Time Communication (RTC) applications and the difference between *native* applications (e.g., Webex Teams) and browser-based platforms (e.g., Jitsi Meet).

Despite the broad landscape of protocols used on the Internet, this section will provide specific details on Real-Time Transport Protocol (RTP) protocol [3] that will be frequently discussed in subsequent chapters, in particular in Chapter 3, Chapter 6 and Chapter 7.

Media streaming. RTP is used to transport multimedia content in real time. In the ISO/OSI structure, we imagine the transport layer divided into two parts: below we find UDP/TCP, while above we place RTP. This is because although RTP is designed for transporting multimedia data, it makes use of UDP for actual data delivery.

Why the choice of UDP? To understand the reasons behind this choice, it is useful to analyze the use cases. RTP, as mentioned, is employed in real-time data transmission scenarios, such as video conferencing, cloud-gaming, and real-time streaming. In these scenarios, the user values an immediate response over perfect information integrity. Think, for example, of a video call: if a packet were lost, waiting for its retransmission before playout would lead to significant delays, compromising the real-time experience.

For transporting multimedia content, RTP extends the capabilities of UDP by introducing the concept of a multimedia stream. Usually, there is an RTP stream for each type of active media (such as audio or video). However, there are cases where the service provider uses a central server to manipulate and recombine the streams into one, adapting the quality or codecs, and sends it to the receivers as a single stream. This represents a more specific case that will not be addressed here.

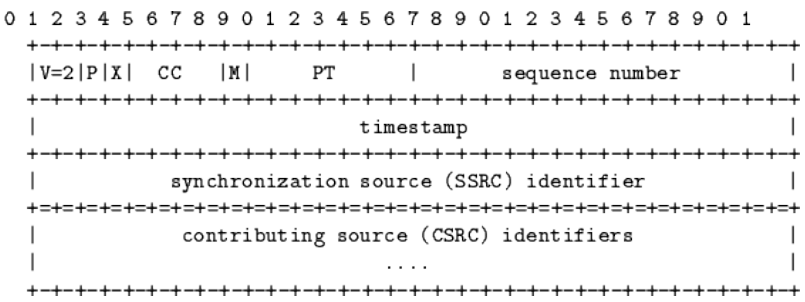


Fig. 2.1 RTP header

Let us analyze the RTP header, as shown in Figure 2.1, for a more detailed description of some of the fields.

- **SSRC (Synchronization Source)**: Identifies the synchronization source of the session, unique for each RTP stream.
- **PT (Payload Type)**: Specifies the encoding format of the data contained in the RTP packet.
- **Timestamp**: Indicates when the audio or video sample was taken.
- **Sequence Number**: Incremental numbering of packets to assist the receiver in reordering them.
- **CCSRC (Contributing Source)**: List of sources that contributed to composing an RTP packet.

Thus, among RTP's main functions, the use of timestamping stands out, which allows audio and video data streams to be properly synchronized during transmission. This timestamping facilitates the proper playback of received media elements, ensuring smooth and synchronized streaming. In addition, RTP implements a packet numbering mechanism, which makes it possible to detect the loss of data or its reception out of sequence. This is especially crucial in contexts where transmission reliability is paramount, such as in video conferencing or VoIP calls. In parallel, RTP works in synergy with another protocol, called RTP Control Protocol (RTCP). RTCP takes on the task of monitoring Quality of Service (QoS) during real-time data transmission. This means that RTCP collects information about network performance,

such as delay, packet loss and congestion, and provides this data to communication participants in order to perform Transmission Adaptation[4].

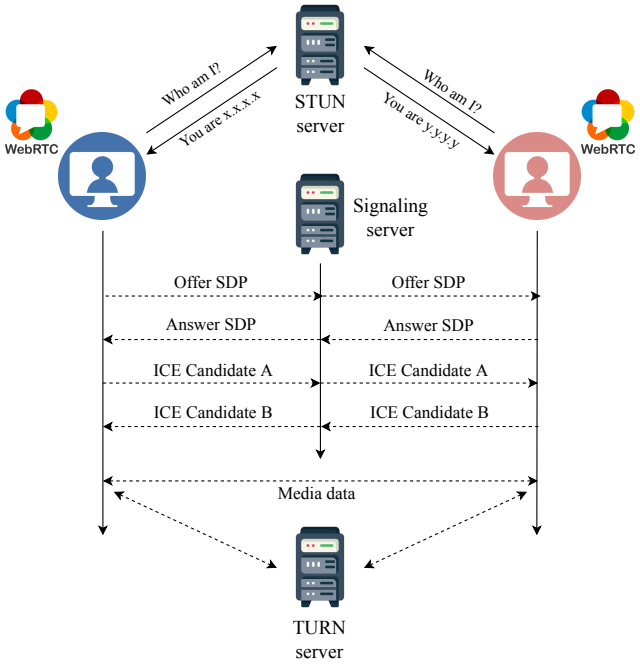


Fig. 2.2 RTC Session Setup

Session Setup. In order to start a media session, it is necessary for the endpoints to be able to communicate with each other, especially in the case of peer-to-peer communication between participants. This is complicated by the presence of NATs, firewalls and middleboxes in general. To ensure connectivity, applications often use the STUN protocol [4] for NAT detection and TURN [5] to relay the traffic through a server with a public IP address. ICE [6] combines STUN and TURN into a single technique. RFC 7983 [7] defines a simple mechanism for multiplexing RTP, STUN, and other protocols on the same UDP flow. An example of session setup in Real-Time communication is reported in Figure 2.2

WebRTC. The above protocols need to be carefully coordinated to have a working RTC application. To facilitate the development of new applications, WebRTC [8] is a set of high level and standardized APIs that can be used in browsers and mobile applications for video and audio communication. WebRTC was launched in 2011 and is currently supported by most browsers. It represents the standard way for RTC

[4]Participants can adjust transmission parameters to improve quality

applications to run via web if we exclude application-specific plugins. WebRTC provides programming interfaces to establish media sessions, organizing the use of the RTP, RTCP, STUN, TURN and DTLS protocols.

RTC Applications Under Study. In our study that will be presented in detail in Chapter 7, we focus on two RTC applications: (i) Cisco Webex Teams and (ii) Jitsi Meet. Webex Teams (or **Webex** for short) is a business-oriented service that offers paid plans for enterprises and institutions that require video call service. It is available as a standalone application for PC and mobile devices, but it can also be used through browsers that support the WebRTC standard. Jitsi Meet (or **Jitsi** for short) is a free of charge RTC application that provides a simple browser-based user interface for WebRTC-compliant browsers. It is fully open-source, and it is possible to run a private Jitsi server or rely on the public service available online. Both applications use RTP for streaming multimedia content along with STUN and TURN for session establishment. They support audio and video communication and allow users to share their screens with other participants. Moreover, they adopt the Selective Forwarding Unit (SFU) approach [9], where participants send their multimedia content to a central server. The server then forwards the data, deciding which stream to send to each participant. Although the choice of different RTC applications (e.g., Zoom or Microsoft Teams) would be possible, we opted for Webex and Jitsi, which allow us to easily gather the classification ground truth, as we will illustrate in Section 7.4. For other popular applications, we could not find such a convenient way to collect the needed information.

For the rest of the thesis, we use the following definition for RTP flow: *(IPsrc, IPdst, PrtSrc, PrtDst, SSRC, PT)*.

2.3 Evolution and History of HTTP

The Hypertext Transfer Protocol, known as HTTP, is the foundation of communication on the Web. It defines how clients, such as Web browsers, and servers exchange information. In the ISO/OSI framework, HTTP operates at the application layer (layer 7) in synergy with the TCP transport protocol, which acts at the transport layer (layer 4).

HTTP's ancestor, HTTP/0.9, was introduced in 1991 and allowed only the transfer of text-type data. However, this limitation highlighted the need for a more advanced version. In 1996, HTTP/1.0 was introduced. This version allows the transfer of a variety of media, but still has a significant drawback: it can send only one object per connection, without persistence. This implies that each request to a new object requires the opening of a new TCP connection, resulting in a significant expenditure of time and resources, since 2 RTTs will be required to get the data stream: 1 RTT for opening the connection and another for initiating the transfer as shown in Figure 2.3.

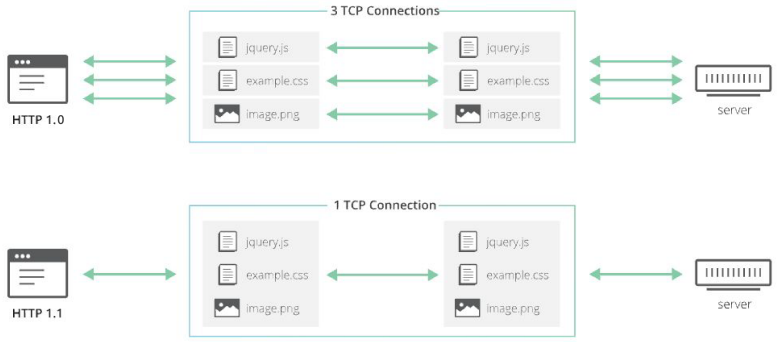


Fig. 2.3 Non Persistent vs Persistent Connection

In 1997, HTTP/1.1 [10] was released, introducing the concept of connection persistent, allowing multiple requests to share the same TCP connection, reusing the same socket, as depicted in Figure 2.3. In terms of scalability, this innovation is a significant advantage because the RTT for opening the connection occurs only for the first object, while the others are requested in the queue on the same socket.

Despite the implementation of persistent connections, HTTP/1.1 still faces a problem, known as "Head of Line Blocking" (HOL). This concept, can be explained by considering the principle of a TCP connection between client and server, i.e, as a direct thread through which one piece of information can travel at a time, in an orderly and sequential manner, as shown in Figure 2.4. If a packet is lost en route between the client and the server, this can block the connection. TCP, being of a lower level than HTTP, does not understand the details of the HTTP protocol and treats data as a stream of bytes. Therefore, if a packet is lost, the protocol waits for retransmission before proceeding.

HTTP/2 [11, 12], introduced in 2015, addressed "Head of Line Blocking" (HOL) by introducing "multiplexing," where the server and client can exchange multiple

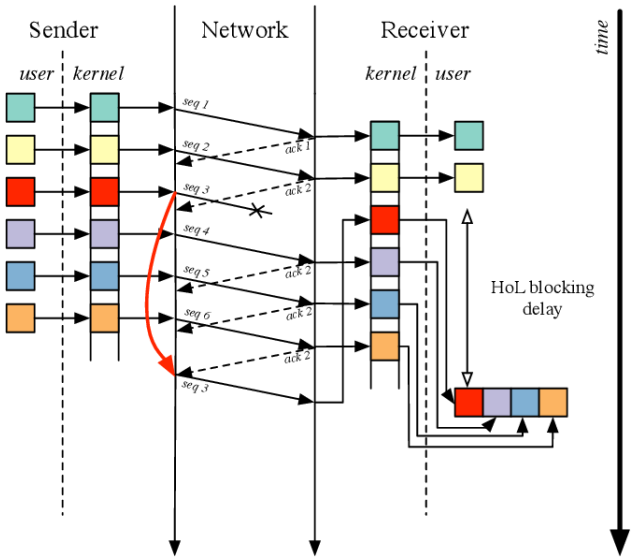


Fig. 2.4 Head of Line Blocking TCP

streams of data within the same connection. Each stream is identified by a unique identifier, allowing for parallel processing of requests and responses. This means that even if one resource is delayed or blocked, it won't prevent other resources from being transmitted, thus eliminating the bottleneck effect caused by HOL. Although HOL was resolved at the HTTP level with this mechanism, it persisted at the TCP level due to its sequential nature as reported in Figure 2.4.

This inefficiency of TCP for modern browsing led Google engineers to develop an alternative called QUIC (Quick UDP Internet Connection), which now makes up about 40% of Internet traffic.

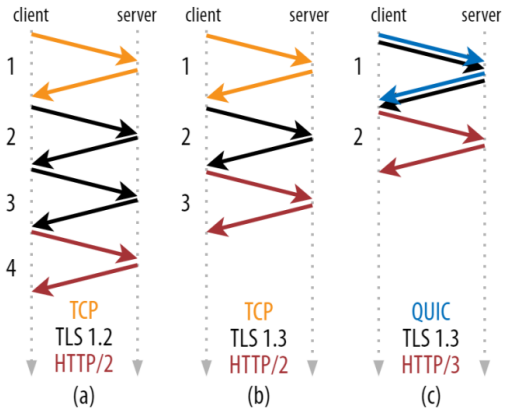


Fig. 2.5 HTTP/2 vs HTTP/3

Key features of QUIC include replacing the transport layer with UDP and implementing native transport of TLS+HTTP. This strategic shift addresses the problem of HOL at layer 4, as UDP doesn't mandate ordered delivery. Additionally, the entire control and retransmission mechanism, formerly managed by TCP at the kernel level, is now executed in user-space. This architectural adjustment offers a significant advantage in agility, allowing the protocol to be modified without necessitating intervention in the OS kernel, thereby simplifying the adoption and diffusion of the protocol and facilitating customization to develop tailored alternatives.

QUIC natively uses and supports TLS 1.3 [13], the latest iteration of the TLS protocol. In TLS 1.3 has been introduced the zero round trip time (O-RTT) connection resumption. This operational mode allows a client to commence transmitting application data, such as HTTP requests, without waiting for the TLS handshake to finalize as shown in Figure 2.5 (c). The fundamental concept behind 0-RTT connection resumption is that if the client and server had previously established a TLS connection between them, they can utilize cached information from that session to initiate a new one without the necessity of renegotiating the connection parameters from scratch. QUIC brings a significant advantage by consolidating the transport and cryptographic handshakes into one, thereby eliminating a complete round-trip from the typical connection establishment process. This innovation not only trims the handshake duration but goes a step further, achieving a true 0-RTT connection setup.

2.4 Metrics of QoS and QoE in Networking

Quality of Service (QoS) metrics in networking are key parameters used to evaluate and ensure the level of performance and reliability of a network. These quantitative measurements include indicators such as latency, packet loss, bandwidth and jitter, providing an objective assessment of data transmission performance. Latency, for example, indicates the delay in sending and receiving packets, while packet loss indicates the percentage of data that does not reach its destination. QoS is essential to ensure that time-sensitive applications, such as video conferencing or online gaming, run smoothly and without interruption. Optimizing these metrics allows to deliver a superior user experience and ensure that the network can handle varying traffic loads efficiently. Let's look in detail at what the most popular QoS metrics are.

- **Round-Trip Time (RTT):** Indicates the time it takes for a data packet to travel from sender to receiver and back again. It can affect applications such as online games.
- **Packet Loss:** Indicates the percentage of packets that do not reach their destination. Too much packet loss can cause delays and deterioration of communication quality.
- **Jitter:** Represents the variation of latency over time. High jitter can cause jitters or interruptions in communications, especially in time-sensitive applications.
- **Bandwidth:** Represents the amount of data that can be transmitted over a connection in a given time interval. High bandwidth is crucial to ensure a good user experience, especially in challenging environments such as Cloud Gaming.
- **Throughput:** Represents the actual amount of data that can be transmitted between sender and receiver. It may be less than the nominal bandwidth due to various factors.

All of the above metrics are QoS measurements, and they can all be obtained directly from the flows or packets transited in the network. For example, you can measure RTT using the ping command to different anchors[5] but also packet loss, or you can measure bandwidth using software such as speed-test, which allows you to open a TCP flow to a server trying to maximize the data sent per second. Jitter, on the other hand, can be estimated by extracting the time series of arrival values from the pcap file, calculating the variance on interarrival times. In short, for each of the metrics above there is a direct way to derive the quality measure.

The move from QoS to QoE marks a major shift in the evaluation and optimization of communications networks and services. This is happening because it has been recognized that even if a network is well managed technically, this does not guarantee a high-quality user experience.

An example clarifies this idea: imagine a perfectly paved road with clear signage and smooth traffic, representing a network with good QoS. However, if cars have

[5]They are servers whose location you know that respond to the ping thus giving us the round-trip time information

uncomfortable seats or unintuitive controls, the driving experience will be less pleasant for the driver. This explains the key point: the quality of the user interface, along with other factors, is critical in the total user experience, regardless of the robustness of the QoS.

Therefore, QoE becomes crucial. It combines the evaluation of technical network performance with subjective user feedback, providing a complete and accurate view of the quality of experience. For example, if a VoIP call has low latency (indicating good QoS), but the voice is distorted or unclear, the user will still perceive a low-quality experience.

Thus, attention to QoE is a natural step in the evolution of network design and management. It recognizes that user-perceived quality is as important as the technical performance of the network, and that to offer truly excellent service, both aspects must be considered in a balanced and synergistic way.

Examples of QoE metrics may include:

- **MOS (Mean Opinion Score):** A Mean Opinion Score quantifies the overall quality of an event or experience based on human judgment. In the realm of telecommunications, it specifically evaluates the quality of voice and video sessions.

  Typically assessed on a scale ranging from 1 (poor) to 5 (outstanding), Mean Opinion Scores represent the average of various individual parameters scored by human evaluators. While initially these scores originated from surveys conducted by expert observers, contemporary MOS is frequently generated through an Objective Measurement Method that approximates human assessments.

- **SpeedIndex:** The SpeedIndex measurements a user's perception of the speed at which a web page loads. This metric takes into account the time it takes to make the main content of a page visible, rather than the complete loading of all elements. A lower value indicates faster loading and a better user experience.

- **OnLoad:** OnLoad indicates the time when all elements of a web page have been fully loaded in the user's browser environment. A page that loads quickly contributes to a smoother user experience, avoiding delays and frustration.

- **Time to First Byte (TTFB):** This metric measurements the time between requesting a web page and receiving the first response byte from the server. A low TTFB indicates that the server responds quickly to requests, which can improve the user experience.
- **Time to Interactive (TTI):** This metric measurements the time it takes for a web page to become interactive, which is when the user can begin to interact with the content on the page. A low TTI indicates that the page becomes interactive quickly, which can improve the user experience.

2.5 General Testbed Setup

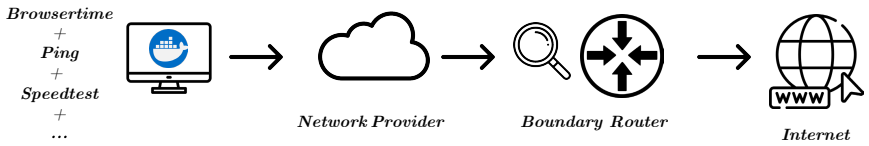


Fig. 2.6 General Testbed Setup

In this last section of the chapter, we present the guidelines that conducted the creation of our network testbeds. We will outline the most commonly used software, providing a generic model that can later be customized and adapted as needed.

Throughout the course of this thesis, the primary focus of the utilized testbeds has been on the collection of Quality of Experience (QoE) metrics, aimed at developing Machine Learning models capable of predicting these indicators. Notably, OnLoad and SpeedIndex are key measurements attainable through various methods, particularly via web browsers such as Google Chrome. By analyzing the .HAR file[6], which comprehensively records web request times, responses, and wait durations, these metrics can be extracted.

In pursuit of scalability, manual collection of thousands of browsing sessions proves impractical. Hence, a high degree of automation becomes essential. Fortunately, numerous tools have been developed over the years to emulate web browsing activities. Among them, Browsertime[7] stands out as one of the most reliable options

[6] https://toolbox.googleapps.com/apps/har_analyzer/?lang=it
[7] https://hub.docker.com/r/sitespeedio/browsertime/

available. This software, accessible through Docker, offers exceptional versatility, making it compatible with a diverse array of operating systems, including Linux, Mac, and Windows. A standout characteristic of this tool is its exceptional configurability. It not only facilitates browsing web pages while capturing both QoE metrics[8] and comprehensive .HAR file, but also offers the capability to integrate external Selenium-based configuration files. This feature empowers users to automate interactions within the web page, enabling tasks such as accessing forms or clicking on specific elements.

In Figure 2.6, we illustrate a typical Testbed setup, featuring a test machine connected to the internet via the operator under analysis. Various services are activated on the test machine to perform both active and passive measurements. Active measurements can include for instance periodic pinging towards known nodes to monitor connection status, along with bandwidth data collection every 15 minutes through a cronjob. Additionally, a continuously active script facilitates web page navigation through Browsertime. The selection of web pages to visit depends on the specific analysis context. Initially, sites providing insights into the top 100 visited domains in a particular region may be considered, although this list can be tailored to meet specific requirements. Recognizing the variability of services is crucial as they may vary depending on the analysis nature.

In a robust testbed configuration, it's essential to ensure measurements reflect real-world conditions, such as accepting cookies in case of browsing activity, mirroring typical user behavior on the internet. Users commonly accept cookies during their initial visit, and subsequent sessions are tied to these accepted cookies until expiration or clearance. It's also vital not to limit measurements solely to homepage interactions, as users frequently navigate deeper into websites like Amazon, YouTube, or news sites. Although this complexity may extend the data collection process, it ensures collected data accurately represents real user activity.

A key aspect of the testbed is passively intercepting network traffic from a strategic vantage point. This vantage point provides an advantageous position for observing network traffic, typically where traffic can be sniffed as it traverses the network cable, included the traffic generated by the active probe. In our scenarios, the passive analysis probe is always placed on the outgoing link of the boundary

[8]The HAR file generated as output by Browsertime is natively enriched with QoE metrics such as SpeedIndex and OnLoad, effectively making the collection of metrics simplified.

router of the provider, so that we can have total view on the transiting data. It is clear that this could pose privacy risks, because if not properly handled it could be easily traced back to the users who visited certain domains, but the software we use in this thesis: Tstat, is created in such a way as to anonymize IPs while still respecting the network masks. It is not possible in any way to reverse the IPs in the tstat logs to the real IPs. The data collected passively at this vantage point serves multiple purposes. Firstly, it is utilized in conjunction with data from active measurements to construct the model. Additionally, it acts as an evaluation tool to gauge the applicability of a model in a given scenario. Identifying the browsing streams associated with a specific internet search activity remains a partially unresolved challenge. While it's feasible to isolate a user's information using their IP address, there's no guarantee that all data generated by the user pertains to the activity being analyzed. For instance, a user might simultaneously stream music from Spotify, browse the web, and have applications running updates in the background. Isolating only the browsing streams presents a challenging task, which is addressed using a greedy approach as detailed in the work discussed in Chapter 8.

Chapter 3

Cloud Gaming Measurements

3.1 Motivation

In this chapter, we will introduce and explain our work in the field of Cloud Gaming taken from our journal: *A Network Analysis on Cloud Gaming: Stadia, GeForce Now and PSNow*, published in MDPI [14].

First of all, let's begin by clarifying what this technology is: cloud gaming [15] [16] is a class of services that promises to revolutionize the video game market. It allows the user to play a video game with minimal equipment while utilizing a remote server for the actual execution. The multimedia content is streamed through the network from the server to the user.

From the description provided, it can be imagined that this type of service requires very stringent specifications, such as low latency and large bandwidth, to function properly. Our contribution has been to investigate the emergence of these technologies, particularly those provided by three different operators: Google[1], Sony, and NVIDIA.

In our work, we study cloud gaming services from the network point of view. We collect more than 200 packet traces under different application settings and network conditions, ranging from a broadband network to poor mobile network conditions, for three cloud gaming services, namely *Stadia* from Google, *GeForce* from NVIDIA, and *PS Now* from Sony. We analyze the employed protocols and the workload they

[1]The project Google Stadia has been terminated in 18th January 2023

impose on the network. We find that *GeForce* and *Stadia* use the RTP protocol [17] to stream the multimedia content, with the latter relying on the standard WebRTC APIs [18]. Depending on the network and video quality, they result in bandwidth-hungry services consuming up to 45 Mbit/s. *PS Now*, instead, uses only undocumented protocols and never exceeds 13 Mbit/s. 4G mobile networks can often sustain these loads, while traditional 3G connections struggle [19]. The systems quickly react to deteriorated network conditions, and packet losses up to 5% do not cause a reduction in resolution.

The remainder of the chapter is organized as follows. Section 3.2 present the related work. Section 3.3 describes our experimental setup, while Section 3.4 illustrates the findings we obtain analyzing the packet traces. Finally, Section 3.5 concludes the chapter, summarizing the lesson learned and the takeaways.

To let other researchers replicate and extend our results, we release sample packet traces available at [20]

3.2 Related Work

In parallel to our work, several studies have delved deeper into Cloud Gaming technology. In particular, [21] and [22] conducted detailed parallel analyses of Cloud Gaming service features, examining the structures of these services in various network scenarios. These studies provide valuable insights into the underlying architecture and protocols, largely corroborating the results obtained in our work while adding further details. From their work, it is confirmed that different games exhibit distinct traffic characteristics such as packet size, inter-packet times, and load. The adoption of the modern VP9 codec [23] is not significantly more efficient than its predecessor, H264 , in terms of traffic load. Stadia consistently strives to maintain output at 1080p and 60fps even when network conditions do not allow it, resorting to 720p only as a last resort.

In addition to these studies, other research efforts have focused on various aspects of cloud gaming technology. For example, authors in [24] delved into the study of cloud gaming platforms and optimization techniques. At the same time, several works provided general frameworks and guidelines for deploying cloud gaming services from the technical [16, 25] and business [26] points of view.

Metzger et al. [27] introduce the technical aspects of video games, focusing on Quality of Service (QoS) and Quality of Experience (QoE) [28]. They define QoE as the perceived quality of a service, incorporating subjective opinions. They distinguish QoE from objective QoS metrics and User Experience (UX). The authors discuss video game QoS metrics, which measure player interactions like task completion times and high scores. These metrics, also known as application layer QoS factors, pertain directly to the game itself. However, the relationship between application layer QoS and QoE is not fully understood, lacking models to correlate specific metrics with QoE. They present a taxonomy of factors influencing a video game's QoS and QoE. This taxonomy encompasses aspects from players, games, game clients, game servers, and networks. Due to cloud gaming, the game client is divided into local and remote factors. These factors are categorized into subjective/context, technical/system, and networking aspects.

Meng et al. [29] explored optimization strategies specifically in the context of Wi-Fi networks, with the goal of improving the delivery of Cloud Gaming services on these wireless networks. Their work illustrates how even a low probability of experiencing very high RTT in the network can lead to instability issues in cloud gaming [30]. In particular, the authors conducted an in-depth analysis of a widely used online live streaming platform with millions of daily active users. Their findings, revealed intriguing insights into network performance. Specifically, they noted that while the median Round-Trip Time (RTT) for wireless users, including both WiFi and cellular connections, was impressively below 100 ms (on par with Ethernet users), the 99 th percentile tail latency spiked to approximately 400 ms. Also other works inspect the importance of an high RTT in cloud gaming scenario [31] [32] [33], this significant latency spike at the 99th percentile implies that users of Real-Time Communication (RTC) applications may experience a delayed video frame approximately once every 100 frames, equating to once every 5 seconds for a 20 frames-per-second stream. Such delays can have a substantial impact on the overall user experience. Moreover, the authors' detailed measurements exposed a concerning trend: wireless users experienced a notable increase in video rebuffering—almost twice as much as their Ethernet-connected counterparts. This highlights a critical area where improvements are needed to enhance the overall quality of user experience on the platform. The researchers have developed Zhuge, an in-AP solution designed to reduce tail latency in real-time communication (RTC) applications on wireless networks.

Marchal et al. [34] leveraged the emulation of network traffic conditions to understand the behavior of Cloud Gaming services. They generated synthetic conditions of jitter, packet loss, and bandwidth to study the impact on video codec and audio adaptation. Subsequently, they delved into the behavior of applications in a real network scenario, focusing in particular on the Orange network in 2022.

Graff et al. [35] focused on the efficient identification of Cloud Gaming traffic at the edge network. They proposed models based on decision trees that achieved an accuracy of 98.5% in detecting Cloud Gaming traffic, demonstrating their applicability in realistic scenarios.

Ky et al. [36] conducted a comprehensive evaluation of eight unsupervised Machine Learning models applied to anomaly detection in Cloud Gaming sessions. They examined the robustness of these models to data contamination and evaluated their performance in various scenarios.

3.3 Measurement Collection

In this section, we describe our experimental testbed and the dataset we collect. We focus on three cloud gaming services, namely Stadia, GeForce Now, and PS Now, on which we created standard accounts that we use for our experiments. We deploy a testbed using three gaming devices: A PC running Windows with a 4K screen, an Android smartphone, and the dedicated Stadia dongle.

All devices are connected to the Internet through a Linux gateway equipped with two 1 Gbit/s Ethernet network interfaces and a 300 Mbit/s WiFi 802.11ac wireless card. The Windows PC is connected to the gateway on the first Ethernet card, while the Android smartphone and the Stadia dongle are connected via WiFi. The gateway uses the second Ethernet interface as an upstream link to the Internet, provided by a 1 Gbit/s Fiber-To-The-Home subscription located in Turin, Italy. Figure 3.1 sketches our testbed. Stadia runs via the Chrome browser on the Windows PC and its mobile application on the Android phone (we used version 2.13). Moreover, we perform additional experiments using the dedicated Chromecast Ultra dongle, which allows the user to play and connect it to a screen. GeForce Now runs from a specific application in both cases (version 1.0.9 for PC and 5.27.28247374 for Android), while PS Now only works from the PC application (version 11.0.2 was used).

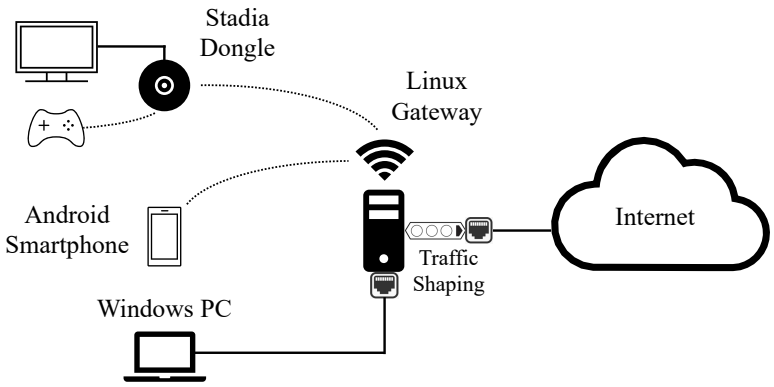


Fig. 3.1 Testbed used for the experimental campaigns.

We play the three services making gaming sessions approximately 5-10 minutes long and capturing all the network traffic the devices exchange with the Internet. We seek reliable results by playing a broad spectrum of videogames on all platforms, from first-person shooters to racing and adventure – e.g., Sniper Elite 4, Destiny, Grid, and Tomb Raider.

With this testbed, we perform five different experimental campaigns, summarized in Table 3.1. Firstly, we run different gaming sessions for each platform using the Windows PC, the smartphone and the dongle (when possible). Secondly, we run different gaming sessions by using the Windows PC and by manually configuring the applications to stream video with different *quality levels*. This option is available on Stadia and GeForce Now. Thirdly, only for GeForce Now, we instrument the Windows application to use one of the 14 available data centres by tuning the *server location* application setting. Next, we artificially impair the network in terms of *bandwidth* and *latency* and *packet loss* to study the behavior of the applications under different network conditions. To this end, we run the `tc-netem` tool on the Linux gateway to progressively decrease the available bandwidth from 100 to 5 Mbit/s, impose additional latency from 10 to 300 ms, or 1-10% packet loss. For Stadia and GeForce Now, we replicate all the experiments using both the PC and the smartphone. Moreover, we perform all experiments also with the Stadia dongle. Finally, we take Stadia as a case study to understand the behaviour with different mobiles networks. To this end, we perform different gaming sessions with the PC and emulated on the Linux gateway different mobile networks using ERRANT [37]. ERRANT is a state-of-the-art network emulator which imposes realistic network conditions based on a large-scale measurement campaign under operational mobile networks. ERRANT

Table 3.1 Overview of the measurement campaign.

Application	PC	Smart-phone	Don-gle	Quality levels	Server location	Traffic shaping	Mobile Networks	Total tests
Stadia	✓	✓	✓	3		✓	✓	94
Geforce Now	✓	✓		2	✓	✓		71
PS Now	✓					✓		60

can reproduce the variability of conditions intrinsically rooted in mobile networks due to different operators, Radio Access Technologies (RATs) (i.e., 3G or 4G), signal quality (e.g., bad quality due to weak signal). ERRANT comes with 32 network profiles describing the typical network conditions observed in different European operators under 3G and 4G. We also use the ERRANT speedtest training dataset to study the possibility of using Stadia on different conditions under mobile networks.

In total, we collect 225 packet traces, summing to 390 GB of data. We share with the research community a sample of these traces from the three services at [20]. Then, we analyze the traffic traces using the Wireshark packet analyzer.[2] We also use Tstat [38], a passive meter, to obtain flow-level logs summarizing the observed TCP/UDP flows. Finally, we use the Chrome debugging console to study Stadia and the disk log files for GeForce Now.

3.4 Results

We now illustrate the findings we obtain from the analysis of the collected network traces. We first show which network protocols each service employs for streaming and signalling (e.g., user's commands) and analyze in detail the different approaches used for audio and video transmission. We then provide quantitative figures on the volume of traffic the services generate at different video quality levels and study the impact of mobile network scenarios. Finally, we study the contacted servers in terms of Autonomous Systems (ASs), RTT distance and discuss how the infrastructures are organized.

[2] https://www.wireshark.org/.

3.4.1 Employed Protocols

In this section, we describe the protocols used by the three cloud gaming providers to stream multimedia content and transmit control data for, e.g., session setup and users' commands. Table 3.2 provides an overview of the protocols we observe, as well as the employed codecs.

Stadia: The service from Google uses the most standard protocol mix as it relies on WebRTC [8]. In few words, WebRTC is a set of standard application programming interfaces (APIs) that allow real-time communication from browsers and mobile applications. It establishes sessions using the Datagram Transport Layer Security (DTLS) protocol for key exchange. The multimedia connection between client and server is set up using Interactive Connectivity Establishment (ICE), which in turn relies on the Session Traversal Utilities for Network Address Translators (STUN) and the Traversal Using Relays around NAT (TURN) protocols for NAT (Network Address Translator) traversal. We find that Stadia uses WebRTC with no substantial modifications, both from the browser and mobile application. The traffic captures using the dedicated dongle device (Chromecast) confirm that the observed traffic is consistently compatible with WebRTC. When the multimedia session begins, the client starts a normal WebRTC *peer connection* towards the server, creating a UDP flow in which DTLS, STUN and RTP are multiplexed according to the RFC 7893 [7]. RTP is used for multimedia streaming, while DTLS carries the user's input. We also observe packets of the Real-Time Control Protocol (RTCP) [39, 40], used to exchange out-of-band statistics between the sender and the receiver of a multimedia stream. The RTCP payload is encrypted to enhance users' privacy, preventing the in-network devices from using it for Quality of Service monitoring.

GeForce Now: It adopts a different approach. The server is first contacted using the TLS (over TCP) protocol to set up the session. Interestingly, the Client Hello messages contain the Server Name Indication extension, which allows us to infer the server hostname (see Section 3.4.6 for details). Then, the client opens multiple UDP channels directly, without relying on the standard session establishment protocols (ICE, STUN and TURN). Only the first packet from the client contains an undocumented hello message. Each inbound flow then carries a standard RTP stream. The client sends the user commands on a dedicated UDP flow using an undocumented protocol. All flows use fixed ports on the client-side, in the range 49003-49006,

Table 3.2 Protocol usage for different gaming session components.

	Stadia	GeForce Now	PS Now
Streaming	RTP (and RTCP)	RTP	Custom (UDP)
Player's input	DTLS	Custom (UDP)	Custom (UDP)
Session setup	DTLS, STUN	TLS	Custom (UDP)
Network Testing	RTP	Iperf-like	Custom (UDP)
Video Codec	H.264, VP9	H.264	-

while they vary on the server-side. Here, we do not observe the presence of the RTCP protocol.

PS Now: This service adopts a completely custom approach, with no standard in-clear header. The client opens a UDP connection towards the server without relying on any documented protocol, and, as such, we can only analyze the raw payload of packets. Still, complex manual work allowed us to catch at least the high-level encapsulation schema that we briefly describe here. The first byte of the packet is used to multiplex multiple data channels. The channel 0 is used for signalling and user's commands, while 2 and 3 are used for multimedia streaming from the server. This is confirmed by the plausible packet size and inter-arrival distributions and allows us to infer which kind of multimedia content is carried on each channel, as we illustrate in Section 3.4.3.

3.4.2 Network Testing

All three services have built-in functionalities to probe the network between the client and (multiple) gaming server machines to determine if the conditions are sufficient for a stable gaming session. In few words, the client applications perform a speed test-like measurement to estimate the network delay and bandwidth. We notice that the network testing is not performed consistently on each session startup, but the applications tend to re-probe the network only after a variation of the client IP address.

Stadia performs a speed test based on RTP packets carried over a session established using the standard WebRTC APIs for the multimedia streams. The server (not necessarily the same used for the subsequent gaming session) sends 5-6 MB of data to the client, resulting in a UDP session 5-60 seconds long, depending on the

network conditions. The RTP packets are large-sized, around 1200 bytes on average, but we cannot inspect their payload since it is encrypted.

GeForce Now uses a schema similar to the one used in the popular tool Iperf[3], in which the client sets up a network test over a UDP channel on the server port 5001 (the same port used by Iperf). The first few packets carry JSON-encoded structures to set up the test. In case the test includes a latency measure, the last flow packets indicate the measured RTT samples. In case the test is only for bandwidth, we observe a stream of large-sized UDP packets lasting 5-10 seconds. Again, the testing server is different from the one used for the subsequent gaming session. The inspection of the JSON messages allows us to understand that the client probes the latency towards multiple alternative measurement servers.

PS Now adopts a fully-custom approach again. At the beginning of each gaming session, the client performs a few-seconds long bandwidth test using a custom or fully encrypted protocol running over UDP. We cannot infer any information from the packets, for which we only observe that they all have size 1468 bytes. The test is performed towards a server different from the one used for the proper gaming streaming session. We note similar additional streams consisting of few packets toward a handful of other servers that we conjecture are used to probe the latency towards more endpoints.

Privacy concerns: While analyzing the GeForce Now network testing mechanism, we notice that the client-side control packets used to set up the test expose the user to a severe privacy concern [41]. The user ID is sent in clear into the UDP packet, allowing an eavesdropper to uniquely identify a user even if she changes her IP address or is roaming on another network. We compared the user ID to the user account number that we obtained on the NVIDIA website profile management page, and they match, confirming that the identifier is uniquely associated with the account. Following the best practices for these cases, we signalled the issue to NVIDIA before making our work public, which plans to resolve it on one of the following updates.

[3] https://iperf.fr/, accessed October 2021.

3.4.3 Multimedia Streaming

We now analyze how the three services stream the multimedia content (audio and video) from the gaming server to the client. In the case of Stadia and GeForce Now, we will provide figures extrapolated inspecting the RTP headers, while for PS Now we can separate the different streams by looking at the first byte of the UDP payload as mentioned in the previous section. In the last row of Table 3.2 we report the employed video codecs as we extract from the browser/application log files. The widespread H.264 codec is used by both GeForce Now and Stadia, employing the newer VP9 if the device supports it. For PS Now, we could not obtain any information about the codecs.

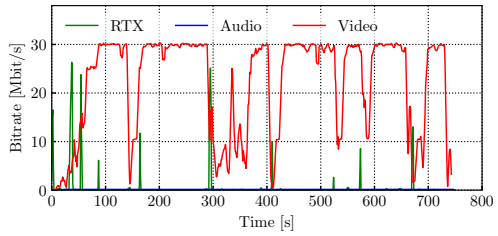
Stadia relies on the WebRTC APIs, and, as such, the multimedia streaming follows its general principles. A single UDP flow carries multiple RTP streams identified by different Source Stream Identifiers (SSRC). A stream is dedicated to the video, while another one to the audio track. We also find a third flow used for video retransmission, as we confirm using the Chrome debug console.[4] During most of the session, it is almost inactive. At certain moments the flow becomes suddenly active, carrying large packets containing video content. Moreover, this behaviour co-occurs with packet losses and bitrate adaptations on the video stream, as we expect for a video retransmission feature.

GeForce Now again relies on RTP for multimedia streaming, as described in the previous section. Differently from Stadia, it uses separate UDP flows for the different multimedia tracks, whose client-side port numbers can be used to distinguish the content as NVIDIA publicly declares.[5] On port 49005, a UDP flow carries a single RTP stream for the inbound video. The audio is contained in a UDP flow on port 49003, in which we find two RTP streams active at the same time.

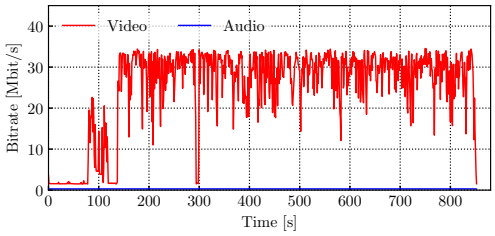
Regarding PS Now, we cannot find any header belonging to publicly documented protocols. However, the inspection of several packet captures allows us to infer the encapsulation schema used by the application. A single UDP flow carries all multimedia streams. To multiplex the streams, the first byte of the UDP payload indicates the channel number, followed by a 16-bit long sequence number. Channel

[4] The associated RTP stream is found to have mimeType `video/rtx`.

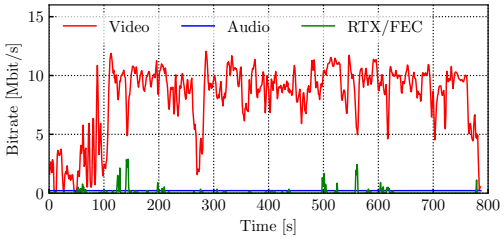
[5] https://nvidia.custhelp.com/app/answers/detail/a_id/4504/~/how-can-i-reduce-lag-or-improve-streaming-quality-when-using-geforce-now, accessed October 2021.



(a) Stadia (1080p).



(b) GeForce Now (1080p).



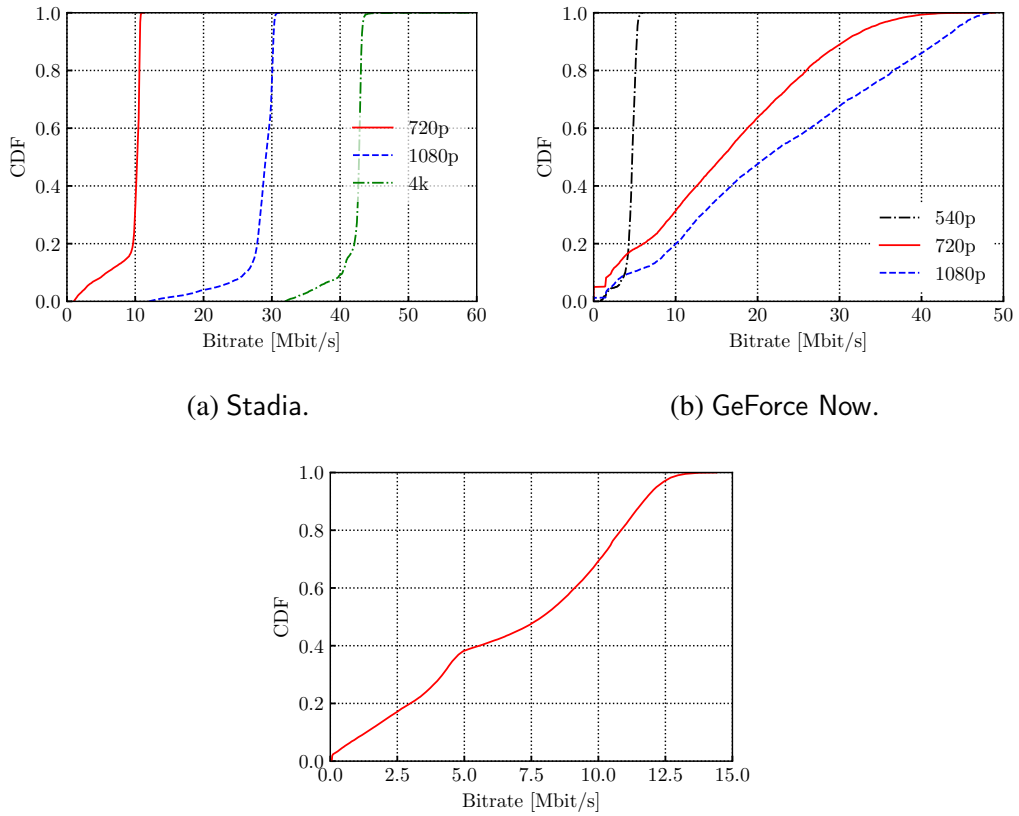
(c) PS Now.

Fig. 3.2 Examples of temporal evolution of bitrate for three gaming sessions.

2 carries the video stream, as we can conclude by looking at packets' packet size and inter-arrival time. Channel 3 carries the audio track as the packets are small and fixed-sized (250 B) and arrive at a constant pace of one every 20 ms. We also find channel 0, especially at the beginning of the flow, which we conjecture is used for signalling. Finally, channel 12 seldom becomes active, especially in correspondence of large packet losses. As such, we conjecture that it is used for video retransmission or some form of forwarding error correction (FEC), similarly to the Stadia approach.

3.4.4 Network Workload

We now focus on the workload imposed on the network by users playing on cloud gaming services. We start our analysis with Figure 3.2, in which we show the evolution of a gaming session of around 10 minutes for each service. We made the corresponding packet traces available to the community at [20]. The picture reports the bitrate of the inbound traffic, due almost exclusively to the video multimedia stream. We first notice that Stadia (Figure 3.2a) has a constant bitrate, while for GeForce Now and PS Now (Figures 3.2b and Figure 3.2c respectively) it is consider-



(a) Stadia.

(b) GeForce Now.

(c) PS Now

Fig. 3.3 Cumulative distribution of the video bitrate, for different quality video levels.

ably more variable. Especially for GeForce Now, we can root this in the different video codecs, as we describe later in this section. Indeed, the application logs confirm that the variations in the bitrate are not caused by resolution adjustments or codec substitution. Looking at Stadia, the role of the video retransmission stream (green dashed line) is clear, which becomes active in correspondence of impairments in the main video stream (solid red line). We notice a very similar behaviour in PS Now, which allows us to conjecture the presence of an analogous mechanism.

We summarize the network workload showing in Figure 3.3 the empirical cumulative distribution function of the video bitrate that we observe. For Stadia and GeForce Now, we report separate distributions for different video resolutions thanks to Chrome debug console and application logs, respectively. For PS Now, we only report the overall distribution as we cannot extract any statistics from the client app.

As mentioned before, Stadia exhibits a rather constant bitrate (Figure 3.3a). The service allows three streaming resolutions, namely 720p, 1080p and 4K (2160p), whose average bitrate is 11, 29 and 44 Mbit/s respectively. This is consistent with what is declared in documentation.[6] Stadia employs both H.264 and VP9 video codecs, with 4K streaming using uniquely VP9.

Different is the picture for GeForce Now, shown in Figure 3.3b. The service allows several video resolutions, both in 16:9 and 16:10 aspect ratios. Here, we report the bitrate of the lowest (720p) and the highest (1080p) resolutions available for the 16:9 aspect ratio. Moreover, we show the 540p resolution, which is only adopted automatically in case of bad network conditions, that we trigger imposing a bandwidth limit on the testing machine. The figure shows that the bitrate has large variability, especially for 720p and 1080p. On median, 720p (1080p) consumes 15 (20) Mbit/s, which is consistent with what is declared on the system requirements of the service.[7] However, the interquartile ranges (IQRs) are in the order of 15 Mbit/s, much more than the 2-3 Mbit/s IQRs observed in Stadia. The bitrate reaches peaks of more than 30 and 40 Mbit/s for 720p and 1080p, respectively. Without access to the unencrypted raw video stream, we conjecture that GeForce Now makes a highly-dynamic use of the H.264 compression parameters to adapt to different video characteristics (static/dynamic scenes) and network conditions. Indeed, our experiments with limited bandwidth show that, for example, GeForce Now can sustain a 1080p video stream also with less than 15 Mbit/s available bandwidth without dropping the frame rate, likely adjusting the H.264 compression parameters.

Finally, Figure 3.3c shows the bitrate distribution for PS Now. Given the lack of application settings or debug information, we only show the overall distribution of the bitrate. The online documentation recommends a minimum bandwidth of 5 Mbit/s and states that video streaming has a 720p resolution. However, we observe the bitrate reaching up to 13 Mbit/s, with a consistent variability. Interestingly, when we impose a 10 Mbit/s or lower bandwidth limitation on the network, the bitrate adapts consequently (see the peak in the distribution at 5 Mbit/s). However, we cannot link it with a resolution lower than 720p.

We now study the impact of packet losses on cloud gaming, focusing on Stadia as a case study. As described in Section 3.3, we run experiments in which we enforce

[6] https://support.google.com/stadia/answer/9607891, accessed October 2021.
[7] https://www.nvidia.com/it-it/geforce-now/system-reqs/, accessed October 2021.

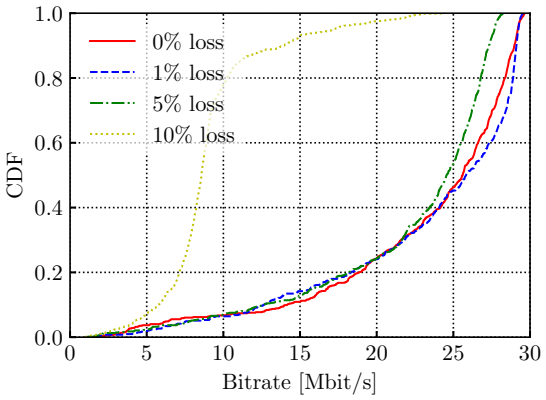


Fig. 3.4 Bitrate distribution with different artificial packet loss.

artificial packet loss via the `tc-netem` tool, ranging from 1% to 10%. We then study how the application reacts to these scenarios, focusing on the sending video bitrate and quality level. In Figure 3.4, we show the distribution of bitrate for experiments with 1, 5 and 10% packet loss and report the bitrate of sessions with no packet loss as a reference (solid red line). We notice that a packet loss of 1 and 5% (blue and green dashed lines, respectively) does not cause significant bitrate variations. Indeed, the bitrate reaches 30 Mbit/s, and we notice the game is running at 1080p most of the time, with some 720p periods. We notice a constant activity on the *retransmission* stream, used to recover lost packets. Different is the case with 10% packet loss (yellow dashed line). In this case, Stadia keeps the video resolution at 720p most of the time, and, as such, the bitrate is limited to 10 Mbit/s. We notice a high activity again on the *retransmission* stream, which, however, is not deemed sufficient to go for higher video resolutions. In a few cases, 1080p is achieved, but for short 5-10 second periods. In all the cases, the games were still playable and enjoyable, as reported by the volunteers.

3.4.5 Cloud Gaming under Mobile Networks

We now investigate the case of mobile networks to understand to what extent cloud gaming is feasible and what is the reached quality level. Moreover, we are interested in understanding the impact of variable network conditions and how applications react to network impairments. Here, we focus on Stadia, which stream video content at fixed bitrates, requiring strict bandwidth constraints.

We first build on a large-scale measurement campaign run on our previous work [37], to study to what extent current mobile networks are ready to sustain the load and offer suitable conditions for cloud gaming. The dataset we use includes more than 100k speed test measurements from 20 nodes/locations equipped with SIM cards of 4 operators, located in 2 distinct countries (Norway and Sweden). Each experiment measured latency, downlink, and uplink capacity using multiple TCP connections towards a testing server located in the same country. Moreover, the dataset indicates physical-layer properties such as Radio Access Technology (3G or 4G) and signal strength. The measurement campaign spans the entire 2018 and includes experiments on different hours of the day and days of the week. We use the dataset to understand how a Stadia cloud gaming session would have performed with the measured network characteristics. This is possible thanks to the steady network usage of Stadia, where different video resolutions utilize almost fixed bandwidth. As such, given the network conditions measured on the speedtest measurements, we consider running Stadia *feasible* if there is at least 10 Mbit/s available bandwidth. In case available bandwidth is in $[10-30)$ Mbit/s, we conclude that a user could only reach a video resolution of 720p. When bandwidth is in the range $[30-44)$ Mbit/s, a better 1080p video could be transmitted, while more than 44 Mbit/s allow the user to receive a $4K$ quality video properly.

Figure 3.5 shows the reached video quality levels, offering a breakdown on 3G and 4G and different signal qualities as measured by the node's radio equipment.[8] The figure shows how a poor 3G link does not allow using Stadia in most cases, as the available bandwidth is below 10 Mbit/s. The picture changes with medium signal quality, where using Stadia is possible most of the time, with the lowest 720p resolution. An excellent 3G connection allows higher resolution in less than 10% of cases. With 4G, the network generally offers higher bandwidth, and 1080p and 4K are possible, especially when good signal quality. Indeed, 4G links with high signal strength can sustain Stadia 4K streaming 40% of the cases, and only 38% of the times the client is limited to 720p.

Next, we study the capability of the gaming platforms to offer a good gaming session and cope with different mobile network conditions. To this end, we use the PC to run a gaming session and track different quality metrics about the frame rate, expressed in Frames Per Second (*FPS*), the packet loss, and the video resolution

[8]The measured Received signal strength indication (RSSI) is mapped to quality levels as recommended at https://wiki.teltonika.lt/view/Mobile_Signal_Strength_Recommendations.

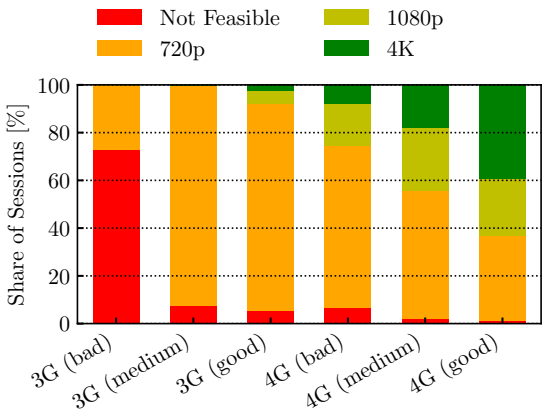


Fig. 3.5 Estimated resolution of Stadia sessions on mobile networks.

quality experienced during the gaming sessions. We focus on Stadia as a case study since it is available on many devices.

To study the gaming session in a controlled mobile network environment, we emulate different mobile network conditions on the Linux gateway by using ERRANT. In detail, we perform experiments by using a 4G good network profile as, currently, it is the best quality and most widespread network, and with a 3G good network profile as it is the lowest quality profile showing enough bandwidth to support a gaming session. To reproduce the mobile network variability, we set ERRANT to resample the network condition every 10 seconds, i.e., we pick and apply new constraints for the download rate, upload rate, and latency from the selected profile.

Figure 3.6 reports an example of the frame rate (right y-axis), the packet loss (left y-axis), and the video resolution quality when a sudden change happened during a gaming session while using the 4G good profile. Interestingly, we experienced stable performance for most gaming sessions with no packet loss, 60 FPS, and 1080p resolution. Only when ERRANT picks a download bandwidth below 10 Mbit/s, we experience, for a short time, a reduction in frame rate and resolution down to 20 FPS and 720p, respectively, with a seldom increase of the packet loss. Interestingly, the Stadia platform can quickly react and adapt itself by reaching a frame rate of 60 FPS and resolution at 720p, rising again to 1080p as soon as a better bandwidth is available. With a 3G good network profile, instead, we could not run an entire gaming session as the network variability introduced by the mobile network caused the game to stop suddenly. This shows how the promising benefits of these solutions

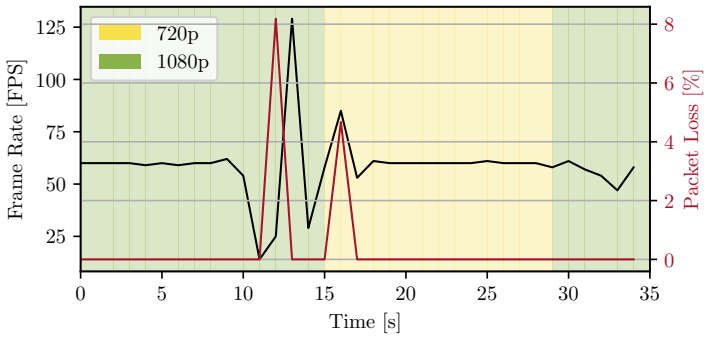


Fig. 3.6 Example of quality-level reduction in Stadia in terms of frame rate and video resolution subsequent to packet losses.

Table 3.3 Gaming servers characterization.

	Servers	Subnets	ASNs	Owner
Stadia	74	22	15169	Google
GeForce Now	37	23	11414, 20347, 50889	NVIDIA
PS Now	36	2	33353	Sony

currently have limitations that could be overcome with the increasing popularity of fast mobile network technologies – 4G and, in particular, 5G.

3.4.6 Location of Gaming Machines

This section provides a preliminary investigation of the cloud gaming infrastructure regarding the number and location of game servers and employed domains, as observed from our location. This can be useful to identify cloud gaming traffic for, e.g., traffic engineering or content filtering.

We first focus on the remote gaming machines, analyzing the server IP addresses the client applications contact, summarized in Table 3.3. Indeed, after the initial setup phase, the client exchanges traffic (almost) uniquely with a single server where the gaming is likely executed.[9] Considering Stadia, at each session, we contacted a different server – i.e., we performed 74 sessions and reached 74 different server

[9]We cannot infer if the server IP address acts as an ingress load balancer or reverse proxy.

IPs. They lay on 22 subnets /24, all belonging to the Google 15129 AS.[10] Different is the case for GeForce Now and PS Now, for which in roughly 50% of the cases we contacted a server IP we had already observed. GeForce Now servers lay on 23 subnets belonging to three different ASes, all controlled by NVIDIA. Remind that we used all the 14 available NVIDIA data centres in our experimental campaign by instrumenting the client application. Finally, all 36 server IPs for PS Now belong to only 2 subnets /24 from the 33353 Sony AS. In terms of latency, additional `ping` measures show that Stadia and PS Now servers are 6-8 ms away from our location while Central European GeForce Now in the order of 15-20 ms. However, we cannot link this to the service quality or infrastructure deployment since we perform measurements from a single vantage point. We did not qualitatively observe a significant lag between users' commands and game response without traffic shaping.

Finally, we analyze the domains that the applications contact during the gaming sessions. We extract them by inspecting the client's Domain Name System (DNS) queries before opening a new connection and extracting the Server Name Indication (SNI) field from the Transport Layer Security Security (TLS) Client Hello messages. In the case of Stadia, we only find `stadia.google.com` as domain-specific to this service. Indeed, the client application contacts a dozen of other domains, but those are shared with all Google services (e.g., `gstatic.com` and `googleapis.com`), and, as such, not specific of Stadia. Regarding GeForce Now, the application contacts the general `nvidia.com` domain as well as the more specific `nvidiagrid.net`. Finally, PS Now relies on the `playstation .com` and `.net` domains and their sub-domains, which are, thus, not specific to the PS Now service. Interestingly, GeForce Now is the only service that uses domains to identify gaming machines, using subdomains of `cloudmatchbeta.nvidiagrid.net`. Indeed, for the other services, the domains we find are associated uniquely to the control servers – used for login, static resources, etc. – while gaming machines are contacted without a prior DNS resolution.

3.5 Takeaways

This chapter has presented a detailed analysis of network protocols and traffic characteristics for three predominant cloud gaming services: Google Stadia, NVIDIA

[10]We map an IP address to the corresponding AS using an updated RIB from http://www.routeviews.org/.

GeForce Now, and PlayStation Now (PS Now). The key findings from this analysis are summarized as follows:

- **Protocol Usage:** Google Stadia extensively utilizes WebRTC, incorporating standard APIs for real-time communication, including DTLS, ICE, STUN, and TURN for session setup and multimedia streaming. NVIDIA GeForce Now, in contrast, employs TLS for session setup and RTP for streaming, diverging from standard session establishment protocols. PlayStation Now adopts a distinctive approach with a fully-custom protocol that lacks standard in-clear headers.
- **Network Testing:** Each service has developed its own methodology for network testing. Stadia uses an RTP-based speed test, GeForce Now employs an Iperf-like mechanism, and PS Now utilizes a custom UDP protocol. Notably, GeForce Now's implementation reveals a privacy concern where user IDs are transmitted unencrypted.
- **Multimedia Streaming and Codecs:** For streaming, Stadia and GeForce Now utilize RTP, with Stadia additionally employing RTCP for out-of-band statistics. PS Now follows a unique route with a custom streaming protocol. Regarding codecs, Stadia alternates between H.264 and VP9, GeForce Now uses H.264, while the codecs for PS Now remain undisclosed.
- **Network Workload and Bitrate Variability:** Stadia maintains a relatively constant bitrate, contrasting with the significant variability observed in GeForce Now and PS Now. Stadia's ability to adjust resolution in response to packet loss ensures sustained playability under various network conditions.
- **Cloud Gaming on Mobile Networks:** The feasibility and quality of cloud gaming on mobile networks are highly dependent on network conditions. For instance, Stadia necessitates a minimum of 10 Mbit/s for 720p quality, scaling upwards for higher resolutions. The performance is markedly improved on 4G networks compared to 3G.
- **Infrastructure and Server Locations:** The gaming servers for each service are primarily located within their respective corporate AS. A unique aspect of GeForce Now is its identification of gaming machines via DNS domain names, a feature not observed in Stadia and PS Now.

In summary, this comprehensive analysis uncovers varied approaches in protocol usage, network testing, and multimedia streaming across the examined cloud gaming services. While Stadia and GeForce Now lean towards more standardized protocols, PS Now embarks on a path of complete customization. The observed variability in bitrate and the adaptive nature of these services to network conditions are pivotal, particularly in mobile network environments. Additionally, the study brings to light potential privacy issues and the diverse strategies employed in infrastructure deployment across these services.

Chapter 4

HTTP/3 - QUIC Measurements

4.1 Motivation

In this chapter, we present the evolution of HTTP protocol towards the modern HTTP/3 based on QUIC, this chapter is mostly based on our paper: *A first look at HTTP/3 adoption and performance*, published in Computer Communications [42].

The Hypertext Transfer Protocol (HTTP) is used to access the vast majority of services on the Internet, from websites to social networks and collaborative platforms. HTTP was born in the early 90s, and its first version (HTTP 1.1) was standardized in 1997 [10]. It was not until 2014 that the second version (HTTP/2 [11]) was standardized, including significant changes to the protocol's framing mechanisms [12]. HTTP/3 is the third version of HTTP and is currently in the final standardization phase at the IETF [43]. HTTP/3 promises performance benefits and security improvements over HTTP/2. One major change is that HTTP/3 replaces TCP as the transport layer in favor of QUIC, a UDP-based transport protocol originally proposed by Google and currently an IETF standard [44]. In addition, HTTP/3 introduces a more effective header compression mechanism and uses TLS 1.3 [13] (or higher) to improve security.

HTTP/3 is expected to take the place of HTTP/2 in the next few years, and some of the leading Internet companies have already announced plans to support it starting

in 2020, such as CloudFlare CDN[1] and Facebook.[2]However, very few works [45, 46] have examined HTTP/3 deployments. More importantly, the impact of the protocol on Web performance has not been widely measured yet. Such efforts are important to externally validate the benefits of the protocol, which have only been evaluated by the few service providers that have deployed it.

In our work, we fill this gap by conducting a large-scale measurement study of HTTP/3 adoption and performance. We first rely on the HTTPArchive dataset[3] to examine the extent to which the Web ecosystem has adopted HTTP/3. Then, we run additional campaigns to measure the benefits introduced by HTTP/3. Considering websites using different versions of the HTTP protocol, we measure various metrics known to indicate user Quality of Experience (QoE). Finally, we emulate different network conditions on the network path to assess whether, and to what extent, HTTP/3 improves performance in different scenarios. Please note that the measurements are conducted without considering whether the congestion control algorithm employed by a given server is consistent across all three protocols. This is due to the fact that such information is often not ascertainable a priori and because the choice of algorithm remains at the discretion of the individual configuring the server. However, our primary focus is to determine whether, for the domains under analysis, the respective protocol implementations exhibit substantial differences, enabling us to discern if one service outperforms the others. It's important to emphasize that, at this point, one of the contributing factors to potential performance improvements could be attributed to the choice of congestion control at the server side. Additionally, it's crucial to note that QUIC, in particular, is designed with flexibility, allowing for the development and testing of alternative solutions beyond the conventional congestion control algorithms integrated into kernel systems.

Using the open source HTTPArchive dataset, we find thousands of websites that support HTTP/3. Initially, we download the dataset using the gsutil CLI[4]. Subsequently, we process the data, searching for rows containing the "alt-svc" field. Domains identified with "alt-svc quic" or "alt svc h3-*" values are considered HTTP/3 websites. Most of them are hosted by a handful of Internet hypergiants, i.e., Facebook, Google, and Cloudflare. We then automatically visit websites that

[1] https://blog.cloudflare.com/http3-the-past-present-and-future/
[2] https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/
[3] https://httparchive.org/
[4] https://cloud.google.com/storage/docs/gsutil?hl=it

support HTTP/3 using the different HTTP versions and under different network conditions to measure performance in terms of QoE-related metrics. We visit a total of 14 707 websites while emulating artificial latency, packet loss and limited bandwidth. We perform 2 647 260 visits over a one-month period to determine the benefits of HTTP/3 for normal web browsing activities. We then supplement the analysis with additional ad-hoc campaigns to measure specific aspects, such as to examine mobile browsing scenarios and video streaming usage.

We find that the benefits of HTTP/3 only emerge under certain network conditions and vary significantly across websites. Our main results are:

- Google, Facebook, and Cloudflare are the early adopters of HTTP/3 and host almost the totality of currently websites supporting HTTP/3.
- The majority of web page objects in websites that support HTTP/3 are still hosted on third-party servers that do not support HTTP/3.
- In current deployments HTTP/3 brings significant performance benefits in high latency scenarios and limited benefits in very low bandwidth ones.
- As expected, sites that require fewer connections to load objects benefit the most.
- The benefits of HTTP/3 are significant in mobile scenarios, such as for users browsing from smartphones and tablets.
- Performance gains largely depend on the infrastructure hosting the website, possibly due to optimizations on the server side.
- Adaptive video streaming services do not seem to benefit from HTTP/3 in terms of key QoE-related metrics.

Finally, to ease reproducibility of our results and enable additional measurement campaigns, the scripts used to set up and run our experiments are now available on GitHub.[5]

The chapter is organized as follows: Section 4.2 illustrates related work. Section 4.3 presents our datasets and data collection methodology. Section 4.4.1 illustrates our results on HTTP/3 adoption, while Section 4.5 and Section 4.6 evaluate

[5] https://github.com/SmartData-Polito/h3-benchmark

the performance of HTTP/3 in web browsing and video streaming, respectively. Section 4.7 concludes the chapter.

4.2 Related Work

Given its recent conception, few works have addressed HTTP/3. Saif *et al.* [45] conduct experiments where they control both the client and the server when accessing a single web page. They study the impact of delay, packet loss, and throughput on HTTP/3 performance without finding major effects. In contrast, we conduct a large-scale measurement campaign that controls only the client and examines thousands of HTTP/3 websites in production. This setup allows us to consider both real-world network conditions and server implementations.

Marx *et al.* [46] compare 15 HTTP/3 implementations and find great heterogeneity in how congestion control, prioritization, and packetization work. They perform single file downloads without providing large scale measurements in the wild, which we provide here. Cloudflare benchmarks its own HTTP/3 implementation in draft 27 in [47] and finds that it is $1-4\%$ slower than HTTP/2. However, their experiments are limited to the website `blog.cloudflare.com`. Guillen *et al.* [48] propose a control algorithm for adaptive streaming tailored for HTTP/3. Saif *et al.* [49] measure performance benefits of using HTTP/3 instead of MQTT and MQTT-over-QUIC in IoT scenarios. Lovell *et al.* [50] compare HTTP/3 support across millions of websites and show that the most popular websites have not yet explored HTTP/3, while less popular websites have higher HTTP/3 adoption.

QUIC has been the subject of numerous studies. Wolsing *et al.* [51] show that QUIC performs better than TCP thanks to its fast connection establishment. Manzoor *et al.* [52] show that QUIC performs worse than TCP in wireless mesh networks due to the poor interaction of the protocol with the WiFi layer in this scenario. Carlucci *et al.* [53] found that QUIC reduces the overall page load time. Kakhi *et al.* [54] conducted a large-scale measurement campaign on QUIC and found that it outperforms TCP in most cases. However, these works target Google's QUIC versions, while the current IETF standard has made significant progress [55]. Moreover, they focus exclusively on the transport layer and neglect the improvements introduced by HTTP/3, which we measure in this work.

Table 4.1 Description of the employed datasets.

Dataset	Runs	Goal
HTTPArchive	53 107 185	HTTP/3 Adoption
BrowserTime-Web	2 647 260	Browsing Performance
BrowserTime-Mobile	1 800	Mobile Browsing Performance
BrowserTime-Video	360	Video Streaming Performance

4.3 Datasets and Performance Metrics

We rely on several datasets to study (i) the adoption of HTTP/3 and its performance on (ii) normal web browsing, (iii) mobile browsing, and (iv) video streaming. We summarize these datasets in Table 4.1.

4.3.1 HTTP/3 Adoption

We examine HTTP/3 adoption using the HTTPArchive, an open dataset available online.[6] The dataset contains metadata derived from visits to a list of more than 5 million URLs provided by the Chrome User Experience Report.[7] The list of URLs is compiled using navigation data from real Chrome users and provides a representative view of the most popular websites and services accessed worldwide.[8] Each month, all URLs visited with the Google Chrome browser are taken from a U.S.-based data center and the resulting navigation data is published. For each visit, the dataset contains information about page characteristics, load performance, and HTTP transactions in HAR format[9] including request and response headers.

Of fundamental importance to our analyzes are the HTTP responses, which contain the eventual `Alt-Svc` header used by servers to announce support for HTTP/3. By setting the `Alt-Svc` header, the server tells the client that subsequent connections can use HTTP/3, while also indicating its support for specific design versions (e.g., 27 or 29).

We download the HTTPArchive dataset from November 2019, when we first observe sites supporting HTTP/3. We monitor the HTTPArchive through September

[6] https://httparchive.org/, visited on February 4, 2021.
[7] https://developers.google.com/web/tools/chrome-user-experience-report
[8] HTTPArchive previously adopted the Alexa Top 1M Websites list, but switched to the Chrome User Experience Report when Alexa discontinued its ranking in July 2018.
[9] http://www.softwareishard.com/blog/har-12-spec/

Table 4.2 Network configurations used in the experiments.

Parameter	Tested settings
Latency [ms]	Native, 50, 100, 200
Loss [%]	Native, 1, 2, 5
Bandwidth [Mbit/s]	Native, 5, 2, 1

2021. We use the data to examine the trend of HTTP/3 adoption. The data is 6.6 TB. Since we are interested in examining HTTP/3 adoption on *websites*, we discard all visits to internal pages (less than half of the total) and keep only visits to home pages. We refer to this dataset as *HTTPArchive*.

4.3.2 HTTP/3 Performance

Our goal is to compare the performance of the three HTTP versions when accessing heterogeneous types of content. To this end, we collect three datasets: (i) *BrowserTime-Web* , including visits to websites supporting HTTP/3 from a regular browser, (iii) *BrowserTime-Mobile* , targeting mobile websites under mobile network conditions, and (iii) *BrowserTime-Video* , targeting video streaming.

Web Browsing

To automate website testing, we rely on BrowserTime, a docked tool for performing automated visits to websites with a large number of configurable parameters.[10] We use BrowserTime to instrument Google Chrome to visit web pages with a specific HTTP version. Importantly for our goal, Google Chrome provides the ability to specify a set of domains to be contacted on the first visit using HTTP/3, i.e., without prior specification via the `Alt-Svc` header. We restrict ourselves to Chrome, since we are not aware of similar features in other browsers (e.g., Firefox).

We are interested in studying the impact of HTTP/3 under different network conditions. For this reason, we perform our measurements under different *network configurations*.[11] We conduct our experiments with two high-end servers connected to the Internet via 1 Gbit/s Ethernet and located on our university campus. We call this baseline scenario *Native*, as indicated in Table 4.2.

[10] https://www.sitespeed.io/documentation/browsertime/

[11] We have included configurations covering the typical network conditions previously observed in real measurements [19]

We then enforce the network configurations during the visits using the Linux tool `tc` tool. For each network configuration, we change one of the three network parameters enforcing: (i) additional latency or (ii) additional packet loss or (iii) bandwidth limit. For each parameter, we use 4 different settings listed in Table 4.2. In the case of latency, we simulate an increasing *Round Trip Time* (RTT) and therefore only apply it in one link, namely the uplink. For loss and bandwidth constraint, we enforce the configuration for both uplink and downlink. For each network configuration, we visit each website (i) with only HTTP/1.1, (ii) with HTTP/1.1 and HTTP/2, and (iii) with all three versions of the protocol. All visits to the same website are performed sequentially, cleaning up all state between repetitions, i.e., browser cache, TCP connections, etc.

We collect the *BrowserTime-Web* dataset by visiting websites that currently support HTTP/3. At the time we run this experimental campaign (December 2020), we find 14 707 websites that announce support for HTTP/3 and test them all.

The visits are repeated 5 times to get more reliable results. So we visit each website $4 \times 3 \times 3 \times 5 = 180$ times. Next, we visit these websites with three HTTP versions (HTTP/1.1, HTTP/2, and HTTP/3) to quantify potential performance improvements. In total, we performed 2 647 260 visits over a period of one month. The metadata of the visits accounts for 189 GB, and we call this dataset *BrowserTime*.

Browsing Under Mobile Networks

We also evaluate the impact of HTTP/3 for mobile users, i.e., users of smartphones or tablets connected via 3G or 4G mobile networks. We conduct an additional measurement campaign in which we emulate both mobile devices and mobile network conditions.

For the former, we rely on BrowserTime's ability to mimic mobile devices by setting the appropriate user agent string in Google Chrome and limiting the size of the view port when rendering the page. We emulate an iPhone 6 and an iPad tablet. For the latter, we use ERRANT [19], a data-driven open-source emulator for mobile access networks. Briefly, ERRANT uses more than 100 thousand speedtest measurements obtained from real mobile networks to simulate network profiles for different Radio Access Technologies (RATs) (3G or 4G) and signal strengths (bad, medium, and good). Each network profile describes both typical behavior and

inherent network variability. Then, ERRANT uses the Linux tool tc-netem Linux tool to enforce the selected network profile that emulates both the typical behavior and the network variability.

In this experimental campaign, we target a random subset of 100 websites that support HTTP/3. We have reduced our sample in this experiment to limit both the time needed to complete the measurements and the generated traffic. To ensure a general coverage of the list of websites used in other experiments, we perform a stratified random sampling across content providers. Specifically, we take 25 websites for the top three content providers (Google, Facebook, and CloudFlare) plus 25 from the remaining websites.

We visit each website using both emulated devices (tablet and smartphone). As an additional comparison step, we revisit the website with the default desktop setting, as in the *BrowserTime* dataset. We test the websites with 6 ERRANT profiles, namely the combination of the two RATs (3G and 4G) and three signal strengths (bad, medium, and good). For each profile, we run 10 experiments, firing a total of 54 000 visits. We refer to this dataset as *BrowserTime-Mobile* .

Video Streaming

In addition to web browsing performance, we evaluate the impact of HTTP/3 on video streaming. Among the dozens of protocols for video streaming, most providers have moved to solutions based on streaming over HTTP. The most widely used solution is called Dynamic Adaptive Streaming over HTTP (DASH), which splits the video into chunks of a few seconds that the client retrieves via HTTP requests.

DASH supports adaptive streaming by allowing the client to choose the best video resolution among those available on the server, depending on network conditions. We run an experimental campaign for video streaming in a controlled test environment.

Popular commercial video streaming services such as Twitch, Prime Video, or Netflix do not yet support HTTP/3 by the time of writing. YouTube, on the other hand, supports HTTP/3 and HTTP/1.1, but surprisingly not HTTP/2. As our goal in this paper is to assess the benefits of HTTP/3 considering *also* HTTP/2, no streaming service currently in production could serve as basis for our analysis. We therefore prefer to use a controlled environment on both the client and server side to measure performance across protocols.

In our setup, an instrumented browser runs a DASH web client that plays a video hosted on our server set to support HTTP/1.1, HTTP/2, and HTTP/3. We use the popular and open source player `Dash.js`[12] and a `nginx` web server set to use Cloudflare's *quiche* HTTP/3 and QUIC implementation.[13] The server hosts a 9-minute video delivered in 150 chunks, available in 10 bitrates ranging from 250 kbit/s to 14 Mbit/s.

We test the same network conditions as in Table 4.2 with all three HTTP versions. Each experiment lasts 9 minutes and we repeat it 10 times. In total, we run 360 video sessions. We call this dataset *BrowserTime-Video* .

4.3.3 Performance Metrics

We rely on different performance metrics for the web browsing and video streaming scenarios. For each case, we select metrics that are known to be good proxies for users' Quality of Experience (QoE).

First, BrowserTime collects various statistics during emulated web browsing, including QoE-related performance metrics. We track two metrics that are correlated with users' QoE [56] during web browsing:

- **onLoad**: The time when the browser fires the `onLoad` event –i.e., when all elements of the page, including images, stylesheets and scripts, have been downloaded and parsed;
- **SpeedIndex**: Suggested by Google,[14] it represents the time at which the visible parts of the page are displayed. It is computed by recording the video of the browser screen and tracking the visual progress of the page during rendering.

Note that we use these metrics for both mobile and non-mobile browsing.

For video streaming, we rely on the following QoE-related metrics [57]:

- **Video resolution**: Image quality is fundamental to QoE and can be estimated from video resolution. We determine the bitrate by parsing the requested

[12] https://reference.dashif.org/dash.js/
[13] https://docs.quic.tech/quiche/
[14] https://web.dev/speed-index/

URLs. We then calculate both the average encoding bitrate per video session and the number of requests for chunks in each bitrate.

- **Playback Startup Delay (*PSD* for brevity):** This is the time between the user request for a video and the start of playback. Most players wait until a buffer (a few seconds) is filled before starting playback. We calculate the startup delay by measuring the time until the client receives the first video chunk.
- **Frequency of Video Downscale:** We evaluate how video resolution evolves in video sessions and track resolution switches. While switching is normal for adaptive video (e.g., to prevent video freezes), frequent switching affects QoE. We count how often the browser experiences a *downscale* in the requested bitrate.

4.4 Dissecting HTTP/3 Adoption

We now provide an overview of the adoption of HTTP/3. Since announcing HTTP/3 support does not equate to delivering content over this protocol, we also quantify the amount of content delivered over HTTP/3.

4.4.1 Websites Supporting HTTP/3

We use the *HTTPArchive* dataset to examine the extent to which HTTP/3 has been adopted since it was first proposed. The first IETF draft was published in January 2017, but we do not observe the first websites adopting HTTP/3 until late 2019. Since then, the number of websites supporting HTTP/3 has steadily increased. Figure 4.1 shows the trend for the last months of 2019, all of 2020, and the first 9 months of 2021. Using the `Alt-Svc` header, we can observe the HTTP/3 draft version supported by the server, shown with different colors in the figure. In case a website provides more than one version, we consider the last one seen in *HTTPArchive*. As of September 2021, we observe a significant number of websites supporting draft_34, which is on its way to becoming the final IETF standard for HTTP/3.

The figure shows that the number of websites supporting HTTP/3 has slowly increased, reaching 0.7 % of the total in early 2020. At that time, only Google and Facebook offered HTTP/3 for their websites. In February 2020, the number of

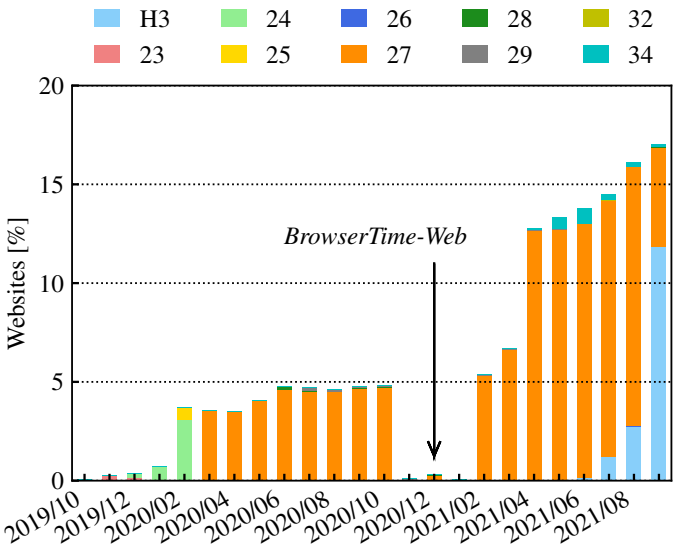


Fig. 4.1 Percentage of websites in HTTPArchive that announce support to HTTP/3, separately by IETF draft (*HTTPArchive* dataset).

websites supporting HTTP/3 jumped. This increase was due to CloudFlare enabling HTTP/3 for the websites it hosts. The percentage of websites supporting HTTP/3 increases over 4 % during this period, reaching a maximum of 4.8 % of websites (203 k) in October 2020. In November, the number of websites suddenly dropped to less than 0.1% (4 024 in absolute numbers). This gap was caused by CloudFlare suspending support for HTTP/3 due to performance issues.[15] CloudFlare re-enabled HTTP/3 for a subset of websites in December 2020. Since our *BrowserTime-Web* campaign took place in December 2020, we only consider these 14 707 websites for the following results.

The number of HTTP/3enabled websites remained low in the following months, mainly due to changes in the Cloudflare CDN configuration. Starting from February 2021, Cloudflare finally enabled HTTP/3 and we observe that the number of enabled websites returned to the October 2020 level. Since then, HTTP/3 support has reached 17.01%. In September 2021, 11.84% of websites announce support for the latest version of HTTP/3.

[15] https://community.cloudflare.com/t/community-tip-http-3-with-quic/117551, visited on 2/20/2021.

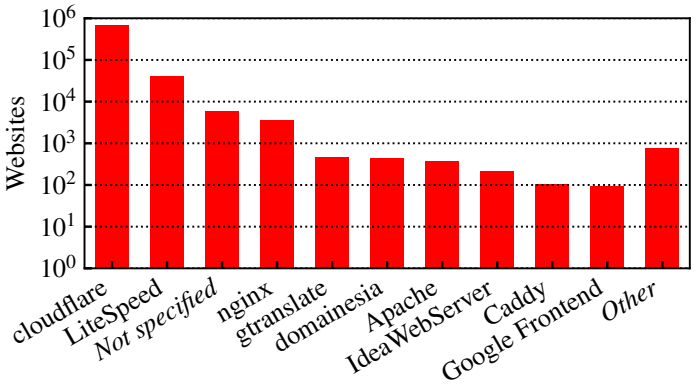


Fig. 4.2 `Server` in HTTP response (December 2020) (*HTTPArchive* dataset).

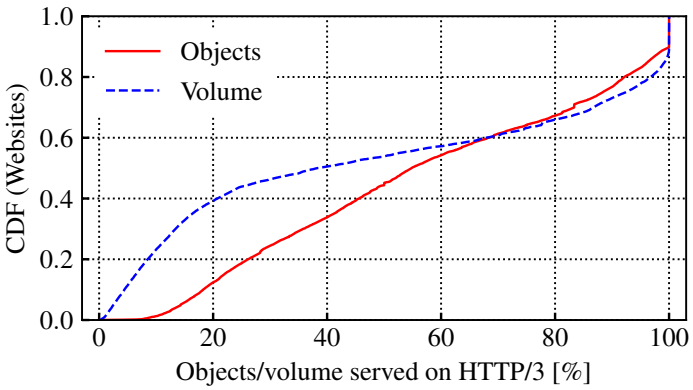


Fig. 4.3 Share of objects/volume served using HTTP/3 on enabled websites (*BrowserTime-Web* dataset).

The majority of websites supporting HTTP/3 are hosted by large enterprises running their own server applications. We break down these numbers in Figure 4.2, which lists the 10 most popular servers, as indicated in the HTTP `Server` header. As expected, CloudFlare hosts the most websites that support HTTP/3 (672 909, notice the log *y*- scale). In second position, we find LiteSpeed, a high-performance web server that supports HTTP/3. Looking at the server IP addresses, we notice that some popular cloud providers use it (e.g., OVH). For 5 957 websites, there is no reference to the `server` in the HTTP responses, and most of them belong to Facebook's domains - e.g., `facebook.com` and `instagram.com`. Google also supports HTTP/3, with `gtranslate` (Google Translate) and Google front-end servers. The remaining websites run other servers (e.g., nginx and Apache).

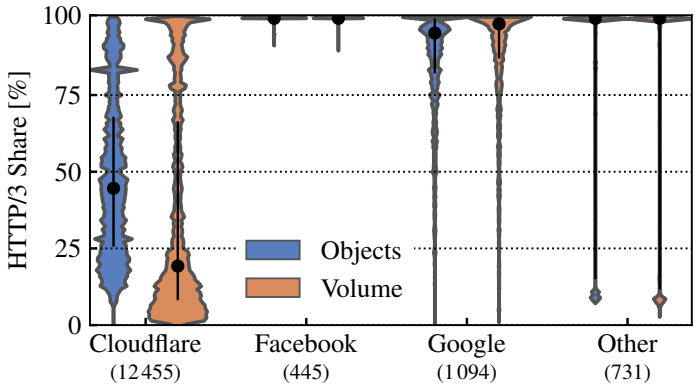


Fig. 4.4 Share of objects/volume served on HTTP/3, separately by provider (*BrowserTime-Web* dataset).

4.4.2 Content Served over HTTP/3

Next, we examine the extent to which objects are served by enabled websites using HTTP/3. This is because even if a website supports HTTP/3, not *all* of its objects are served over HTTP/3. Objects may be downloaded from external CDNs, cloud providers, or third-parties not supporting the same protocol. This is the case, for example, with ads and trackers, which are usually hosted on a different third-party infrastructure. We use the *BrowserTime-Web* dataset, which allows us to observe the protocol used to deliver each object that makes up the websites we visit.

In Figure 4.3, we consider all visits made with HTTP/3 enabled. For each visit, we compute the fraction of objects served via HTTP/3. Since each website is accessed multiple times, we calculate the average of the values over all visits. It is clear that at least the main HTML document is always sent over HTTP/3, but the remaining objects can be served with older HTTP versions. The figure shows the distribution of the percentage of objects transmitted over HTTP/3 (solid red line) and the byte-wise distribution (dashed blue line). Note that in 18% of cases all objects are delivered over HTTP/3, which means that the web page contains only elements hosted on HTTP/3-enabled servers. Web pages with 90% or more objects (volume) on HTTP/3 are 36 (41) % and only 9 (28) % have less than 20 % of objects (volume).

Next, we break down the above analysis by provider –i.e., by the company/CDN hosting the website. We get it by looking at the HTTP header `server`, the name of the website, and the IP addresses of the server. As shown in Figure 4.2, we find that

HTTP/3 is mainly used by (i) Cloudflare CDN, (ii) Facebook, and (iii) Google. The remaining 595 websites (i.e., Other) largely belong to self-hosted websites running updated versions of the *nginx* web server.

Figure 4.4 shows the percentage of objects and volume served over HTTP/3, separately by provider. Websites hosted by Cloudflare tend to be more heterogeneous, with half of the objects accessed via non-HTTP/3 servers (at median). In addition, only 24% of the volume is served via HTTP/3. This is likely due to the diversity of websites that rely on the provider. These websites may use complex web pages consisting of multiple third-party objects stored on external providers that do not yet rely on HTTP/3. In contrast, Facebook and Google serve almost all objects using HTTP/3. For Google, the long tail of the distribution is due to Blogspot, where the creator can add content from external sources. Finally, if we look at the Other category, almost all objects are served using HTTP/3. These websites are usually simple and consist of a few objects stored on the same self-hosted servers along with the main HTML document.

4.5 Web Browsing Performance

Now we investigate how HTTP/3 affects QoE-related web browsing performance metrics and whether the observed improvements can be related to the provider hosting the content (Section 4.5.1), website characteristics (Section 4.5.2), or mobile networks (Section 4.5.3).

We now investigate the impact of HTTP/3 on website performance. For this purpose, we use the *BrowserTime-Web* dataset in which the 14 707 web pages were visited multiple times under different network conditions. In addition to computing performance in the native scenario (i.e., 1 gpbs Ethernet in a campus network), we use `tc-netem` to enforce additional latency, packet loss, and bandwidth constraints. We then contrast the QoE-related performance indicators of the site (onLoad and SpeedIndex) by (i) displaying their absolute value and (ii) computing a metric that we call *H3 Delta*. Given a website and a given network scenario, we obtain the *H3 Delta* as the relative deviation of the metric when using HTTP/3 ($h3$) instead of HTTP/2 ($h2$). Since we always perform 5 visits for each case, we consider the median values. The *H3 Delta* for a website w in scenario s is calculated as follows:

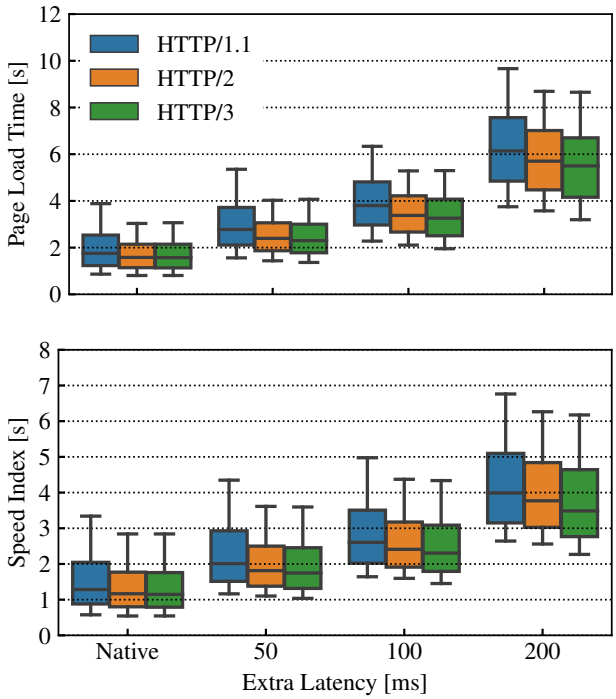
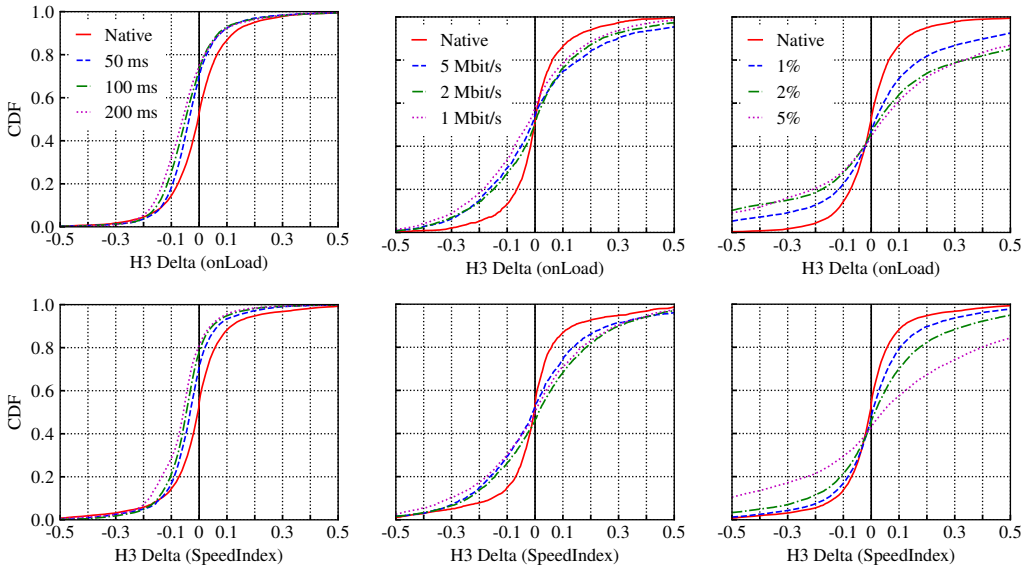


Fig. 4.5 onLoad (top) and SpeedIndex (bottom) with extra latency, separately for HTTP/1.1, HTTP/2 and HTTP/3 (*BrowserTime-Web* dataset).

$$\text{H3-Delta}(w,s) = \frac{\text{median}(w,s,h3) - \text{median}(w,s,h2)}{\max(\text{median}(w,s,h3), \text{median}(w,s,h2))} \tag{4.1}$$

By definition, H3 Delta(w,s) is bound in $[-1,1]$ and negative if a website loads faster under HTTP/3, and positive if not. We calculate the *H3 Delta* for both onLoad and SpeedIndex.

We illustrate how the values of the metric change under different network conditions by first focusing on the additional latency in Figure 4.5. Using boxplots, we show the distribution of onLoad (top) and SpeedIndex (bottom), separately by HTTP version (coloured boxes). The boxes range from the first to the third quartile, the whiskers indicate the 10^{th} and the 90^{th} percentiles, while the black dashes represent the median. When no additional latency is added (*native* case), we observe that the median onLoad time is about 2s, while SpeedIndex is about 1s, without much difference between HTTP versions. When adding additional latency, websites load slower as more time is needed to download the page objects, 6 seconds on median with 200 ms of additional latency. Not shown here for brevity, also packet loss and limited



(a) Latency. (b) Bandwidth. (c) Packet loss.

Fig. 4.6 *H3 Delta* on different scenarios. onLoad (top) and SpeedIndex (bottom). Negative values indicate that HTTP/3 is faster (*BrowserTime-Web* dataset).

bandwidth cause a similar degradation in performance indicators. Figure 4.5 shows that HTTP/1.1 has the worst performance at high latency, while HTTP/3 shows the greatest benefits. With an additional latency of 200 ms, websites onLoad in a median of 6.4, 5.8 and 5.4 s with HTTP versions 1.1, 2, and 3, respectively.

To better capture the differences between HTTP/3 and HTTP/2, we now examine the *H3 Delta* in Figure 4.6, where we show the distribution across the 14 707 websites for both onLoad (top row) and SpeedIndex (bottom row). The three columns respectively refer to scenarios with additional latency, limited bandwidth, and packet loss, respectively. Solid red lines represent the *native* case. Dashed lines represent scenarios with emulated network conditions as indicated in Table 4.2.

Starting with latency, we confirm what has already been shown in Figure 4.5. In the native case, we do not observe a general trend: looking at the solid red lines, we find that in about 50 % of cases, websites load faster with HTTP/3, and in the remaining cases HTTP/3 is slower. When latency is high, HTTP/3 offers significant advantages over HTTP/2. With an additional 50 ms of latency, 70 (71) % of websites have a lower onLoad time (SpeedIndex), meaning they load faster. The number of websites that load faster increases to 73 (77) % at a latency of 100 ms latency. At

200 ms, the number of websites that load faster reaches 74 (77) %, and the median *H3 Delta* is -0.059 (-0.056).

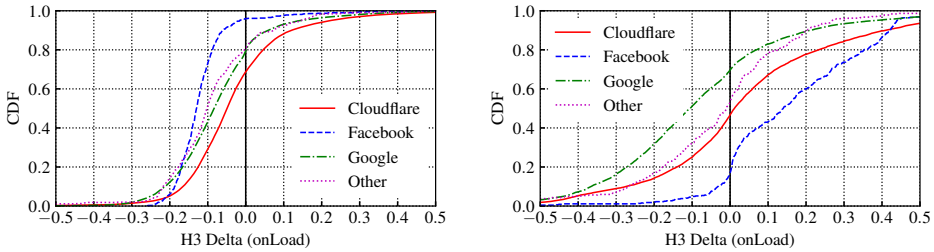
When we focus on bandwidth-limited experiments (central plots in Figure 4.6), different considerations hold. We observe (limited) benefits only for the onLoad time when bandwidth is limited to 1 Mbit/s, with 57 % of websites loading faster with HTTP/3. Note that this benefit does not come from indirect higher latency due to queuing delays (also called bufferbloat), since we limit host queues to 32 kB. In other cases, no clear trend emerges, but we do notice greater variability in the *H3 Delta* measure due to the constrained configuration. For example, in the case of SpeedIndex, 52, 45, 49 % of websites load faster with HTTP/3 with 5, 2 and 1 Mbps bandwidth. Similar considerations apply to packet loss (rightmost graphs in Figure 4.6). Despite greater variability, we cannot see an overall trend, and the *H3 Delta* values are evenly distributed above and below 0.

In summary, we observe limited improvements in onLoad time with very low bandwidth and substantial benefits for both metrics in the high latency case. We do not testify performance improvements of HTTP/3 in high packet loss scenarios. In fact, in several tested cases, some websites may even perform worse when HTTP/3 is enabled.

4.5.1 HTTP/3 Performance by Provider

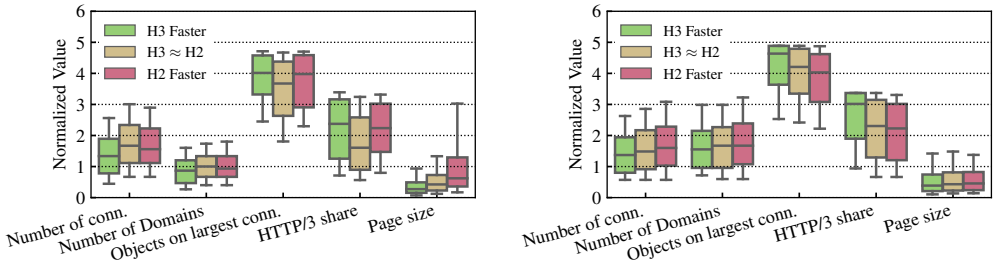
Next, we investigate whether the performance gains of HTTP/3 might be related to the provider hosting the websites. Since we observed considerable performance benefits for HTTP/3 only in cases with high latency or low bandwidth, we limit our analysis to these cases.

Figure 4.7 shows the distribution of *H3 Delta* for onLoad, separated by provider. We focus on scenarios with 200 ms of additional latency and 1 Mbit/s bandwidth limit. We find that the *H3 Delta* varies significantly by provider. When we focus on latency (Figure 4.7a), Facebook websites show the highest performance gain (*H3 Delta* -0.13 in median), which is represented by the blue dashed line in the figure. In addition, 95% of websites load faster with HTTP/3 than with HTTP/2. Cloudflare (red solid line) shows the lowest benefits, with only 72% of websites loading faster. Google and the rest of the websites are in the middle. Similar considerations hold for SpeedIndex, which is not shown here for brevity.



(a) Latency. (b) Bandwidth.

Fig. 4.7 onLoad *H3 Delta* by website provider for scenarios with extra-latency and bandwidth limit (*BrowserTime-Web* dataset).



(a) Latency. (b) Bandwidth.

Fig. 4.8 Visit characteristics vs. *H3 Delta* class (normalized values, *BrowserTime-Web* dataset).

With limited bandwidth (Figure 4.7b), we observe a completely different situation. Here, Facebook generally has the worst performance with HTTP/3, with 91% of its websites loading faster with HTTP/2. Conversely, Google (green dashed line) shows the best values, with a median *H3 Delta* −0.14 and 79% of its websites loading faster with HTTP/3. Cloudflare and the rest of the websites do not show a clear trend, with about half of the websites loading faster with HTTP/3.

4.5.2 Page Characteristics

Now we examine the characteristics of the pages and possible correlations with performance when using HTTP/3. To this end, we compute several metrics that describe the loading process of a web page and contrast them to understand if they have correlations with *H3 Delta*. For each visit to the 14 707 websites in our dataset, we calculate the following metrics in addition to the *H3 Delta*:

- **Number of connections** made by the browser to load the website when using HTTP/3.
- **Number of domains** contacted while loading the website (i.e., including third-party domains).
- **Share of objects on the largest connection**, which measures the percentage of objects transferred over the connection where the most objects were requested. Recall that HTTP/3 best practices recommend avoiding domain splitting to improve performance.
- **Share of objects served on HTTP/3**, which is used to investigate possible correlations between the share of objects transferred over HTTP/3 (in Figure 4.3) and the *H3 Delta* metric.
- **Page Size** to break down performance for small and large web pages.

In Figure 4.8, we compare the distribution of the above metrics and group the websites by classes defined by the onLoad *H3 Delta*:

- **H3 Faster**: websites loading faster with HTTP/3, i.e., onLoad *H3 Delta* < -0.1.
- **H3** $\approx$ **H2**: websites having a similar loading time in HTTP/2 and HTTP/3, i.e., onLoad *H3 Delta* $\in [-0.1, 0.1]$.
- **H2 Faster**: websites loading faster with HTTP/2, i.e., onLoad *H3 Delta* > 0.1.

In the figure, boxes with different colors represent these three classes. The y-axis represents the metrics normalized by scaling to a unit variance for ease of visualization and comparison. Again, we examine the scenarios with added latency (top row) and limited bandwidth (bottom row) as they provide the most interesting insights. With additional latency, H3 Faster websites are 32%, H2 Faster 11% and H3 $\approx$ H2 are 57%. With limited bandwidth, they are 38%, 25% and 37%, respectively.

We first focus on the left group of boxes in Figure 4.8, which shows the (normalized) number of connections the browser established to load the web page. Green boxes indicate that websites that make fewer connections (smaller metric values) are faster with HTTP/3 than with HTTP/2. This is true for both scenarios, i.e., low

latency and low bandwidth. Similar considerations apply when we focus on the second set of boxes, which represents the number of domains contacted. Indeed, we find that the number of connections per site and the number of domains contacted are 0.91-correlated (Pearson correlation). The third set of boxes provides a similar perspective, measuring how web page objects are split across multiple connections/domains. The web pages that benefit the most from HTTP/3 are those that tend to collect objects on a single connection – see the highest position of the green boxes, which means more objects are on a single connection. This is very clear when bandwidth is limited (Figure 4.8b) and not when latency is high (Figure 4.8a).

Serving most objects using HTTP/3 (instead of HTTP/2) also has a positive effect, as we can see from the fourth group of boxes in Figure 4.8. Again, this is most evident in the case of bandwidth limitations (Figure 4.8b), while it is hard to see a clear trend in the case of additional latency (Figure 4.8a). Finally, the case of web page size (last box group) is interesting. In high latency scenarios, the websites that benefit from HTTP/3 are small, while large websites tend to do better with HTTP/2. When bandwidth is tight, the picture is more even: again it is the small websites that load faster with HTTP/3, albeit only moderately.

In summary, the websites that benefit from HTTP/3 are those that limit the number of connections and third-party domains, fully adopt HTTP/3 for all website objects, and limit page size. These considerations hold for high latency or limited bandwidth scenarios, while we do not observe a clear trend for optimal network conditions or high packet loss, where the distributions of the metrics largely overlap.

4.5.3 Performance for Mobile Users

We now use the *BrowserTime-Mobile* dataset to investigate how HTTP/3 impacts the browsing performance of Internet users. To this end, we evaluate the impact of using different devices (i.e., smartphone and table) that we emulate with the BrowserTime features. Then, as described in Section 4.3.2, we emulate different mobile network conditions using ERRANT [19]. Here, we restrict our analysis to the 100 selected websites and measure the gain of HTTP/3 by computing the *H3 Delta*.

To give a compact overview of our results, we plot the median *H3 Delta* across all 100 websites in Figure 4.9 in the form of a heatmap, separately by OnLoad (Figure 4.9a) and SpeedIndex (Figure 4.9b). We emulate three different devices: an

(a) onLoad.

(b) SpeedIndex.

Fig. 4.9 Web Browsing Performance of Mobile Users, separately by user device type and emulated network (*BrowserTime-Mobile* dataset).

iPhone 6 smartphone, an iPad tablet, and the default desktop version of Chrome. We arrange them in the columns of the heatmap. The different rows represent the 6 ERRANT network profiles, including 2 Radio Access Technology (RATs) (3G and 4G) and three signal quality levels. We refer the reader to [19] for the details of the emulated network conditions. In summary, ERRANT emulates the typical latency, uplink and downlink bandwidths measured in a large-scale speed test measurement campaign with 4 European network operators. Importantly, ERRANT emulates not only the average conditions, but also their variability measured during the speed tests at training time.

Starting from OnLoad in Figure 4.9a, we find that HTTP/3 performs better than HTTP/2 in all scenarios – note the negative median of the H3 Delta between -0.05 and -0.14. Websites that benefit from HTTP/3 range from 66% to 88% depending on device and network conditions. Desktop websites generally see the greatest improvements – see the last column in the figure-but smartphones and tablets also see improvements. In terms of network conditions, the improvements are more pronounced on Good 4G, the best network profile we tested. Similar conclusions

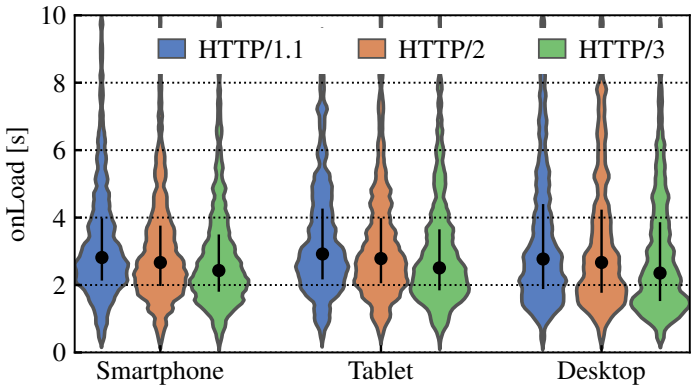


Fig. 4.10 onLoad time with emulated 4G good network (*BrowserTime-Mobile* dataset).

apply to the SpeedIndex (Figure 4.9b). Visits with an emulated mobile device cause the browser to render the *mobile* versions of the web pages. As a result, they tend to be easier to render and therefore smaller in terms of bytes. In fact, the average page size is 680 kB for desktop versions, which reduces to 630 kB in the case of tablet versions and 402 kB for smartphone versions. We also find that latency on mobile networks is on the order of $50-150$ms, as measured in the real experimental campaign behind ERRANT [19]. These results are consistent with those in Section 4.5 and show that HTTP/3 offers significant benefits in high latency scenarios.

Figure 4.9 clearly provides a rough median of the H3 Delta, but the absolute metrics exhibit large variability due to the varying characteristics of the 100 websites and the variable network conditions imposed by ERRANT. We illustrate this variability using the Good 4G profile as an example by plotting the distribution of onLoad time using violin plots in Figure 4.10. The distributions are wide, as we notice from the black vertical line within each violin, which represents the interquartile range (IQR). For example, in the case of a smartphone using HTTP/1.1, the onLoad time ranges from 2s to 4s. For HTTP/2 and 3, the onLoad time tends to decrease. The median is 2.8s for HTTP/1.1, which drops to 2.6s for HTTP/2 and 2.4s for HTTP/3. Similar considerations hold for the other cases, with the median onLoad value decreasing by about 0.2s for newer HTTP versions. In summary, our experiments show that HTTP/3 provides significant benefits in mobile networks, even when users use mobile devices that retrieve the lighter mobile versions of websites.

Table 4.3 Summary of the takeaways from *BrowserTime-Video* .

	Resolution	**Speedup**	**Downscale**
Native	No difference	No difference	No difference
Loss	No difference	minor difference	None for h3 Rare for h1 and h2
Bandwidth	h3 worse	h3 little slower h2 h2 faster h1	Many for h3 Few for h2 and h1
Latency	No difference	h2 faster h1 h3 slower h2	No difference

4.6 Performance of Adaptive Video Streaming

4.6.1 Metrics

In this section, we study the performance of HTTP/1.1, HTTP/2, and HTTP/3 in video streaming using the *BrowserTime-Video* dataset. To do so, we conduct experiments with different network conditions *NC* and measure the quality of video streaming using metrics known to correlate with subjective QoE [58, 57]. In particular, we focus on three QoE-related metrics, namely: (i) *video resolution*, (ii) *playback startup delay (PSD for brevity)*, and (iii) *number of downscale adjustments*.

To ease the comparison between different HTTP versions, we compute the *Speedup* of *PSD* similar to the H3-Delta defined in Equation 4.1. Specifically, given two experiments j and k, we compute the speedup as follows:

$$\text{Speedup}(j,k) = \frac{PSD(j) - PSD(k)}{\max(PSD(j), PSD(k)} \tag{4.2}$$

Since we are interested in comparing experiments with the same network conditions $nc \in NC$, we define the set of experiments $S_{i,nc}$ for HTTP version $i \in \{1,2,3\}$. Finally, we compare successive HTTP versions by measuring how the median Speedup changes. Thus, given a network condition nc and an HTTP version i, the median Speedup $MS_{i,nc}$ is simply:

$$MS_{i,nc} = Median_{j \in S_{i,nc}, k \in S_{i-1,nc}}(\text{Speedup}(j,k)) \;\; i \in \{2,3\} \tag{4.3}$$

4.6.2 Results

We summarize our key findings on video streaming results in Table 4.3. Starting from the case without network impairments (i.e., the *Native* case in Table 4.3), we note that regardless of the HTTP version used, the chunks are always delivered at the highest resolution ($4k$) and without downscaling. Also, when looking at the Average Speedup (see also Figure 4.11), we cannot see any radical differences between the HTTP versions. In sum, when the network has good quality, all HTTP versions perform similarly.

When we introduce controlled *Packet Loss*, some initial differences start to appear. Specifically, for HTTP/1.1 and 2, we observe some rare video adjustments (downscale) where the resolution of the chunks jumps between $1920 \times 1080p$ and $4k$. In contrast, HTTP/3 proves to be more stable, with all chunks delivered at $4k$ resolution. Considering other metrics (video resolution and Speedup), we see nearly the same figures across all HTTP versions, e.g., with video chunks delivered at either $4k$ or $1920 \times 1080p$.

More interesting differences emerge when we consider bandwidth limitations. We find that HTTP/3 results in lower resolution video and more frequent video adjustments. We show this effect in Figure 4.12, which shows the distribution of the number of downloaded chunks in relation of their corresponding bitrate.[16] Recall that we varied the bandwidth between 1 and 5 Mbit/s and, as expected, the video quality does not exceed $1024 \times 576p$. HTTP/1.1 and HTTP/2 show the same performance, while for HTTP/3 we find a higher percentage of chunks delivered at the lower resolutions.

This result is also detailed in Table 4.4, which gives the average number of resolution downscales for each scenario. At only 1 Mbit/s, all HTTP versions struggle and result in a high (and similar) number of downscales (first line in the table). At 5 Mbit/s, the situation improves and the video settles at $1024 \times 576p$ (third line in the table). Interestingly, HTTP/1.1 and HTTP/2 show stable performance with almost no adaptation in the 2 Mbit/s case, while HTTP/3 averages 13.3 downscales per experiment for the 150 chunks (second line in the table).

To explore further this phenomenon, Figure 4.13 shows an example of the quality level of the chunks for an HTTP/2 and an HTTP/3 video session at 2 Mbit/s. On the

[16] We do not use resolution because some of the chunks in our setup are encoded at two bitrates.

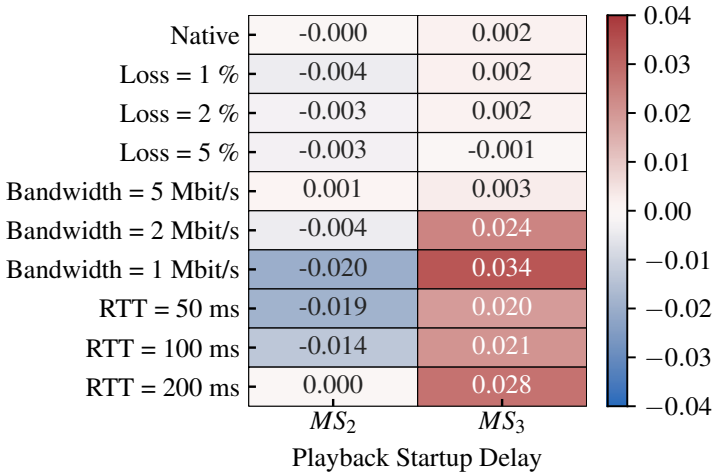


Fig. 4.11 Median Speedup per network condition (*BrowserTime-Video* dataset).

second chunk, the player downgrades with HTTP/2, but then immediately stabilizes the quality level to 1000 kbit/s ($640 \times 360p$). In contrast, the player with HTTP/3 is more eager for high-resolution chunks and fetches the video several times at the 1500 kbit/s quality level, i.e., $768 \times 432p$. However, the lack of bandwidth results in downgrading to the lowest possible resolution.

While we cannot clearly identify the causes for this behavior, we conjecture that it is caused by complex interactions between several components. Recalling that HTTP/3 runs on top of UDP/QUIC with a different congestion control algorithm, the client player seems more aggressive in this scenario. For example, we observe cases in which the player requests multiple times the same chunk with different resolutions, probably as a reaction to delays in lower network layers.

Looking at the Speedup metric, we see that HTTP/2 improves greatly over HTTP/1.1, while HTTP/3 has the worse performance, especially at very low bandwidth (0.034). We provide further details about this result in Figure 4.11, which show the median Speedup for HTTP/3 (h3) and HTTP/2 (h2) for different network conditions. The bluer the color, the greater the benefits of the newer HTTP version. Conversely, the red color indicates that the newer HTTP version results in a larger *PSD*, i.e., slower transmission of the first chunk

Finally, when we impose additional latency (last row in Table 4.3), almost all chunks are delivered at the highest resolution, and we find that video resolution

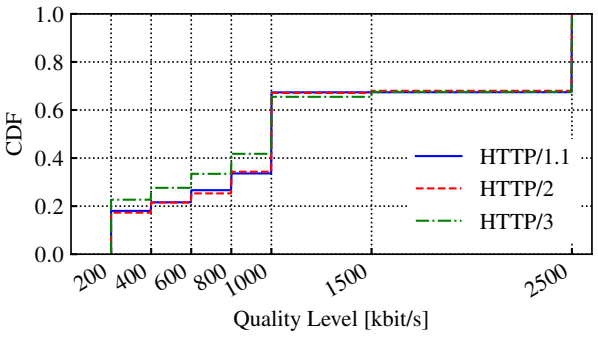


Fig. 4.12 Distribution of chunks resolution with limited bandwidth (*BrowserTime-Video* dataset).

Table 4.4 Mean number of downscale per experiment.

	HTTP1/1	**HTTP/2**	**HTTP/3**
Bandwidth = 1 Mbit/s	38.7	43.3	38.0
Bandwidth = 2 Mbit/s	1.0	1.0	13.3
Bandwidth = 5 Mbit/s	1.9	2.0	2.0

adjustments are very rare (i.e., no differences between HTTP versions for this metric). Looking at the median in the last three lines of Figure 4.11, we find that HTTP/3 tends to be slower than HTTP/2, i.e., the first video chunks take more time to be delivered to the client. We again conjecture that this effect is due to the different congestion and flow control algorithms implemented in QUIC, and their interactions with the player. Investigating how performance could be improved by changing the video server configuration (note that we use *nginx HTTP server* with *quiche* HTTP/3 implementation) is left for future work.

4.7 Takeaways

This chapter presents a comprehensive study on the adoption and performance of HTTP/3, highlighting its impacts in various network scenarios and its gradual adoption by leading Internet companies since 2020. Our findings can be summarized as follows:

- **Adoption Trends:** The adoption of HTTP/3 began in late 2019, with a notable increase in website support over time, especially with the involvement of

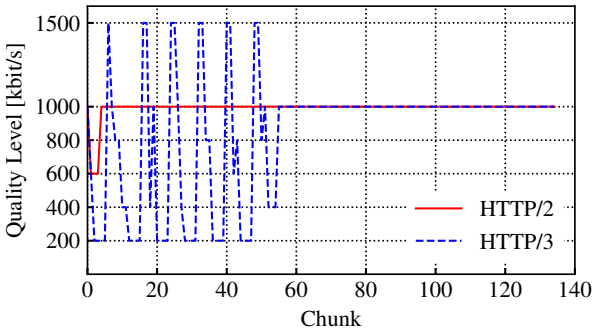


Fig. 4.13 Example sessions with 2 Mbit/s bandwidth (*BrowserTime-Video* dataset).

major entities like Google, Facebook, and Cloudflare. By September 2021, approximately 11.84% of websites had adopted HTTP/3, with Cloudflare being a significant host.

- **Content Delivery:** Despite the growing support for HTTP/3, a substantial amount of web content, particularly from third-party sources, continues to be delivered over older HTTP versions.
- **Web Browsing Performance:** HTTP/3 shows considerable performance improvements in high-latency environments and offers limited benefits under low bandwidth. However, it does not significantly outperform HTTP/2 in high packet loss scenarios. Websites hosting most content on third-party HTTP/2 servers show less improvement.
- **Provider-Specific Performance:** The performance gains from HTTP/3 vary significantly across different hosting infrastructures. Sites hosted on platforms like Facebook and Google show notable improvements in specific network conditions.
- **Mobile Browsing:** HTTP/3 enhances browsing performance for mobile users, with smartphones and tablets benefiting from faster onLoad times and improved SpeedIndex metrics across various network conditions.
- **Video Streaming:** In controlled environments, HTTP/3 does not demonstrate notable benefits for adaptive video streaming, with performance similar to or worse than HTTP/2 under bandwidth constraints.

- **Overall Impact:** The study underscores that HTTP/3 brings more significant benefits in scenarios with high latency or limited bandwidth, particularly for websites that minimize third-party domain loading.

In conclusion, our large-scale measurement campaign highlights the growing adoption of HTTP/3 and its impact on web browsing and video streaming performance. While HTTP/3 offers considerable improvements in specific scenarios, its advantages vary depending on network conditions and hosting infrastructures.

Chapter 5

Satellite Network Measurements

5.1 Motivation

The work we present in this chapter is mostly taken from our paper *When satellite is all you have: watching the internet from 550 ms*, published in *Proceedings of the 22nd ACM Internet Measurement Conference* [59].

While 5G and Fiber-To-The-Home (FTTH) technologies give us access capacity on the order of Gb/s [60–63] and Content Delivery Networks (CDN) can guarantee end-to-end delay in the range of milliseconds [64–66], there are significant parts of the world where economic and technological constraints force people to rely on solutions that provide far more constrained access to the Internet. These include mountainous and rural areas in developed countries, as well as the entire territory of underdeveloped countries, where even the supply of stable electricity can be problematic. In such scenarios, Satellite Communications (SatCom) offers a practical connectivity solution. Among the available SatCom technologies, geostationary (GEO) satellites are the oldest and most widely used solution [67], with the first offerings dating back to the early 2000. Here, a satellite orbits the Earth at an altitude of about 36 000 km, and moves at an angular velocity equal to the Earth's rotational speed. To an observer on Earth, the satellite appears immobile, making it easier to establish a communication link. A single GEO satellite can cover entire continents, and the directional beams enable efficient space and frequency multiplexing, with each beam providing a total channel capacity on the order of 10 Gb/s [68].

In addition to limited shared capacity, GEO satellite communications suffer from high propagation latency, which is about 550 ms for a round trip (including two passes through the satellite). For this reason, complicated Medium Access Control (MAC) and scheduling protocols coordinate the access and sharing of the satellite's uplink and downlink, while traffic shapers, Performance Enhancing Proxies (PEP), and TCP optimization solutions attempt to mitigate the effects of end-to-end delay and limited capacity [69–71].

In this chapter, we have the unique opportunity to present the first large-scale passive characterization of a global GEO SatCom Internet access solution. Through passive instrumentation of the satellite ground station, we observe traffic from tens of thousands of customers in more than 20 countries in Europe and Africa. On the one hand, this allows us to characterize the different Internet usage habits in the different scenarios, if any. On the other hand, we observe the impact of SatCom technology on performance and identify possible optimization strategies.

The main observations can be summarized as follows:

- In Africa, chat and social media applications consume 100 and 10 times more data than in Europe. This is due to the presence of community WiFi points that share SatCom access.
- Since these applications are accessed throughout the day, the typical peak time in African countries is anticipated in the morning.
- Satellite channel protocols and solutions increase the Round Trip Time (RTT) much more than just the propagation delay. Link channel quality and congestion (if any) can actually add seconds to the end-to-end RTT.
- In SatCom networks, all traffic must pass through the same ground station, in our case in Europe. This impacts local popular services. For example, services in Africa suffer from the additional delay caused by traffic being routed back and forth through the ground station.
- To complicate the picture, most customers use open DNS resolvers, some of which are located in China and Africa. This increases DNS response time to hundreds of milliseconds and jeopardizes the server selection policies of CDNs and DNS resolvers.

We believe that the characterization offered in this paper contributes to understanding the complexity of the Internet by providing a novel perspective on SatCom networks and customers. We also discuss possible technical improvements that SatCom providers might consider to improve the quality of service offered to their customers. These include the use of additional ground stations to route traffic more efficiently and the control of DNS requests or responses to limit the impact of incorrect server selection. To allow the research community to conduct experiments with an emulated GEO SatCom connection and compare it to other connection technologies (including the novel Starlink connection with data from [72]), we have created a data-driven model for our ERRANT network emulator tool [19] and make it available at https://github.com/SmartData-Polito/errant.

In the remainder of the chapter, we present first the related work (Section 5.2), then we introduce the SatCom technology and monitoring infrastructure (Section 5.3) and provide an overview of our dataset (Section 5.4). We then illustrate our results in terms of volumetric traffic distribution (Section 5.5), user habits and service consumption (Section 5.6), and performance indicators (Section 5.7). Finally, we discuss and draw conclusions and takeways (Section 5.8).

5.2 Related Work

Since the Internet was born, characterization of network traffic has been an important topic to understand its evolution and usage. Seminal works analyze trends of Internet traffic trends in the 1990 s and early 2000 s [73, 74]. Since then, there has been a large body of research using passive measurements to understand Internet operations and usage patterns. Most works focus on characterizing Internet traffic generated by cable/fiber customers [75, 60, 76, 64], while others analyze traffic captured in the backbone network, mixing users with different access technologies [77, 78]. Recently, researchers have focused on analyzing the impact of the Covid-19 pandemic on Internet traffic, noting sudden and remarkable changes [79–81]. Our work is in this area and uses a similar methodology for collecting and analyzing passive measurements. We note that most measurements are related to Europe and North America, while we also provide insights into African Internet traffic, similar to Johnson *et al.* [82]. In addition, in this work, we have a unique opportunity to specifically study traffic from SatCom subscribers. To the best of our knowledge, we are the first

to conduct a longitudinal study of SatCom traffic and present a comparison of traffic from different countries on two continents.

Satellite-based consumer Internet access [67] was launched in 2003 and has been the subject of a wide corpus of literature. Most of the work targets the design and optimization of communication channels, and a few studies successfully cover the topic [83, 68, 84]. Other work focuses on measuring and improving the performance of Internet protocols in SatCom. Tropea *et al.* [85] evaluates different TCP versions on geostationary satellite links, while Peng *et al.* [86] and Muhammad *et al.* [87] focus on the interaction of TCP and PEPs. More recently, the performance of QUIC and its interaction with PEPs have been studied [88–90]. For a complete benchmark of SatCom performance, see Deutschmann *et al.* [91], which provides figures on SatCom latency, throughput, and web page load time. Recently, the authors in [92, 72] examined Starlink, the new low-orbit satellite communication system, which provides a comprehensive overview of the impact of this new communication on user-perceived performance when accessing globally distributed resources and the impact of different HTTP versions. Similarly, Mohamed *et al.* [93] propose a study from different vantage points to understand how the performance changes from the browser's perspective.

All of these works rely on active measurements, i.e., they deploy test environments with devices using SatCom and analyze the results. In this work, we provide a different perspective by providing performance data on SatCom through passive measurements by observing traffic from about 10 k customers and complementing the results of previous work.

5.3 Measurement Setup and Methodology

In this section, we describe our measurement setup and the methodology we use to gather and analyze data from the actual deployment of a large international SatCom operator. We first provide an overview of the specific data-link technologies that can affect Internet access performance, but avoid detailing the complexity of the physical layer of SatCom transmissions. In this section, we describe our measurement setup and the methodology we use to gather and analyze data from the actual deployment of a large international SatCom operator. We first provide an overview of the specific

data-link technologies that can affect Internet access performance, but avoid detailing the complexity of the physical layer of SatCom transmissions.

5.3.1 The SatCom Network

In traditional SatCom networks, the operator has deployed a satellite infrastructure consisting of satellites in geostationary orbit, and a ground infrastructure. Referring to Figure 5.1, subscribers employ a dedicated equipment, i.e., the customer-premises equipment (CPE), to connect their devices (PC, smartphone, etc.) to the SatCom network. The CPE consists of a dish antenna and a router/modem that manages the satellite links and access protocols on one side, while offering WiFi and Ethernet connectivity on the other side.[1] The satellite acts as relay for subscriber traffic, which traverses 35, 786 km twice to reach the ground station, accumulating from 240 ms to 280 ms, depending on the location on Earth of the subscriber. The ground station terminates the satellite segment and forwards the traffic to the Internet.[2] Notice that this forces all traffic to enter the Internet from the location where the ground station is. In our measurement setup, we monitor the traffic managed by one satellite in geostationary orbit. This satellite offers service in Europe and Africa, from Ireland to South Africa. At the time of data collection, the satellite operator operates a single ground station in Italy, through which all traffic passes to reach all Internet services.

The satellite is equipped with multiple directional antennas, each managing a transmission beam that points to a specific region on Earth. This allows the reuse of frequencies to increase overall capacity while optimizing the use of spectrum reserved for satellite communications. Each beam acts as a separate and independent physical channel, providing aggregate capacity on the order of Gb/s, the actual capacity being configurable. Two separate beams (and frequencies) cover each area, one for the uplink (from users' CPE to the satellite) and one for the downlink (from the satellite to users' CPE). A separate beam pair also connects the satellite and the ground station.

On the shared uplink channel, the transmission of packets involves a complicated MAC protocol: a slotted-Aloha protocol allows the CPE to access the shared reser-

[1] A customer may represent a single individual, a household, a company's office, or a community-based WiFi Internet access solution.

[2] The total round trip time (RTT) of any communication is in the order of 550 ms since the packets must go through the satellite link on both the forward and backward paths.

vation channel the first time it needs to transmit. Then, a Time Division Multiple Access (TDMA) scheduling protocol run by the satellite allocates time-slots to each active CPE to avoid collisions and to fairly share capacity among the active users at each TDMA frame. The satellite then forwards the packets to the ground station via a dedicated high-capacity beam.

On the downlink channel, the ground station transmits the packets directly to the satellite, which then forwards them to the destination CPE by selecting the correct frequency and beam. In this case, the packets are broadcasted to all receivers, which filters those destined for their CPE MAC address and discard the packets destined for other CPEs. In addition to the TDMA and MAC schemes, Forward Error Correction (FEC) and Automatic Repeat Request (ARQ) mechanisms provide a reliable data-link service. All in all, these proprietary algorithms provide a reliable, almost error-free, bi-directional point-to-point link between each CPE and the ground station. By combining these MAC, scheduling, FEC and ARQ protocols, further random delays are added to the communication between the CPE and the ground station.

To mitigate the potential performance degradation caused by this high latency, the SatCom operator relies heavily on a Performance Enhancing Proxy (PEP) to improve TCP performance on the satellite segment. A PEP is a network component that improves end-to-end performance by transparently manipulating TCP connections. Defined in RFC 3135 [94], PEP works as follows in our case. In the lower part of Figure 5.1, the subscriber CPE acts as a transparent TCP proxy for the end user's TCP traffic. It terminates all TCP connections initiated by applications on end-user devices and forwards TCP payload to the ground station via a bidirectional reliable tunnel over UDP. In detail: When a subscriber's device initiates a new TCP connection via a SYN packet, the CPE impersonates the destination server and immediately completes the TCP three-way handshake, allowing the client application to send the initial data with no delay.

Here, the CPE acts as a L4 proxy. It buffers the TCP data stream and forwards it to the ground station via the bidirectional UDP tunnel at the allowed uplink rate. The ground station again works as a L4 proxy. When it receives a *Connect* request from the CPE, it establishes a new TCP connection to the actual destination server. It then forwards the data to/from the CPE via the downlink/uplink satellite tunnel. In this way, the TCP congestion control algorithm is effectively decoupled, allowing the ground station proxy to retrieve the data from the origin server at the backbone-path

rate and the CPE to forward the data to the end user's device as quickly as possible. Note that the download rate of the ground station from the origin server is still regulated by the download rate of the end device because the buffer capacity of PEP in the ground station is limited. Note that user traffic using UDP (e.g., DNS, QUIC) cannot benefit from PEP acceleration and therefore UDP packets are forwarded as is.[3]

At last, the ground station also acts as a Network Address Translation (NAT) box, DNS resolver, and supports Quality of Service (QoS) schedulers to prioritize and shape traffic depending on the application. To this end, the SatCom operator uses L3/L4 and domain name-specific rules to prioritize interactive traffic and shape video streaming flows. The shaper allows also to enforce commercial maximum capacity of up to 5 Mb/s in the uplink, and 10, 20, 30, 100 Mb/s in the downlink based on the subscriber's contract.

In our setup, the SatCom operator provides private IPv4 addresses to each customer CPE. This means that all connections must be initiated by an end-user client and no server can be run on the customer's premises.[4]

5.3.2 Passive Measurements

We instrument the SatCom operator's network to collect passive measurements of all subscriber traffic. To this end, we deploy a passive probe at the operator's ground stations in Italy. Here we collect all traffic after the operation of the PEP, which handles traffic in the satellite segment. We observe all packets exchanged by each customer – that we uniquely identify by their SatCom CPE IPv4 IP address. Using a router span port, we mirror both downlink and uplink traffic to a high-end measurement server equipped with two Intel X710 network cards. The server runs Tstat[38], a custom flow monitoring software that generates rich per-flow summaries in real time from the processed data packets. To handle the high rate, our software resorts to the Data Plane Development Kit (DPDK) library for packet capture [95], which enables accelerated packet processing by bypassing the kernel-space drivers and protocol stack, and guarantees that all packets are processed in real time without information loss. Using the classic 5-tuple, Tstat identifies and tracks the evolution

[3]The PEP can only act as a L4 proxy without violating the authenticity provided by TLS.

[4]The SatCom provider offers hosting of servers in the data center for customers interested in running services.

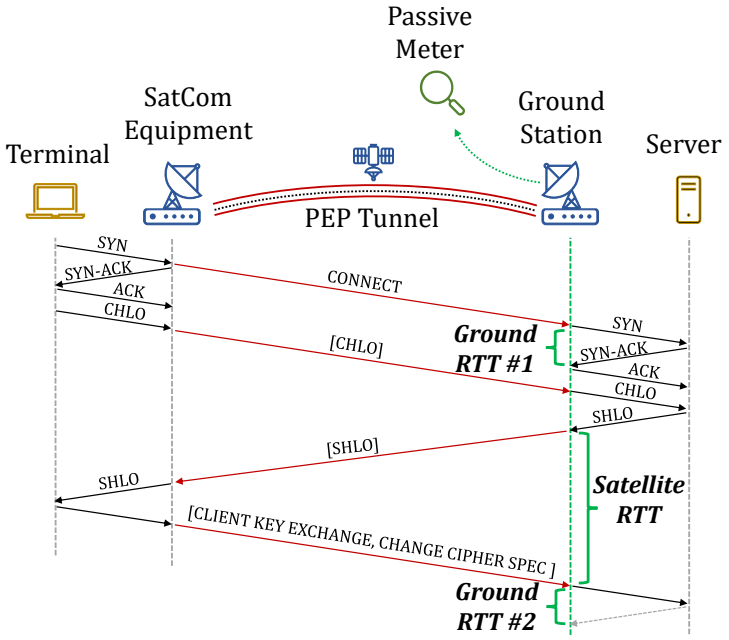


Fig. 5.1 Methodology for the estimation of the Satellite Segment RTT.

of TCP and UDP flows. For each flow, it extracts hundreds of statistics for both flow directions. The metrics we mainly rely on are: the i) flow size and duration, ii) the timing information of the first 10 packets, iii) the server and client IP address, iv) the TCP RTT between data and ACK segments, and v) the name of the contacted server as retrieved by the SNI, HTTP or DNS protocol..

The measurement of the TCP RTT deserves a careful explanation because the presence of PEP makes the measurement of RTT particularly troublesome. Indeed, the PEP causes the total RTT to be divided into three components, as shown in the bottom part of Figure 5.1: (i) *home RTT* – between the user device and the user's SatCom CPE; (ii) *Satellite RTT* – between the CPE and the ground station, where TCP segments are forwarded over the satellite PEP tunnel; and (iii) *ground RTT* – between the PEP terminator at the ground station and the destination server. Our vantage point is co-located with the ground station and therefore observes traffic exchanged from the PEP proxy to the Internet. For the ground RTT measurement (iii), TSTAT uses the TCP connection initiated by the ground station PEP. For each TCP segment sent, it measures the time to the corresponding ACK, calculating the minimum, maximum, average, and standard deviation of all RTT samples in a TCP flow. To measure the RTT (ii) of the Satellite segment, we need an additional ingenuity. Specifically, we leverage the TLS handshake of a TCP data flow to

measure the time from the Server Hello message to the next Client Key Exchange message/Change Cipher Spec message. This time also includes the Home RTT (i), which we can consider negligible compared to the satellite. In this way, and only for TLS flows completing the TLS negotiation, we can safely estimate the delay caused by the satellite at least once per flow.

At last, for both TCP and UDP, the software runs a Deep Packet Inspection (DPI) module that identifies the most popular protocols and extracts various information from headers. In particular, it annotates each flow with the server *domain name* as extracted from the *Host* header in case of plain-text HTTP, or from the *Server Name Indication* (SNI) field in the case of TLS or QUIC flows.[5] For DNS traffic, the software logs each requested domains and obtained responses, including the DNS server IP address the client used to resolve the name.

5.3.3 Ethical Aspects

Passive monitoring involves capturing and processing traffic generated by human beings, thus we need to take proper actions to protect as much as possible the individual's privacy. Indeed, IP address is considered a Personally Identifiable Information (PII) and it can be used to identify and track individuals. The characteristics of traffic, such as the list of visited websites, can be considered Sensitive Personal Information (SPI), as they can reveal personal aspects and habits of an individual.

For this work, we take all possible countermeasures to properly handle our measurements. First, the setup, management and data collection were physically managed uniquely by the operator personnel, who control the data collection process. Second, we configured the data collection to limit as much as possible the exposed information. We process packets in real time and save only strictly required information in flow logs. In details, we do not store any information present in headers that can be associated with a single user. Customers IP addresses are anonymized in real time using the CryptoPan algorithm [96] which preserve the subnet structure of the original IP addresses. The only sensitive information remains the server IP addresses and the domain names customers' visited, which we process only to extract aggregated statistics for the most popular services. Third, we only have access to the anonymized logs that we store in our secure Hadoop cluster, which is not reachable

[5]We use the term *domain* meaning Fully Qualified Domain Name.

from the public Internet and has strictly controlled physical access. The operating system and software are kept up to date to avoid possible vulnerabilities, and strict user access policies limit the access to the data only to authorized users.

The operator SatCom Data Protection Officer (DPO) has approved the above process and we have verified with our institutional review board that the data we collect is exempt from their approval.

5.4 Dataset Processing and Overview

In this paper, we consider all data collected during February-April 2022, resulting in 4.3 PB, for a total of 34.4 billion flows. On a daily basis, we transfer the flow summaries from the measurement server located in the SatCom provider premises to the Hadoop storage cluster, where we post-process the data using Apache Spark with custom designed analytics to compute various statistics and distributions.

5.4.1 Data Enrichment and Aggregation

In the first step of processing, we enrich the data by adding information about the customer's country (obtained by mapping the encrypted customer subnet to the corresponding country with the support of the SatCom operator) and about the service offered by the server. We focus here on six classes of services: Video Streaming, Social Networks, Audio Streaming, Chat, Work-related applications, and Search Engines. We rely on custom regular expressions that map popular server names to services. In detail, for each service class, we enumerate the top and local players by manually inspecting the list of most popular domains by volume and popularity. For each service, we enumerate the list of fully-qualified domains and second-level domains used to serve its content.[6] In some cases, we use regular expressions to generalize the set of domains. This is the typical case of CDN server names, which often include numbers or country codes in the domain. We uniquely use the domain to classify the service, as we do not capture the full HTTP URL, which is typically encrypted within the TLS session. For TLS flows, we obtain the domain from the SNI field of the Client Hello messages. As a result, we are sometimes unable to

[6]We handle the case of two-label top level domains – e.g., `co.uk`.

Table 5.1 TCP/UDP traffic breakdown by protocols.

Protocol	Volume share
TCP/HTTPS	56.0 %
TCP/HTTP	12.1 %
Other TCP	7.0 %
UDP/QUIC	19.6 %
UDP/RTP	1.1 %
UDP/DNS	< 0.1 %
Other UDP	4.2 %

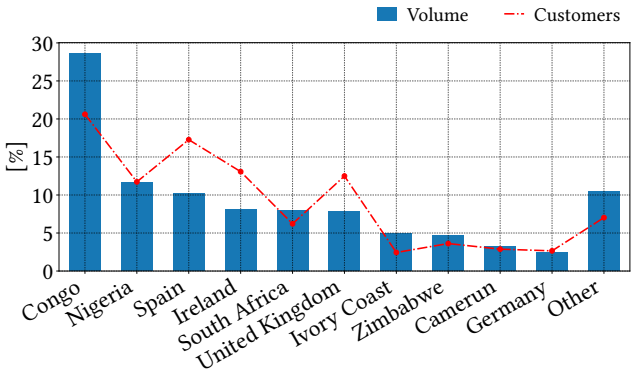


Fig. 5.2 Per country breakdown of traffic volume and user base.

distinguish between sub-services from the same provider (e.g., Google Search and Maps both share the SNI `*.google.com`). As an alternative to manually curated lists, we could rely on online ranking and analytics tools (e.g., Alexa, Cisco Umbrella, or Similar Web). However, these services are known to list only the main domain of a given service, while not listing the domains of third party services such as content-delivery providers and support services used by the first-party service. Given the small number of services we are interested in, we choose to create the lists manually.

The second step is to create aggregated views of the data to obtain traffic breakdowns by protocols, server domains, time (with 1hour granularity), country of the customer, and contacted service. This aggregation step facilitates subsequent data processing by reducing the amount of data to be processed by several orders of magnitude, enabling real-time data exploration.

5.4.2 Dataset Overview

We first present an initial overview of the dataset by presenting the breakdown by country and protocol. Table 5.1 summarizes the latter. As expected, web traffic accounts for most of the traffic, with HTTPS and QUIC accounting for 56 % and 19.6 % of the total volume, respectively. HTTP still accounts for 12.1 % share. This is consistent with other studies that showed the convergence of Internet protocols towards encrypted web protocols [64, 97, 79]. Interestingly, despite the high latency due to the satellite link, we observe a non-negligible amount of video or voice traffic using Real Time Protocol (RTP).

Looking at traffic by country, we observe a large imbalance in the number of customers and thus traffic, as shown in Figure 5.2. The blue bars indicate the share of data traffic per country. The red line shows the share of customers. The countries are sorted by decreasing total volume. Interestingly, the two figures are not completely proportional and show that customers in African countries consume much more traffic on average than customers in European countries. For example, Congolese customers are 20 % of overall customers, but they generate 27 % of volume (each generates about 600 MB per day). Spaniards are about 16 % of customers, but generate only 10 % of volume (each generates only 170 MB per day). This suggests that African customers may share Internet connections with multiple end-users, while European customers may resort to SatCom access only when forced to do so. In section 5.6 we will explore this direction in more detail.

To complete this overview, we list the breakdown of protocols by country. We limit the analysis to the top-10 countries. Figure 5.3 shows the results that exhibit some considerable differences. In European countries, a large portion of TCP traffic is not due to web protocols. The case of Germany is extreme: 35 % of all TCP traffic is due to other protocols. Manual inspection suggests that this is due to the use of Virtual Private Network (VPN) solutions (unknown protocol, non-standard port, long-lived flows without parallelism). Note also that the percentage of unencrypted HTTP traffic is higher in Ireland and the U.K. than in other countries. This is due to popular Microsoft and Sky services that use HTTP to distribute software updates and video content. Conversely, Congo, Nigeria, and South Africa show very similar protocol breakdown. This may reflect the different customer base in Europe and Africa, with some business customers in the former (confirmed by the SatCom operator) using non-web-based protocols for VPN and internal services.

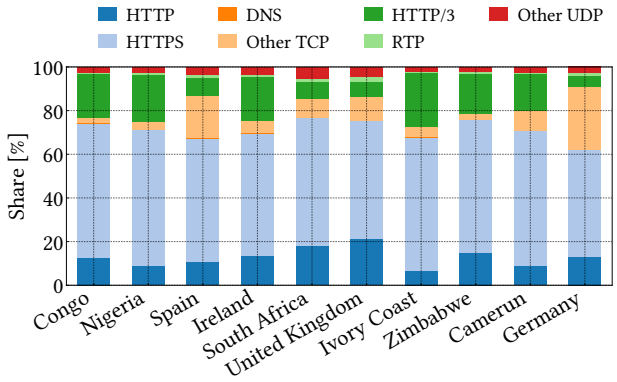


Fig. 5.3 Protocol share per country.

Given this initial overview, we limit our analysis below to the top 3 countries in Europe and the top 3 countries in Africa to compare usage, performance and services used.

5.5 How much Customers Consume

In this section, we discuss the temporal pattern and volume of traffic generated by customers in different countries. We start our analysis from the traditional hourly traffic pattern, which we report in Figure 5.4. The y-axis reports the percentage of traffic volume at a specific hour, normalized over the maximum value for the given country. For each time bin, we report the average value seen at that time during the whole time period, summing the upload and download traffic. We use the UTC time zone and countries in different time zones appear shifted.

We immediately observe how African (dashed lines) and European (solid lines) countries exhibit very different traffic patterns. In Europe, the traffic peak happens during evening prime time between 18:00-20:00-UTC. Conversely, during the day the traffic volume settles to lower values, down by 50 % in the morning and as low as 20 % at night. Conversely, in African countries, we observe a much higher traffic consumption during the morning too. For Congo (dashed red line), the absolute peak is at 9:00-UTC (10:00 local time). In Nigeria and South Africa, the morning peak

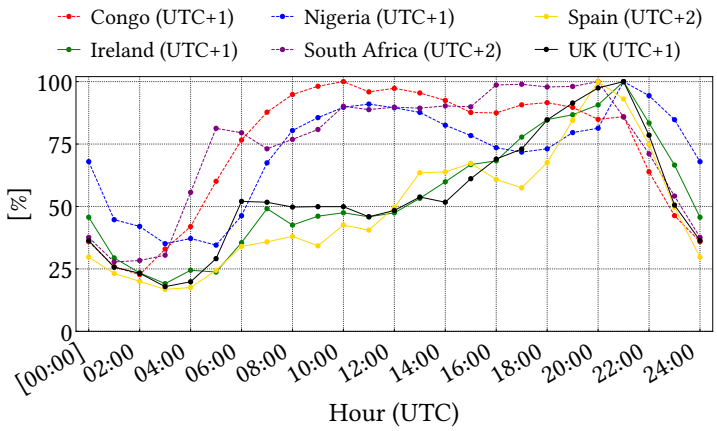
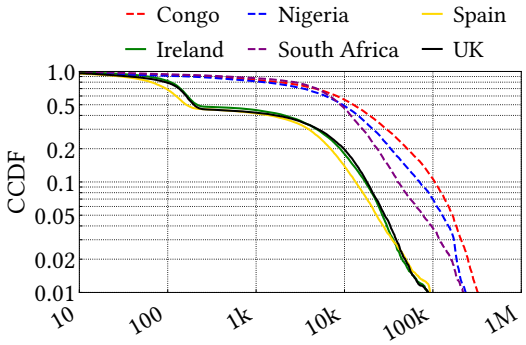


Fig. 5.4 Daily trends per country.

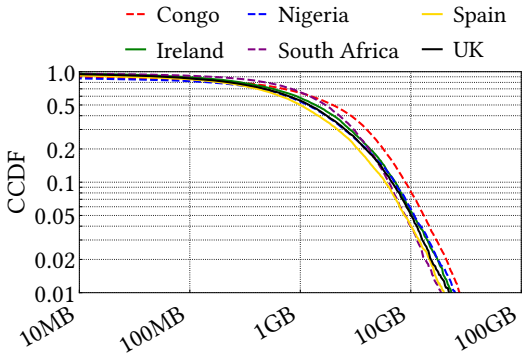
reaches 90 % of evening peak time. Notice also at night the low-peak that is almost as high as 40 % of peak-time. This suggests that customers' use of the SatCom access differs between Europe and Africa, hinting a classic leisure usage for the former.

We now focus on the volume of traffic generated per each customer, per each day. In Figure 5.5, we report the empirical Complementary Cumulative Distribution Function (CCDF), using log-log scales. We consider both the total number of flows and the total bytewise volume generated in each day by a given customer. We consider both TCP and UDP, and distinguish upload and download volume. Note that a single subscription generates many samples in the distribution, one per day.

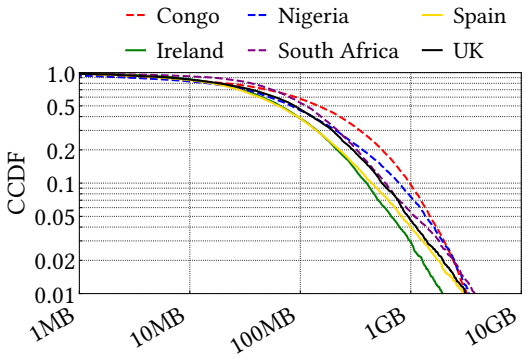
Start from the number of daily flows per customer shown in Figure 5.5a, and focus on the main body of the CCDF (the leftmost part of the figure). We clearly observe that a significant fraction of European subscribers (solid lines) generate less than some hundred flows in 24 hours. In fact, the curves show a clear knee between 50 and 250 flows – i.e., more than 50 % of customers generate less than 250 flows in a given day. Those flows are likely being generated by the SatCom equipment or devices left connected to the network but unused. This could be the case of customers buying satellite access for their second houses in the remote regions that they use only during holidays. Conversely, in African countries (dashed lines) this effect is not present. From now on, we define *active customers* those that generate at least 250 flows per day.



(a) Flows (all customers)



(b) Download volume (active customers)



(c) Upload volume (active customers)

Fig. 5.5 Distribution of daily volume per customer in different countries. Notice the log scale on both axes.

Focus now on the tail of the curve. Here we can clearly see that African customers generate almost an order of magnitude more flows per day than European customers. This is due to the presence of some WiFi access points that share SatCom access in community internet solutions or internet cafés. In fact, most Africans in Congo and more in Nigeria and South Africa than in Europe, lack home Internet access. Therefore, people have to go to public places – Internet cafes, libraries, workplaces, etc. – to access the Internet. This situation was observed by the scientific community as early as 2013 [98] and more recently in 2019 [99]. The multiplexing of several end-users behind a single customer CPE IP address results in an inflated number of per-customer daily flows.

Next, we characterize the amount of daily downloaded and uploaded amount of traffic by each active customer. We show results in Figure 5.5b and 5.5c, respectively, for downlink and uplink. First, we observe that African customers download more data than European customers. Yet, the increase is reduced. For instance, in Congo, the percentage of heavy hitters (those customers downloading more than 10 GB in a day) is twice as much (8 %) than in Spain (4 %). This different behavior clearly impacts the congestion on the SatCom link, and the per-customer *cost* the SatCom operator in terms of volume.

Interestingly, the difference between European and African countries is more pronounced in upload volume than in download volume. A Look at Figure 5.5c shows that Congo, Nigeria and South Africa have 10 %, 7 % and 5 % heavy hitters (those customers uploading more than 1 GB of data in a day), respectively, compared to less than $3-4$ % in the U.K., Spain and Ireland. As we will see in the next session, customers who tend to upload a lot of content tend to exhibit a large usage of instant messaging applications, likely sharing images and videos from their mobile app. Overall, we find that SatCom customers generate similar traffic volumes as FTTH and ADSL customers. Compared to some recent work [64], the average download (upload) volume per FTTH customer was on the order of 1 GB (100 MB) per day in 2017. We thus observe a significant increase in the volume of traffic exchanged by customers despite the limited possibilities offered by SatCom access.

	Congo	Nigeria	South Africa	Ireland	Spain	UK	Average
Google	62.96	61.26	64.72	68.58	68.30	65.48	64.20
Whatsapp	61.22	51.18	62.88	59.59	63.82	53.75	58.62
Snapchat	33.93	28.90	19.14	38.52	12.33	28.50	28.10
Wechat	6.42	3.55	1.11	0.49	0.06	0.41	2.99
Telegram	1.83	3.17	1.28	0.53	1.75	0.29	1.64
Instagram	48.81	41.04	40.67	48.53	45.59	40.43	44.84
Tiktok	41.56	31.99	36.31	40.11	31.89	36.53	36.95
Netflix	17.34	17.84	38.91	50.91	39.20	46.41	30.21
Primevideo	3.90	3.77	8.42	21.30	22.78	28.21	11.94
Sky	15.71	7.86	7.26	27.68	6.04	28.37	14.87
Spotify	37.78	30.31	33.19	46.79	45.20	39.73	38.15
Dropbox	11.50	9.22	16.57	10.39	9.34	16.81	11.75

Fig. 5.6 Heatmap of the service popularity in different countries.

5.6 What Customers Consume

We now examine the habits of SatCom subscribers in terms of services they access. As described in Section 5.3, we identify the services by examining the domain of TLS, HTTP, and QUIC flows. For each country, we extract the list of popular services, manually create regular expressions to identify them from the domain, and assign a category among Audio steaming, Chat, Search engine, Social, Video streaming, and Work.

In Figure 5.6, we first give, for each category, the percentage of customers accessing different services on a daily basis, separated by country. We focus on a subset of the services for which we can write regular expressions that match domains that we know the user intentionally visited. For example, we do not report on Social Media Networks or widely used services (e.g., YouTube) because these services often appear as third-party services in web pages (e.g., embedded social buttons, videos, tracking services).

Among Chat services, WhatsApp is the most widely used service - comparable to Google, which is the most popular service, as expected. This is consistent with our earlier findings [64]. Interestingly, Snapchat ranks second, while in Congo more than 6 % of customers also use WeChat for communication. This suggests the presence of Chinese-related communities. Telegram has yet to gain momentum.

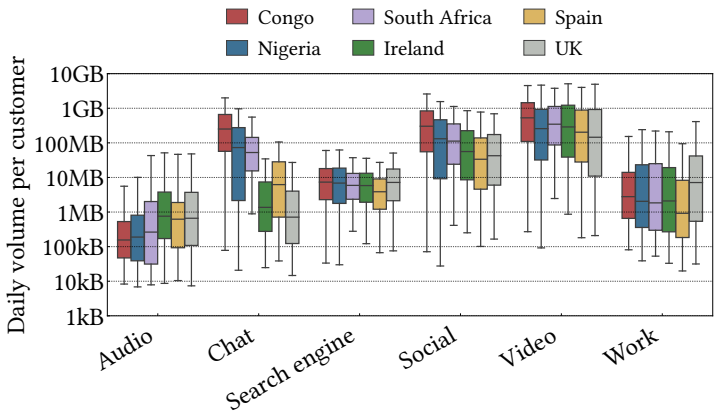


Fig. 5.7 Boxplot of the daily volume consumption per customer when accessing different service category.

Instagram and TikTok have similar penetration in all countries, with the latter being only $4-7$ percentage points less popular than Instagram.

In the paid-video category, it is notable that they are more popular in Europe than in Africa, with only South Africa achieving similar penetration. This is likely due to both economical and cultural effects, as well as the investments these platforms make in each country. For completeness, as previously said, Sky uses HTTP rather than HTTPS for serving the video content, and its popularity in Ireland and the U.K. leads to the increase in HTTP traffic observed in Figure 5.3.

Our findings corroborate recent research on African Internet usage, which shows the strong presence of social and chat services. For example, a recent report from the Pew Research center [100] shows that chat and social networking are much more popular than paid services in Africa. In fact, paid video streaming services are not yet very popular in Africa, with Netflix having an estimated 2 million users on the continent, according to the 2022 annual report from Digital TV consulting [101]. However, the same report predicts rapid growth, which is also confirmed by a report from Conviva, a major video distribution company [102]. We note that South Africa is peculiar among African countries and that the strong penetration of streaming in South Africa is already well known.

Next, we focus on the volume of traffic generated when accessing the different service categories per country. Here we consider all services, assuming that social buttons, tracking cookies, etc., consume little volume compared to customers using the actual service. Figure 5.7 uses boxplots to show the distribution of daily volume per customer accessing each category. The box extends from the lowest to the highest quartile, with a line at the median. Whiskers that extend from the box show the 5^{th} and 95^{th} percentiles.

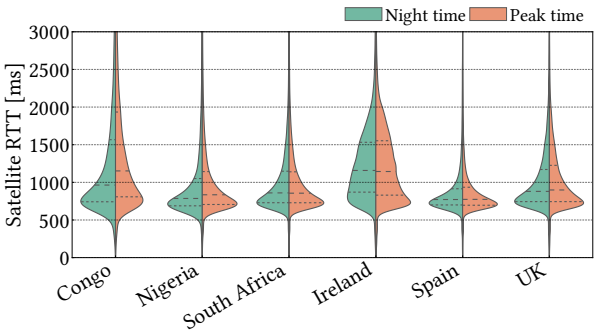
Audio streaming services consume the least amount of traffic in Africa and slightly more in Europe, where some customers consume more than 50 MB in some days. Chat application usage is much more heterogeneous. While customers in Europe consume a median of less than 10 MB per day for Chat services, this value surprisingly increases by more than three orders of magnitude in African countries. Customers in Congo have a daily median of 250 MB, with the top-5 % of the heaviest customers consuming more than 2 GB on some days. These are likely those community WiFi Access Point (AP) that share SatCom access with multiple end users. The same effect is observed in the Social Media category, with a daily median of 300 MB in Congo, but only 30 MB in European countries. In contrast, the differences are smaller in the Video streaming category. However, the share of video traffic comes from different services: free YouTube in Africa, and paid video streaming services in Europe.

5.7 Which Performance Consumers Get

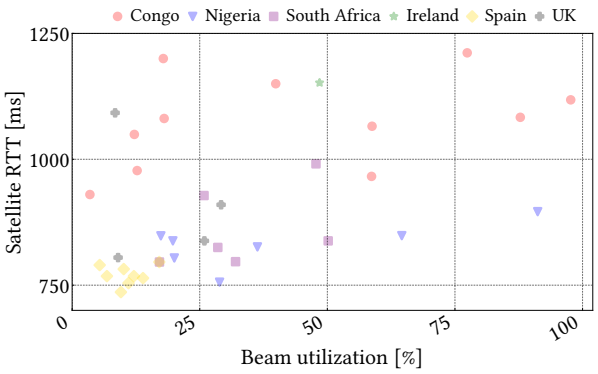
We now discuss the performance SatCom customers get. We focus on classical Quality of Service (QoS) indicators, namely RTT and throughput, and finally drill down on DNS performance.

5.7.1 Satellite RTT Analysis

Here, we examine how the satellite access link and SatCom network architecture affect end-to-end RTT. We consider the satellite RTT and the ground RTT separately, as we defined in Section 5.3.



(a) Distributions and quartiles of the satellite RTT per country for night time (from 2:00 to 5:00 local time) and peak time (from 13:00 to 20:00 local time).



(b) Median satellite RTT per beam.

Fig. 5.8 Satellite RTT computed from TLS handshake.

Focus first on the satellite RTT shown in Figure 5.8a. We report the measurements at night – when we expect low congestion on the satellite link – and at peak time. The dashed lines indicate the median, while dotted lines are used for the 25th and 75th percentiles. As expected, the minimum satellite RTT is above 550 ms. However, the distributions show very large variability with RTT that can be higher than 2 s and varies widely in each country. This variability is due to several factors: Queuing delay at various forwarding elements; Processing and transmission delay for limited performance terminals; Packet losses and TCP retransmission; but the main reason is the SatCom access technology. Specifically, Spain has the best RTT at night in general, with 82 % of RTT samples less than 1 s. Nigeria follows in second place and has even better RTT than Spain at low values. This is due to Nigeria favorable position, where the satellite is closer to the zenith (and thus has a shorter line of sight than Spain). South Africa and U.K. suffer from a larger zenith angle and thus a larger RTT.

In contrast, Congo and Ireland suffer from a much higher and more variable RTT. For Congo, the main cause is the congestion on the satellite beams covering the country. In such bandwidth-constrained scenarios, the MAC protocol and the PEP scheduler may delay the transmission of packets by several frames, affecting the satellite RTT. For example, note that about 20 % of RTT samples are longer than 2 s. The high RTT values already occur during periods of low peak traffic and worsen during periods of high traffic, with RTT values increasing significantly - compare median values. For completeness, note that congestion also affects some Nigerian beams, while it is practically unnoticeable in Spain, U.K. and South Africa.

For Ireland, on the other hand, the different shape of the CDF and the practically identical RTT during nighttime and peak hours rule out congestion as the main cause of RTT variability. In fact, Ireland is located at the edge of the satellite coverage area with a large zenith angle, so the satellite transmission channel suffers from severe transmission impairments. The Satellite data-link protocol must deal with such impairments, which affect access time for those customers in particularly unfavorable locations.

To give more details, Figure 5.8b shows the median satellite RTT for each beam and relates it to the beam utilization[7]. We consider the peak time interval and

[7]We normalize the results to the maximum utilization observed across all beams to avoid disclosing the actual per-beam utilization.

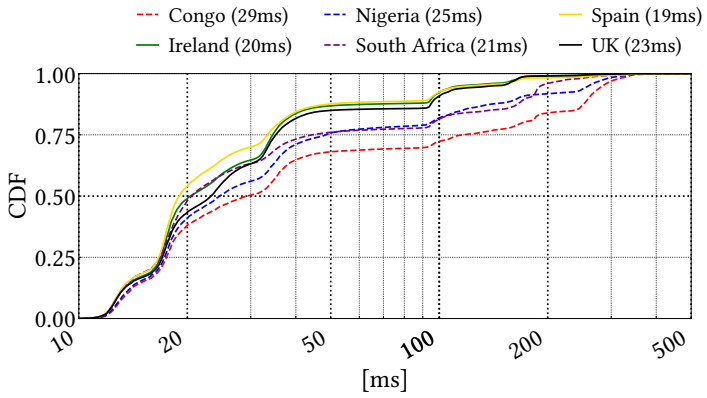


Fig. 5.9 Ground segment RTT computed as the average RTT in each TCP flow. Legend details the median.

mark each beam with the corresponding country it serves. Both Congo and Ireland suffer from high delay almost regardless of beam utilization. Nigeria, Spain and U.K. exhibit in general lower per-beam RTT. When checking this with the SatCom provider, the staff confirmed that there is some congestion that occurs, but it is not due to the beam capacity, but rather to the saturation of the PEP processing ability. This, in turn, slows down the forwarding of packets, especially during the initial phase of the connection setup. The amount of PEP resources that the SatCom provider allocates to each beam and country depends on the SLA and the cost of the service. This clearly shows the overall complexity of the SatCom access technology, with implications on end-user quality of experience.

5.7.2 Ground RTT Analysis

We now focus on the ground RTT shown in Figure 5.9. This RTT considers only the part of the path between the SatCom ground station and the server in the Internet.

In general, the ground RTT is much more deterministic than the satellite RTT. Here, clear bumps reflect the proximity of the servers on the Internet to the SatCom ground station. Focus on the European countries. The closest group has an RTT of about 12 ms and serves about 20 % of the traffic. These are CDN nodes of well-known players with widespread infrastructures with which the SatCom provider has direct peering agreements. A second group of servers is around $15-17$ ms, and a third around 35 ms. All of these servers are located in Europe and serve more than

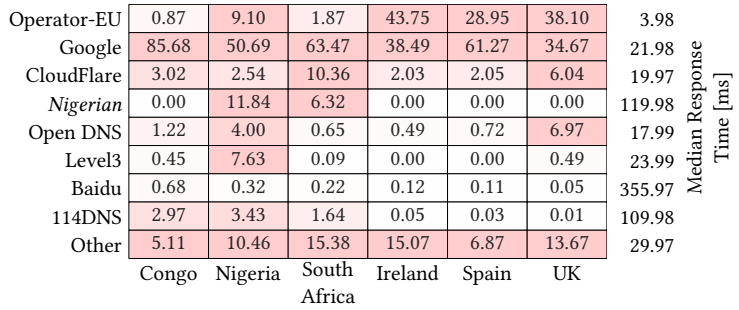


Fig. 5.10 Adoption and median response time of DNS resolvers.

80 % of the traffic for European customers. Continuing this analysis, we observe another group of servers at about 95 ms (180 ms). Most of these servers are cloud servers located in the East (West) coast of the U.S., which suffer from the latency of crossing the Atlantic Ocean (and the U.S.). This reflects the typical RTT on the common Internet paths [64–66].

Now look at the RTT for African countries. Surprisingly, they have a higher RTT than their European counterparts. Since all traffic must be routed through the same ground station in Europe, African countries experience additional ground RTT when the final server is located in the original African country, e.g., when the end-user accesses a local service that is not served by global CDNs. In other words, the location of the ground station in Italy forces all traffic to be routed through Italy. This creates the rightmost bumps, where RTT on the ground increases to $300-400$ ms.

By manually examining the services offered by these servers, we can confirm that they are likely popular services in the country of origin. Again, we also observe a significant proportion of Chinese services that are particularly popular in Congo (note the last bump in the ground RTT). This is related to the presence of Chinese companies in the country.

The SatCom provider is well aware of the RTT inflation due to the forced routing through the single ground station in Italy. They are already evaluating the possibility of setting up a ground station in Africa to optimize traffic routing and reduce groung RTT for those service located in Africa. In terms of performance, the numbers are clearly in favor of this decision.

	UK		Nigeria		
	Op-EU	**Google**	**Op-EU**	**Google**	**114DNS**
captive.apple.com	19.1 ms	26.0 ms	23.1 ms	38.4 ms	110.4 ms
play.googleapis.com	16.3 ms	17.7 ms	38.7 ms	36.0 ms	114.2 ms
*.nflxvideo.net	-	25.5 ms	33.6 ms	28.8 ms	20.1 ms

Table 5.2 Average ground segment RTT per country and DNS resolver.

5.7.3 DNS Performance

Given the importance of the server IP address location to reduce latency, DNS resolution plays an important role. For this, we drill down on DNS resolver choice and performance. We consider DNS/UDP traffic, for which we observe the original end-user device request and resolver response. First, we look for popular DNS resolvers, quantify their resolution latency, and next we observe the impact on server choice.

In total, we observe 4 195 of different resolvers, some of them only sporadically. Interestingly, we found that customers use well-know open resolvers instead of operator resolver, and strangely choose custom, unusual, and geographically distant resolvers. In Figure 5.10, we break down the top-8 resolvers in terms of volume, separated by country. For a given country, each column shows the percentage of DNS traffic for the different resolvers. In the rightmost column, we report the median response time observed at the ground station.

We note that the operator DNS (first row) is quite used only in European countries. In Ireland, Spain, and the U.K., it accounts for 44,%, 29,%, and 38 % of the DNS volume, respectively. With a mean resolution time of only 3.98 ms, it offers the best performance. As expected, Google DNS is popular everywhere. In Africa it resolves 86 % of requests in Congo and more than 50 % in the other African countries. Other popular open resolvers, namely CloudFlare DNS and OpenDNS, have a different popularity, usually below 10 %. The resolution time for all of them is on the order of 20 ms.

Nigeria is a peculiar case. We find that 12 % of traffic goes to a local Nigerian operator resolver. For this resolver, the ground RTT artificially inflates the resolution time to about 120 ms since packets have to travel from the ground station in Italy to Nigeria and back.

Interestingly, we observe two Chinese DNS resolvers (Baidu DNS and 114 DNS) in African countries, confirming the assumption that there is a significant Chinese community that use homeland resolvers. For Baidu, the resolution time is terrible, with a median response time higher than 350 ms that have to be added to the satellite RTT to reach the actual end-user device.

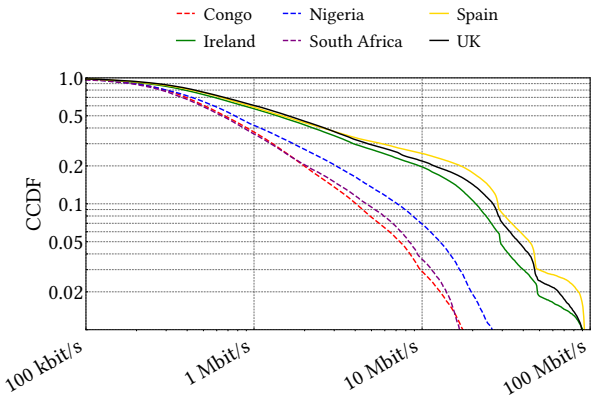
In summary, in most cases SatCom customers do not adopt the operator DNS and resort instead to open resolvers. Due to the particular routing in the SatCom network, we observe cases of resolvers that suffer very high RTT, yet they are widely used in Africa. This greatly impacts the DNS response time and introduces an additional $100 - 300$ ms delay on top of the satellite RTT. This has a clear negative impact on the user experience.

5.7.4 Implications on Server Selection Policies of CDNs and DNS Resolvers

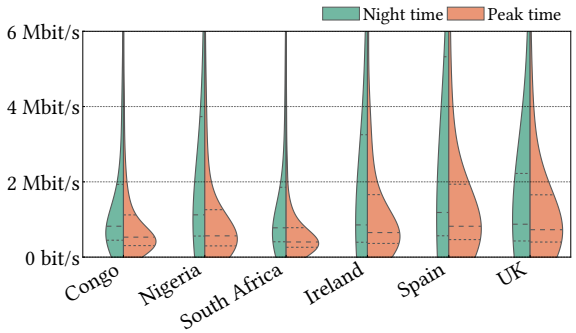
The superposition of i) routing constraints through the same ground station in Italy, ii) an intercontinental service presence that includes African and European countries, iii) the adoption of different DNS resolvers by customers, creates a very tangled picture that complicates the server selection policies of different CDN and DNS resolver operators. To examine these implications, we observe whether there are differences in ground RTT to the same service when using different DNS resolvers.

Table 5.2 shows some examples. We report the average ground RTT for some sample domains and some of the most popular resolvers for Nigeria and U.K. For U.K. (and European countries in general), the DNS resolver has little impact on performance. This is expected since i) the ground station is located in Europe and ii) customers tend to access European services that are well served by CDNs in Europe.

However, this is not the case for African countries such as Nigeria. For example, the server IP address resolved to serve the `captive.apple.com` service results at 19.1 ms if resolved via the Operator-EU DNS for U.K. customers. It results instead at 110.4 ms if resolved by the 114DNS for customer in Africa. Interestingly, even the Google DNS resolver returns two different CDN nodes for U.K. (26.0 ms) and Nigerian (38.4 ms) customers. These resolvers provide more distant CDN IP addresses because they are likely confused by the originating customer request geo-position which conflicts with the routing through Italy. Not shown for brevity, we



(a) Per country.



(b) Per time of the day.

Fig. 5.11 Download speeds per customer.

even observe that some DNS resolvers point to some CDN server in the original African country of the customer. This clearly inflates the ground RTT by several hundreds of milliseconds. Some domains, e.g., nflxvideo.net, are less affected by these phenomena. This may be because resolvers and CDNs have accurate information, or because they do not rely on DNS resolution to determine the closest CDN node, e.g., because they use Anycast-based CDN solutions (which are not affected by the DNS resolution issue).

A possible solution to the DNS inconsistency problem is to either force the use of the SatCom operator's resolver or work with the Open Resolver providers to correctly instruct the server selection policies to return the closest server to the ground station instead of the original location of the end user's terminals.

5.7.5 Throughput Analysis

For the sake of completeness, we now briefly discuss download throughput. We measure it for TCP connections by calculating the gross ratio between bytes downloaded and the duration of the flow (calculated from the first to the last TCP segment with data sent).

Recall that we observe the TCP data flow from the Internet server to the ground station PEP. This download throughput is regulated by the actual download throughput from PEP to the end device, which happens to be the bottleneck. To obtain a reliable measurement, we only consider flows large enough for the throughput to reach stable values and to neglect the effects of buffering at the PEP.[8] For this purpose, we only consider flows carrying at least 10 MB of data, for which we limit the representations to 1 million samples from a three-day interval. Even in these cases, not all flows can be considered valid bulk download samples (e.g., persistent HTTP flows or rate-limited video streaming flows), and competing traffic may limit throughput. This figure can only be considered a rough estimate of the actual performance a customer gets.

Figure 5.11a shows the CCDF for download throughput separately by country. The operator offers several commercial plans with different maximum throughput. This is reflected in the knees of the curves in the figure. In Europe, where 30 Mb/s, 50 Mbit/s, and 100 Mbit/s plans are popular, we find that these customers can saturate their capacity with a single TCP flow. Overall, European countries have similar download throughput, with Ireland achieving slightly lower values due to its particular physical channel characteristics (see Section 5.7.1). For brevity, we limited our analysis to video streaming flows, and separated off-peak and on-peak times. In both cases, we could not find any signs of congestion.

In the African countries, the picture is quite different. First, the operator sells plans with a capacity of 10 Mb/s and 30 Mbit/s. Only few customers can saturate this capacity, with Nigerian customers tending to achieve slightly higher throughput. This is likely due to lower congestion on the satellite link. However, the higher congestion on the link, the less optimal server selection and routing, the presence of community WiFi APs, and likely the less powerful end-user terminals limit the maximum download throughput that customers can achieve.

[8] The PEP has a limited buffer per-user.

This is confirmed by Figure 5.11b, which shows the distribution of download speed for each country, with nighttime hours separated from peak hours. Again, European customers have higher throughput than African customers. In all countries, throughput is lower during peak hours than at night, as shown in the body of the distribution and the lower percentiles and medians. The change is more pronounced in Congo and South Africa.

5.8 Takeaways

In this comprehensive study, we have performed the first characterization of the SatCom network through passive measurements at a major SatCom operator's ground station. This unique vantage point, aggregating traffic from tens of thousands of customers, enabled us to provide a multi-faceted perspective on this mature yet complex technology. The key findings of our study are synthesized as follows:

- **Usage Patterns**: Our analysis reveals distinct usage patterns across regions. European customers primarily use the internet during evening peak hours, indicative of leisure-oriented usage. In contrast, African countries exhibit a morning traffic peak, suggesting at least partial business usage. This regional divergence is further highlighted by the higher data flow and consumption in Africa, often through communal internet access points like internet cafes.

- **Service Preferences and Traffic Patterns**: SatCom customers in both regions favor chat services such as WhatsApp and social media platforms including Instagram and TikTok. However, African customers consume much more of these services, indicating multiple users per subscription, contrary to the typical sole or domestic use in Europe. Despite the routing through a single ground station in Europe, African customers frequently use DNS resolvers in other continents, adding latency but reflecting the global nature of internet connectivity.

- **Performance Metrics**: We observe significant variability in satellite RTT, with African countries experiencing higher RTTs due to factors like congestion, satellite positioning, and technological constraints. Ground RTT is lower for European countries, but higher for African countries, impacted by routing

through European ground stations. DNS performance is also critical, with many African customers using high-RTT resolvers.

- **Throughput and Access to Services**: Customers are able to reach the nominal throughput of their plans, often accessing popular high-definition video streaming platforms. However, notable differences in download throughput exist between European and African users, influenced by service plans, congestion, and end-user terminal capabilities.

In conclusion, this study sheds light on the distinct usage habits and traffic patterns of SatCom customers, emphasizing the necessity for targeted optimization strategies in SatCom services. This involves considering establishing local ground stations in Africa, enhancing coordination with DNS resolver providers, and optimizing CDN server selection policies. Such strategies are crucial for improving the quality of experience for SatCom customers globally, taking into account the regional variations in service usage, traffic volume, and performance metrics.

Chapter 6

Retina: An open-source tool for features extraction

6.1 Motivation

The work presented in this chapter is mostly taken from our paper *Retina: An open-source tool for flexible analysis of RTC traffic*, published in *Computer Networks 202* [103].

Retina is a user-friendly command-line utility designed to extract advanced network metrics for RTC sessions found in packet captures. Additionally, it generates visual output comprising various charts and visual representations of the metrics, facilitating easy analysis. The primary focus of *Retina* is on the Real-Time Protocol (RTP) [3], which is the prevalent protocol used in most RTC applications [104]. This includes its encrypted version SRTP (where packet headers remain unencrypted). *Retina* delves deeper into the understanding of RTC traffic compared to more generic tools. Starting from a capture, it scans for RTC traffic, identifies streams, and provides over 130 statistics pertaining to packet characteristics like timing and size. It also traces the evolution of the stream over time intervals of a chosen duration. The utility is highly adaptable, allowing users to customize the output metrics and various other parameters. Moreover, *Retina* can enhance its output by incorporating information from RTC application logs, offering the necessary ground truth for numerous classification tasks.

Retina is open-source and accessible to the research community and network practitioners.[1] We believe it holds value for traffic monitoring, and it has been effectively employed for data processing and feature extraction to support Machine Learning (ML) algorithms in the realm of RTC-aware network management.

6.2 Related Work

Numerous tools are already proficient in conducting thorough traffic analysis, with packet dissectors like *Wireshark*[2] (along with its command-line variant *Tshark*) serving as primary resources for network diagnostics. Flow monitoring is also commonly employed for examining traffic summaries [105], with NetFlow [106] being the widely accepted standard for collecting and processing flow records. Advanced network monitors also furnish application-level statistics through Deep-Packet Inspection on Layer-7 protocols. For instance, Tstat [38] provides comprehensive global statistics on RTP streams, while nProbe [107] introduces a VoIP plugin as a closed-source commercial offering. In contrast, *Retina* supplies extensive statistics on both a per-time-unit and per-flow basis. Its specialization lies in RTC traffic, encompassing the detection of numerous RTC applications, including those that modify the RTP protocol. Moreover, it provides a broad array of parameters for tailoring log creation to individual preferences.

6.3 System Overview

In this section, we describe *Retina*'s operation. As input, *Retina* takes one or more packet captures as well as optional configuration parameters. It processes the traffic and outputs the desired output in various forms. Figure 6.1 depicts its overall architecture. *Retina* is written in Python and depends on *Tshark* and a number of modules that can be installed via the package manager `pip`. We also provide a dockerized version to allow the use as a standalone container.[3]

[1] https://github.com/GianlucaPoliTo/Retina
[2] https://www.wireshark.org/
[3] The dockerized version is available at: https://hub.docker.com/r/gianlucapolito/retina

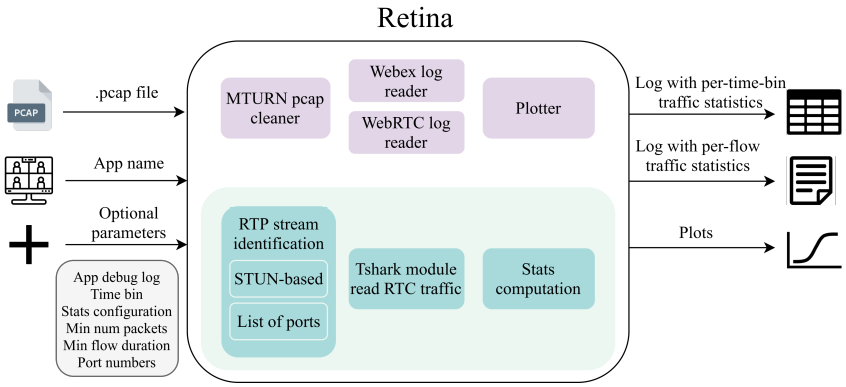


Fig. 6.1 *Retina* architecture.

6.3.1 Inputs and Configuration

Retina requires the user to specify one or more captures in PCAP format, the most common format used in many traffic capture software (*Wireshark*, *TCPdump*, etc.). *Retina* can also process an entire directory by searching for all captures in it. If it finds more than one, *Retina* uses multiprocessing to process multiple files at once. The number of processes is a configurable parameter.

For some RTC applications, the user can provide application log files that *Retina* uses to calculate additional statistics and enrich the output. The application logs typically contain details about the media sessions, including the Source Identifiers of the RTP streams, the type of media (audio, video, or screen sharing), the video resolution, the number of frames per second, etc. When available, *Retina* uses this additional information to provide finer-grained per-second statistics – e.g., media type, video resolution or concealment events at the codec level. Currently, *Retina* supports log files of: (i) Cisco Webex[4], which logs second-by-second details for each RTP stream, and (ii) Google Chrome, by collecting WebRTC debugging logs with WebRTC browser-based RTC services.[5]. This way we can download logs of each application used through Google Chrome (Google Meet, Jitsi etc.).

In *Retina*, the user can customize a variety of parameters. All are optional, with carefully set default values. *Retina* has personalized features for many RTC applications, which can be enabled by specifying the name of the RTC application

[4] https://www.webex.com/

[5] These logs can be obtained by creating and downloading a dump at `chrome://webrtc-internals`

whose traffic is included in the capture as an input parameter. While it supports all applications that use RTP at their core, we have tested it extensively for Webex, Jitsi, Zoom, and Microsoft Teams. *Retina* accepts threshold parameters, such as the minimum number of packets or the minimum duration of a stream for it to be considered valid. The user can also control the statistics computed at each time bin (see Section 6.3.3) and can ask *Retina* to create (interactive) graphs. The full list of parameters can be found in the documentation, while in the rest of the chapter we will only mention the most important ones.

6.3.2 System Core

The overall architecture of *Retina* is shown in Figure 6.1, with the middle rectangle indicating the building blocks at its core. We depict the basic functionalities in blue, at the bottom, and the optional modules in purple, at the top. We also show a sample command line at the top of Table 6.1.

The basic functionalities of *Retina* analyze the raw packets contained in the input PCAP captures and gather statistics, organized in tables per stream and per time-bin. For example, consider a PCAP capture collected at a user side, containing RTP traffic from a two-party call consisting of 4 RTP streams (outgoing and incoming audio and video). Setting a time bin duration of 1 second, *Retina* maintains a table where, for each of the 4 streams and for each second, it accumulates several statistics. Given a packet characteristic, such as packet size or interarrival time, *Retina* calculates several statistical indicators, such as mean, median, third and fourth moments, or percentiles. We report the list of packet features and available statistics in Figure 6.2, which summarizes the whole process of statistics extraction. The user can configure the duration of the time bin for this aggregation of packets, which is 1 s by default. The duration of the time bin directly affects the number of packets used to compute the statistics, and should therefore be varied judiciously. For example, in 1 s of audio, 50 packets are sent, while, in 1 s of HD video, more than 200. Clearly, if the time window is 200 ms for audio, no significant features can be computed, while this time window would be fine for video.

To identify RTP streams in traffic, *Retina* internally relies on *Tshark*, the command-line version of *Wireshark*. This step is not straightforward, as RTP packets often appear in a UDP flow along with other protocols. In fact, many applications

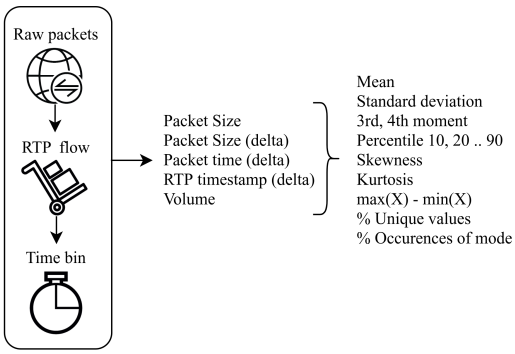


Fig. 6.2 Aggregation process and some of the statistics computed by *Retina*.

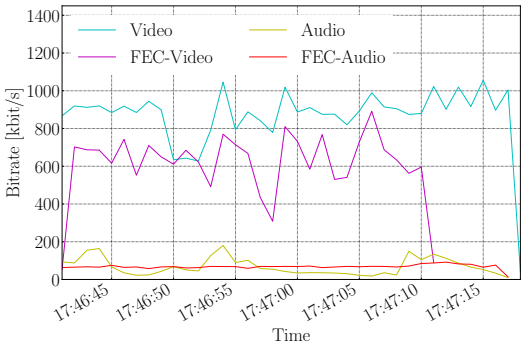


Fig. 6.3 Example plot of the stream bitrate in a call.

use STUN [4] to establish the media session and/or TURN [5] to relay the streams if no direct connection between peers is possible. In addition, it is common to use DTLS [108] interleaved among RTP packets to exchange control information such as encryption keys. *Retina* supports two methods for identifying RTP streams: (i) with a user-defined list of ports or (ii) by examining the STUN-initiated UDP flows. *Retina* attempts to decode the UDP payload as RTP and verifies that the protocol headers are compatible with RTP. We define an RTP stream using the combination of IP addresses and ports (the classic *tuple*) plus the RTP Synchronization Source Identifier (SSRC), which is used to multiplex multiple streams within a single UDP flow. For some RTC applications, we also use the RTP Payload Type (an RTP field that specifies the media codec). *Retina* maintains internal data structures to efficiently collect statistics for each RTP stream. This way to identify RTP streams in a traffic mix is used throughout the thesis.

Retina has a number of optional modules that target RTC applications, for which we have implemented special support. First, the traffic of some popular

RTC applications (Zoom and Microsoft Teams) needs to be preprocessed to become standard RTP traffic. This is because they use the RTP protocol in a non-standard form[6]. Microsoft Teams encapsulates RTP in a proprietary version of TURN called MTURN, while Zoom adds its own undocumented header. To make *Retina* work for these RTC applications, we have created specific modules that can also be used as standalone command line tools. They can be found in a separate folder in the code repository.

Second, *Retina* can read and process the application log of (i) Webex and (ii) Google Chrome, as mentioned in Section 6.3.1. *Retina* can parse these logs and provide additional information about the RTP flows. If the application logs are available, we enrich the output logs from *Retina* with information such as the video resolution, employed codec, frames per second, jitter, codec concealment events, etc. We also provide a classification of media types into 7 classes, such as audio, FEC streams, 3 different qualities of video and screen sharing, for easier recognition. The information in the application logs is particularly useful for training supervised ML models, as it contains the necessary ground truth for many QoE-related problems, such as number of losses, smoothness of the video, concealment etc. *Retina* matches the timings of the logs with the timings of the packets exactly, so it outputs a labelled dataset.

Lastly, *Retina* includes a plotting engine based on the Matplotlib and Plotly libraries[7] to create both static and responsive graphs of all RTP streams. It draws the time-series of stream characteristics, such as bitrate or inter-arrival time, so that the user can easily get an overview of the traffic or debug an RTC application. It also draws several histograms for each stream to show the stream-wise distribution of packet characteristics (e.g. packet size). For an example graph, see Figure 6.3. Here we show the bitrate of 4 RTP streams present in a portion of a Webex call. The plotting engine also labels the time-series with their media type (audio, video, FEC etc.), if the information is provided (e.g. through an application log file).

[6]Based on recent tests conducted on Microsoft Teams version 23320.3110.2622.1325, it appears that there are no longer any masking mechanisms concealing SRTP traffic. When decoded in Wireshark as RTP, the headers are found to be in plaintext.

[7]Matplotlib: https://matplotlib.org/, Plotly: https://plotly.com/

Command: `./Retina.py -d capture.pcap -so webex -log webex.log`

Timestamp	Packet size (mean)	Packet size (std dev)	Bitrate (kbit/s)	Interarrival (max)	Packets/s	Frame width	Frame height	Frames/s
2021-06-08 14:32:11	1041.84	66.74	1163.93	0.043	143	480	270	30
2021-06-08 14:32:12	1080.72	100.75	1578.86	0.045	187	640	360	30
2021-06-08 14:32:13	1023.49	72.21	1023.49	0.045	128	640	360	30
2021-06-08 14:32:14	1076.80	52.91	1362.82	0.043	162	640	360	30
2021-06-08 14:32:15	1055.50	52.41	1410.08	0.044	171	640	360	30
2021-06-08 14:32:16	1074.62	62.71	1989.73	0.089	237	640	360	30
2021-06-08 14:32:17	1055.22	40.09	2588.59	0.033	314	640	360	30
2021-06-08 14:32:18	1057.73	51.67	1479.17	0.040	179	640	360	30

Table 6.1 Example command and *Retina* log for an RTC stream. The last three columns are derived from the application logs.

6.3.3 Outputs

Retina produces a CSV file for each RTP stream found in the input capture, reporting the selected statistical features for each time bin. The logs contain different columns according to user preferences and additional stream information if the RTC application log is provided. We show an example output log in Table 6.1, along with the command line used to create it. Optionally, *Retina* creates a summary log file in which it reports stream-wise statistics. The file contains the most important information for each stream – i.e., the source and destination IP addresses and ports as well as general statistics such as the number of packets, duration, etc. Having per-stream information is useful for many applications that rely the analysis of flow/stream records for e.g., traffic accounting. Additionally, *Retina* provides a rich set of graphs that describe the traffic, which are discussed in Section 6.3.2.

Finally, *Retina* also provides a dashboard for analyzing RTC traffic through an interactive interface.[8] The dashboard requires an input `.pickle` file, which can be produced by passing one or more packet captures to *Retina* and specifying an argument for the plot. Here the user can see interactive plots of stream statistics and compare streams of interest.

[8]An online demonstrator of the dashboard is available at: https://share.streamlit.io/gianlucapolito/retina-dashboard/main/dashboard.py

6.4 System Design Assets

Retina is designed following principles of scalability and modularity, so that it can be easily extended. It adopts a multiprocessing architecture, so when there are multiple PCAP files to process, it uses an independent process for each of them and stores separate output log files. These files can then be merged at the end of the processing. This also increases the robustness of the tool.

Retina is highly modular, with separate functions organized into logical modules for all the different operations. This also allows for extensibility, as a user can write new functionalities with minimal effort. For example, it is easy to support the application log of a new RTC application (e.g. Microsoft Teams), as it is only necessary to add a parser function and call it with an argument.

Retina can be used to analyze any kind of RTP traffic, and it is not limited to video conference applications. For example, we have successfully used *Retina* to gain insights into the operation of cloud gaming applications running over the browser [109]. Similarly, our parser for the Chrome WebRTC log works seamlessly for any type of browser-based application.

Finally, *Retina*, as described in Section 6.3.1, is highly configurable. The user can limit the statistics to be computed (potentially speeding up the computation), the desired time aggregation, and several internal parameters - e.g., the minimum length of an RTP stream for it to be considered - which are detailed in the README file.

6.5 Publications Enabled by the Software

Retina was first developed at the end of 2019, and within 4 years of its existence, it has already been a valuable asset for 6 scientific publications that target RTC traffic. *Retina* sits at the core of [110] and [111], described in Chapter 7. There, we used it to engineer features and extract the ground truth for an ML classifier that distinguishes media types. Using these features, we developed a Decision Tree classifier that performed with 97% accuracy. We further built on it in [112], to do data preprocessing and identify RTC streams in traffic. It also served for data characterization in [104], where we compare 13 different RTC applications. We also successfully employed it to study cloud gaming traffic, and it allowed us to

understand the networking operation behind Google Stadia, GeForce NOW and PSNow in [109] presented in Chapter 3.

6.6 Takeaways

This chapter has introduced *Retina*, a comprehensive command-line utility designed to extract advanced metrics from network traffic of Real-Time Communication (RTC) applications. It provides a systematic overview of its functionalities, including the inputs, core processing system, and outputs, complemented by practical examples. Moreover, the chapter highlights the robust design principles of *Retina*, such as its modularity, scalability, and adaptability, making it a versatile tool in the field.

The significant contributions of this chapter are as follows:

- It outlines the integral role of *Retina* within this thesis, establishing it as the primary tool for processing RTC traffic as per the traffic classification system detailed in Chapter 7. *Retina* is positioned as the foundational step for all RTC packet analysis.
- The utility serves as a valuable asset for both the academic and professional spheres, facilitating research into RTC applications and assisting network administrators in resolving RTC traffic-related challenges.
- *Retina* acts as a pivotal framework for feature generation and as a reliable source for ground truth data, essential for machine learning endeavors aimed at improving RTC traffic visibility and Quality of Experience (QoE).

Given the current scarcity of modern, user-friendly tools and datasets for the efficient development of algorithms and the establishment of benchmarks, we believe that *Retina* can significantly contribute to this area. By making this software publicly available, we aim to partially close this gap, providing the networking community with a resource to enhance their investigative and developmental capabilities.

Chapter 7

Machine Learning for QoE in Real-Time Communication

7.1 Motivation

The work we present in this chapter is mostly taken from our paper *Real-time classification of real-time communications*, published in *IEEE Transactions on Network and Service Management 2022* [111].

In recent years, real-time communication (RTC) applications for video calls and virtual meetings have become a fundamental pillar of leisure and business. They help people communicate with each other and businesses save significant travel costs. Their value was especially proven during the months of self-isolation due to the COVID -19 pandemic, when online conferencing allowed many businesses to continue operations using remote working, mitigating the economic impact of the outbreak. This was largely possible due to the Internet being ubiquitous and the available bandwidth increasing [113]. After the first phase of IP telephony based on SIP and H.323, during the early 2000s, Skype opened the business for RTC applications, entering into competition with traditional telephony providers. At that time, most users were connected via cable modems, which offered low bandwidth and high latency. Today, the market offers countless competing video calling applications that benefit from the widespread adoption of broadband access and cellular networks. Each application employs different technical solutions and network protocols, al-

though Real-Time Protocol (RTP) [3] is most widely adopted [104]. There are also efforts towards standardization, WebRTC being the notable example.[1]

In this context, it is essential to maximize the Quality of Experience (QoE) of users at the network level in order to avoid impairments, service misbehavior and consequently user churn. QoE depends on many factors, such as the quality of the participants' connection, network topology, and network management. Classification of RTC traffic is the first and most important step towards effective traffic management, allowing in-network devices to get an informed view of network flows and, if the classification is done in real time, to take appropriate actions to counter any degradation. The widespread adoption of encryption [114] has made it difficult for routers and middleboxes to separate traffic based purely on deep packet inspection (DPI) [115], while the convergence towards HTTPS has made port-based classification ineffective. However, most RTC applications adopt the RTP protocol [3] to encapsulate the multimedia content in its encrypted version, *Secure* RTP (SRTP). SRTP employs in-clear and straightforward packet headers, making its identification straightforward using existing DPI techniques. However, in SRTP, the media payload is encrypted, making it difficult to guess the type of content it carries. This calls for novel techniques based on machine learning (ML) to re-obtain visibility on application traffic and help decision-making at routers. For real-time communications, this could amount to distinguishing between top-priority flows and possibly less critical streams – e.g., audio as more important than video, the presenter's media as more valuable than the audience's media.

In this Chapter, we propose a novel ML-based application for classifying, in real-time, the RTP streams to the type of content they carry. Our approach is based on a few, but well-chosen features derived from the statistical properties of the traffic, which allow us to classify RTP streams using off-the-shelf supervised learning algorithms. Our approach identifies not only audio or video streams but also other properties of the media, such as the video quality or the use of Forward Error Correction (FEC) streams. Our solution works with minimal delay, deciding on the type of each stream within just 1 second of traffic. We design it as a software module that can be plugged into network devices (e.g., routers) or integrated into Software Defined Networks (SDN) to provide fine-grained traffic categorization and management.

[1] https://webrtc.org/

Our study is based on two popular RTC applications for online multi-party meetings with audio, video, and screen sharing: Cisco Webex Teams[2] as a business-oriented platform and Jitsi Meet[3] as a lightweight in-browser application. Using data coming from more than 62 hours of real calls, we evaluate the impact of feature selection and different classification algorithms. After careful feature selection and using a lightweight decision tree classifier, we achieve an overall accuracy of 96% for Webex Teams and 95% for Jitsi Meet, with no large differences across classes. Our models require little traffic to train and do not introduce systematic errors. We note that models trained for one RTC application (e.g., Webex Teams) are hard to transfer to another application (e.g., Jitsi Meet) due to the different feature distributions. However, in section 7.6.6 we show that we can partially overcome this limitation by using domain adaptation techniques. A few works already proposed their use for problems related to networking, albeit in different contexts. Authors of [116] use transfer learning in wireless networks for a caching procedure. Instead, the approach proposed in [117] used it in combination with Deep Reinforcement Learning to solve the reconfiguration problem in the context of experience-driven networking. It has also been used for QoE estimation of video streaming [118, 119]. The transfer learning technique we use (CORAL [120]) has already been used in optical networks for assisted quality of transmission estimation of an optical lightpath [121]. Here, we apply it to the RTC scenario, trying to align statistical features of network traffic.

7.2 Related Work

Network traffic classification has been extensively studied since the birth of the Internet [115]. Due to the widespread adoption of encryption and the use of proprietary protocols, traditional approaches based on mere DPI and port numbers fall short, and the current research tends to use statistical traffic features and machine learning techniques [122]. Recent efforts aim to identify the web services [123] or mobile applications [124] behind network traffic, predicting the QoE of web [125], video [126] or smartphone [127] users.

Focusing on RTC traffic, many works propose techniques to identify it among other traffic categories. The authors of [128] use a stochastic characterization of

[2] https://www.webex.com/team-collaboration.html
[3] https://meet.jit.si

Skype traffic to obtain an ML-based model to be used for classification. In [129], UDP flows are classified into different classes, including Skype and RTP-based traffic,using SVM models and statistical signatures of the payload. The approach proposed in [130] leverages statistical properties of RTP to differentiate between voice and data traffic. The authors of [131] propose a method to detect WebRTC sessions at run-time based on statistical pattern recognition. Finally, some approaches target signaling mechanisms of RTC applications to identify Skype traffic through in-clear headers exchanged during session setup [132]. The ultimate goal of RTC traffic classification is the improvement of QoS and users' QoE. These aspects have been studied, focusing on the relationship between QoS and QoE [133], targeting the WebRTC [134] and mobile [135] scenarios. Another way to improve the QoE of RTC traffic is optimal media bridge placement. The media bridge relays the traffic between the peers. Some works target cache placement in SDN, which can be adapted to the RTC scenario [136–139].

Fewer works address the classification of media streams carried by RTP streams. Authors of [140] train machine learning classifiers to distinguish, among other classes, video and audio flows, targeting the WeChat messaging application. The approach presented in [141] identifies 20 codecs used for compression of audio,based on packet size, RTP timestamp delta, payload type and ratio between RTP timestamp delta and payload size. However, they do not use machine learning but a simple lookup table. In [142], the authors use statistics on the packet size as a distinguishing feature between audio, video streaming, browser, and chat traffic. They use interesting features, albeit fewer than we do, and divide into broader traffic classes. We only target RTC traffic and divide it into 7 classes, while for them it is a single macro class. As a model, they opt for an interpretable decision tree, similar to ours.

The closest work to ours is the approach proposed by Choudhury *et al.* [143]. There, the authors design a system to classify RTP traffic to the employed codec. They develop an ML pipeline similar to ours, to classify audio traffic into three Variable Bit Rate (VBR) codecs, thus identifying three types of audio. Conversely, we distinguish seven classes, two of which are audio (audio and FEC audio), four are video (three video qualities and FEC video) and one is screen sharing. Similar to us, they classify RTP streams separately by time bin, with a granularity coarser than ours – 10-20 seconds vs 1 second. They use two types of features: statistical features of packet sizes (such as mean, standard deviation, mode, etc.) and entropy-based features (4 types of entropy calculations on the RTP payload of the packets). We

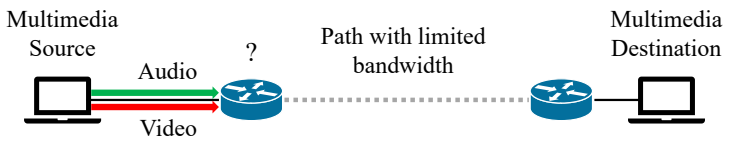


Fig. 7.1 Example of RTC-aware traffic management.

follow a similar approach, using five feature groups and calculating various statistics on the distributions. They also perform feature selection, reducing from 10 to 7 features. We use 8 for Webex and 4 for Jitsi.

Like in our system, they train offline, using 18-second streams and then the classifier is deployed in real time, over 10 seconds of stream data. They get overall 97% accuracy, similar to us (95%). Concerning the algorithms, they opt for a 1 Nearest Neighbours, while we finally choose a Decision Tree.

In summary, our work aims at unveiling the nature of media streams. Differently from previous works, we classify streams into a rich set of classes including media type (audio and video), video quality and redundant data (FEC). We engineer a wider range of features and then run an thorough feature selection process. Moreover, to the best of our knowledge, we are the first ones to explicitly target real-time applications with a 1 second (or shorter) classification delay, while the past approaches base their decision on the characteristics of an entire stream, lasting 10 seconds or longer.

The remainder of this Chapter is organized as follows. In Section 7.3 we motivate our work illustrating the advantages of RTC-aware traffic management. In Section 7.4, we present and characterize our dataset, while in Section 7.5 we describe our methodology for feature engineering and classification. Section 7.6 shows our experimental results, and, finally, Section 7.7 concludes the Chapter and discusses future work.

We make our dataset, code, and trained classifiers available online.[4] We believe they can help researchers reproduce our results or apply them to different contexts.

Table 7.1 Experiment summary

Experiment	Media type	Packets lost	Packets received	Packet loss %
1	Audio	2136	6673	24,2
RTC-unaware	Video	4997	17031	22,6
2	Audio	0	8809	0
RTC-aware	Video	-	-	-

7.3 Deployment Scenarios

In this section, we discuss the advantages of our proposal and possible deployment scenarios in real networks. We first illustrate how RTC-aware traffic management can practically improve user QoE, using a simple experimental setup. Our goal is to show that routing traffic not only based on the classical L3 packet headers, but also based on the media stream type, leads to sizable benefits under certain conditions. To this end, we setup a small testbed and assume our ML algorithm correctly classifies the media streams.

We outline our setup in Figure 7.1. Two hosts (the multimedia source and the destination) are each connected to a switch. The two switches are connected by a single path with limited capacity. The total available bandwidth for the path is 240 kbit/s. We set up a multimedia transmission where we send audio and video in two RTP streams. The audio is a high quality track with a bitrate of 140 kbit/s and the video has a constant bitrate of 200 kbit/s. The streams last 10 minutes and we send them simultaneously to emulate a video call. We build the testbed using the *Mininet*[5] tool to create the virtual network. We also use the Linux tool *tc netem*[6] to impose network constraints and *FFmpeg*[7] to stream the multimedia traces. To show the usefulness of our approach, we conduct two experiments: (i) *RTC-unaware*: the switch uses the classical approach to forward packets and thus treats both flows in the same way (ii) *RTC-aware*: our classifier is present and allows the switch to differentiate its behavior depending on the media type. Note that in both cases the required bitrate for both streams exceeds the capacity of the link.

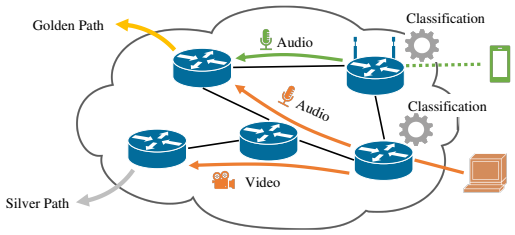
[4] https://smartdata.polito.it/rtc-classification/
[5] http://mininet.org/
[6] https://www.linux.org/docs/man8/tc-netem.html
[7] https://ffmpeg.org/

In both experiments, we quantify the impact on user QoE by using packet loss as a metric, since it has been shown to be closely related to QoE [144]. The results are summarized in Table 7.1. In the *RTC-unaware* experiment, the switch drops packets from both audio and video streams, so we observe 24% of losses for audio and 23% for video. This renders the communication impossible, as such packet loss prevents audio and video streams from being decoded correctly. In the second experiment, *RTC-aware*, the switch detects that the bandwidth on the path is insufficient for both streams and therefore decides to forward the entire audio stream and discard the video stream instead of sacrificing both. In this scenario, the audio stream reaches the destination without any losses, so that the interlocutors enjoy at least audio communication.

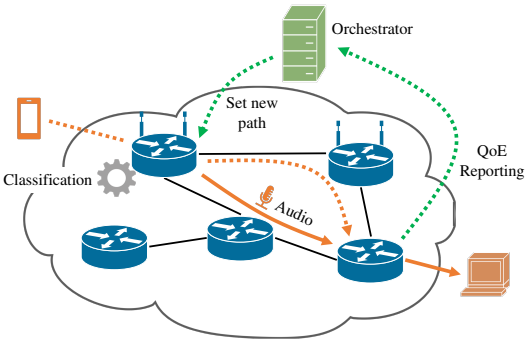
As this simple experiment exemplifies, our system can improve the QoE users perceive. Moreover, it enables RTC-aware traffic management so that scarce but expensive resources can be reserved for more valuable payloads (i.e., audio). Indeed, even in case we are unable to salvage the video, the network can preserve the audio. As proven in the literature [145, 146], when presented with a good audio and several different degraded versions of video, users perceive sufficient QoE.

On a general level, we foresee the use of our system in the context of RTC-aware network management and engineering, in which the network can make its decisions based on the type of multimedia content carried in streams. For example, the emerging SDN paradigm could benefit from RTC stream classification, allowing switches to steer traffic not only according to the classical protocol fields (i.e., addresses and ports) but also based on the media type of a stream. Similarly, in a Multiprotocol Label Switching (MPLS) network, the ingress node can set the label according to the classification outcomes. Also, approaches inspired to DiffServ or the IP *Type of Service* header are viable ways to differentiate traffic upon classification. Here, we do not explicitly target any of these possibilities but only show a few general cases where RTC traffic classification can be beneficial.

We sketch a first deployment scenario in Figure 7.2a. The edge network equipments run classification and select the path of each stream based on the media content they carry. In the example, audio packets are considered more critical and are routed to a *Golden* (reliable yet expensive) egress link, while the video is routed to a *Silver* (unreliable, yet cheap) path – e.g., a congested peering link.



(a) Media-Aware Path Selection



(b) Path Selection based on media type and QoE feedback.

Fig. 7.2 Deployment scenarios benefiting from our classification system.

We illustrate a second deployment possibility in Figure 7.2b. Here, our classification module is a building block of a more complex RTC-aware management system. Besides classifying streams to the content they carry (close to the source), network equipments report QoE-related metrics (close to the destination). The latter can be done in different ways, using well-known industrial standards [147, 148] or with other ML models [144]. A controller (or orchestrator) implements RTC-aware traffic management and can, like in our example, select new paths for RTC streams if it detects degradation in the measured QoE.

Note that the scenarios we envision are robust to possible flaws or delays in the underlying classification task. In Section 7.6, we report classification performance of 96.3% and 95.3% for Webex and Jitsi, respectively, with a delay of 1 second for collecting statistics and a few milliseconds for computing features and running the classification. Now we evaluate the impact on our proposed deployment scenarios, taking into account that both proposals (Figure 7.2a and Figure 7.2b) work by promoting streams to a better path when QoE is poor or the link is congested. There are generally two types of misclassification. In the first case, the error causes the system to respond unnecessarily – for example, we classify a video stream as an

audio stream and promote it to a more reliable path. In this case, the system wastes resources unnecessarily. In the second case, the error does not trigger a system response when it should have – e.g., we classify a stream that is actually an audio stream as video and do not promote it. In this case, the system would maintain the *status quo*, i.e., a "bad" QoE. In this sense, an accuracy of 95-96% means that the system improves the QoE in 95-96% of the cases, while in 4-5% of the cases we maintain the status quo or we waste some resources. Although undesirable, these situations do not entail severe impairment in QoE or in the whole system, provided they are sufficiently sporadic.

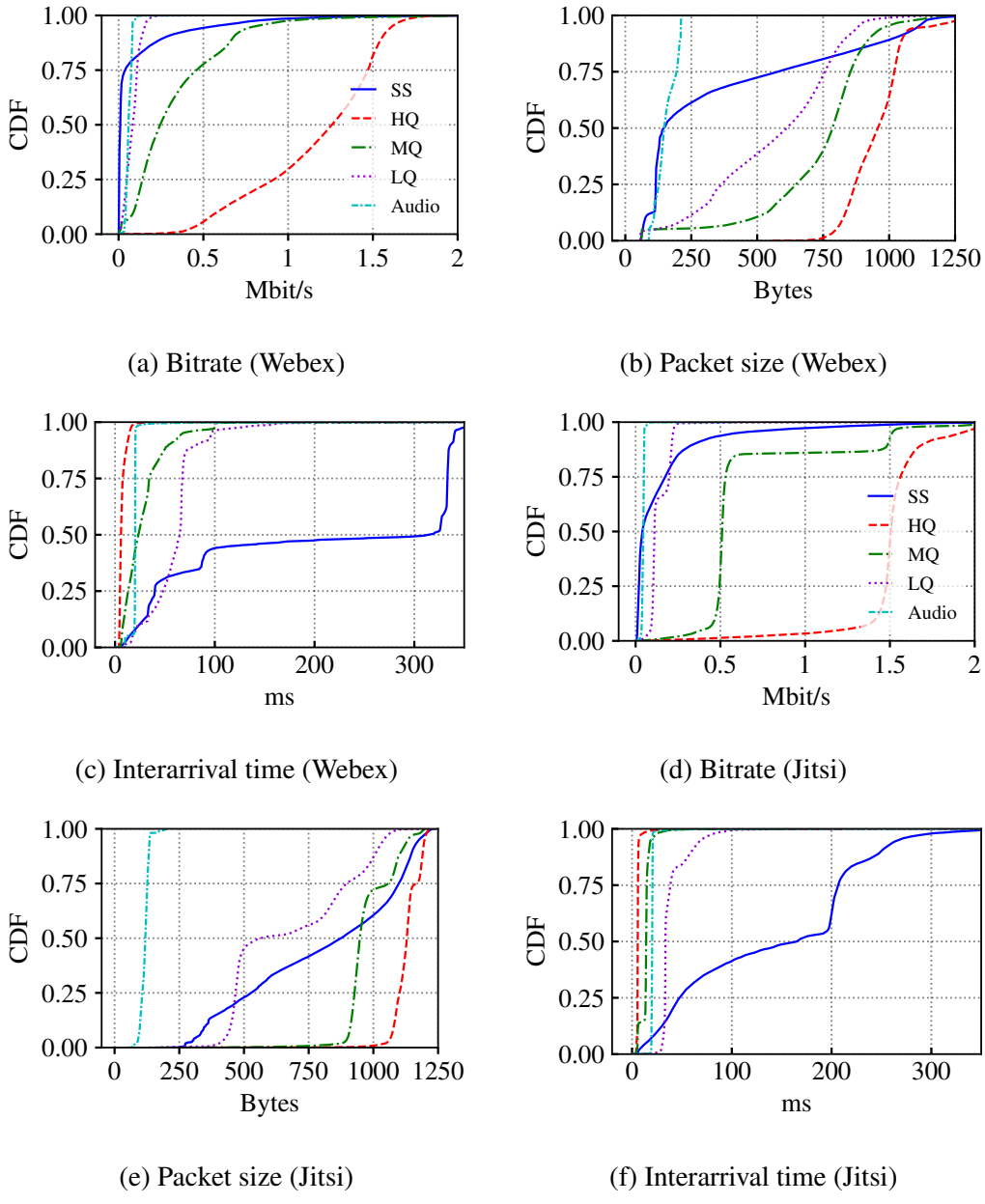
As for delay, we believe a delay in system reaction in the order of 1 s is tolerable for video calls, since their lifetime is in the order of minutes or hours. Collecting information about a stream for 1 second allows us to compute representative statistics about the stream, thus increasing the accuracy of the classifier. In Section 7.6, we also show that it is possible to use our classifier with slightly worse accuracy at a reduced delay of 200ms.

7.4 Dataset

7.4.1 Data Collection

In this section, we describe the dataset we use throughout this work. We target the two RTC applications described in the previous section, namely Webex and Jitsi. With both applications, we capture real calls made under different conditions, with a different number of participants (from two to ten), multimedia content (audio, video, screen sharing), and user equipment (PC, tablet, or phone). The calls run in a real environment where participants are connected via different networks from 3 countries and use different devices, from Windows PCs to iPhones and Android phones. During each call, at least one participant captures all the exchanged traffic and stores it in `pcap` format. The calls took place over a period of 6 months.

In our classification problem, we target RTP streams, which we identify with the tuple: source IP address, source port, destination IP address, destination port and RTP SSRC. In other words, we target a single stream that carries a specific multimedia content. We divide the streams into 5 classes: Audio, Low Quality (LQ) Video (180p), Medium Quality (MQ) Video (240-640p), High Quality (HQ) Video



(a) Bitrate (Webex)

(b) Packet size (Webex)

(c) Interarrival time (Webex)

(d) Bitrate (Jitsi)

(e) Packet size (Jitsi)

(f) Interarrival time (Jitsi)

Fig. 7.3 Distribution of traffic characteristics for Webex (top) and Jitsi (bottom), separately for media stream type.

(720p), and Screen Sharing. For Webex, we consider two additional classes: FEC audio and FEC video. Indeed, Webex uses FEC to mitigate packet losses, sending streams with redundant information to be used at the receiver if some packets are lost or contain errors. We observe FEC streams for audio and video, and we are interested

Table 7.2 Dataset summary

Class	No. of seconds			
	Webex		Jitsi	
	Train	Test	Train	Test
Audio	224 295	80 781	123 745	30 180
Video LQ	200 380	76 825	84 134	20 192
Video MQ	55 112	18 156	34 708	7 817
Video HQ	59 073	19 526	33 049	7 920
Screen Sharing	41 170	8 800	29 216	6 870
FEC Audio	146 567	41 247	-	-
FEC Video	45 591	2 164	-	-

in identifying them as separate classes. Hence, seven classes are considered when analysing Webex data.

We employ the debugging logs to gather the ground truth, which maps each RTP stream to the content type. For Webex, logs are automatically generated during each call, while for Jitsi we use the Chrome browser WebRTC logs.[8] The logs for both applications contain per-second statistics for each stream, including the type of media (audio, video or screen sharing), the video resolution and the number of frames per second. During each call, the participant who captures the traffic also collects the logs, which we store alongside the `pcap` trace. Note that we cannot use the RTP Payload Type field for this, as it is dynamically assigned.

We collect traffic for approximately 62 hours of video calls, exchanged during 27 meetings with Webex and 50 meetings with Jitsi. They sum up to 90 GB of `pcap` files, which include the call traffic as well as a small amount of background traffic that we neglect. The dataset contains 3 977 RTP streams for Webex and 521 for Jitsi. Each call contains a different mix of the above classes, and includes traffic generated by all participants as captured from the point of view of a single individual. Out of the 77 calls, 35 have only two participants, 11 have three participants and 31 include more than three. In Table 7.2 we give an overview of the dataset, separating the training and test set. In Section 7.5 we describe our training/testing methodology in detail. For each RTC application and class, we report the amount of data we collected, in seconds. The most represented classes for both applications are Audio and LQ video.

[8]This log can be obtained by creating and downloading a dump at *chrome://webrtc-internals*

While this is somewhat expected for audio, the prevalence of LQ video is due to the video thumbnails used in the applications to show inactive participants during calls with more than three participants. Note that for Webex, FEC audio is also widely represented. The least represented class is Screen Sharing, but the overall dataset imbalance is still limited, with the ratio between the support of the most and the least represented class being less than 6.

7.4.2 Characterization and Challenges

We provide a high-level overview of the dataset in Figure 7.3, where we plot the Cumulative Distribution Functions (CDFs) for different stream features, separately by application. We use different lines to contrast the four video-based classes, plus audio. The leftmost figures show the bitrate distribution for Webex (Figure 7.3a) and Jitsi (Figure 7.3d). For each stream, we compute the average bitrate using 1-second bins. We first note that better video qualities tend to have higher bitrates (e.g., red and green lines). Audio (cyan line) has the lowest bitrate, as expected. However, the two applications present different shapes for the video curves. Webex displays smooth distributions, indicating that it adjusts the target bitrate of the video codec. In contrast, Jitsi exhibits a cascading behaviour, indicating thresholding and somewhat quantized bitrates. Note that the same video quality appears with multiple evident bitrate peaks. For example, MQ video (green dashed line) presents two peaks roughly at 0.5 and 1.5 Mbit/s, both corresponding to 640×360 video. The Screen Sharing class (solid blue line) exhibits the greatest variability. Again, this is expected, as it carries diverse contents, from slide sharing to scrolling through the screen, to effectively playing a video. This leads to a generally low bitrate with short periods of high activity. We note that setting a simple threshold on the bitrate would not yield accurate class predictions. This is especially true for Webex, where the distributions overlap significantly. In particular, for screen sharing, the bitrate ranges from a few kbit/s to more than 1 Mbit/s. Interestingly, the Screen Sharing bitrate is often as low as an audio stream, for both applications.

Similar considerations hold for the packet size (Figures 7.3b and 7.3e). Better video qualities tend to use larger packets as they sustain a higher bitrate. Again, we observe a high overlap of Screen sharing with all other classes. For Webex (Figure 7.3b), Screen Sharing packets can be as little as those of audio streams. Con-

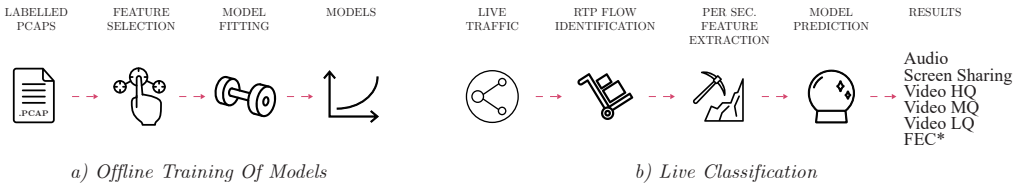


Fig. 7.4 Overview of the training and classification pipeline.

versely, for Jitsi (Figure 7.3e), only audio uses small (100-150B) packets, potentially easing its identification.

Finally, the rightmost figures show the distribution of packet inter-arrival time for Webex (Figure 7.3c) and Jitsi (Figure 7.3f). We compute it as the time interval between two consecutive packets in the same RTP stream. The video distributions partially overlap, with Screen Sharing presenting inter-arrival time as large as 400 ms when nothing on the screen is changing. Figure 7.3 shows that a careful mixture of these features is required for accurate prediction. In the remainder of the paper, we show that it is possible to identify the type of media stream with high accuracy using features derived from these traffic characteristics and a machine learning classifier.

7.5 Methodology

In this section, we describe the proposed approach, from RTP traffic identification to feature extraction and classification. We envision an offline training of a classification model and its application to live traffic in real-time. We sketch a high-level overview of our approach in Figure 7.4. We also describe in detail the methodology to build and select the features from RTP traffic. We follow the same approach for both Webex and Jitsi and create a separate classifier for each. Throughout this section, we use Webex as a running example to facilitate the understanding of the methodology.

Problem Statement. Our goal is to classify the RTP streams that we observe on the network to one of the classes listed in the previous section and Table 7.2. We want to solve this task in real time, i.e., make a decision based solely on the traffic observed in a short time interval, by applying a model trained on historical data. Thus, our classification target is an RTP stream as observed during a certain time bin (from 200 ms to 5 s).

RTP Stream Identification. We identify the RTP traffic with straightforward deep packet inspection, by matching the protocol headers. Indeed, the RTP header includes fixed-sized fields that facilitate its identification, and its sequence number serves as a simple sanity check for identification, since it must increase by 1 for subsequent packets. Popular passive meters identify RTP flows using DPI – e.g., Tstat [149] or nProbe [107]. Note that we do not handle the case of RTP tunneled through an encrypted channel (e.g., over a VPN or IPSec tunnel), since we cannot distinguish the different streams. We separate multiple media streams via their SSRC. We are not interested in the control traffic for, e.g., session establishment or login, and thus neglect it. We also assume that we know the application in use (Webex or Jitsi), since different techniques may be used for this purpose. In some cases, RTC applications provide public lists of the relay server IP addresses or use well-known ports [104]. Webex, for example, uses UDP port 5000 for RTP streams. In case such an approach is not feasible, it is possible to leverage ML-based solutions. In our previous work [112], we showed how to guess the RTC application in use with high accuracy using the domain names that the client resolves over the DNS prior to the call and an ML classifier.

The ML Pipeline. A single RTP stream results in many samples (one per time bin) that we shall classify. For our classification problem, we follow the classical approach of supervised learning. First, we extract meaningful features from the data, guided by domain knowledge on network traffic and RTP protocol. Then, we perform a two-step feature selection process by first discarding highly correlated features and then performing a recursive feature elimination. Finally, we train a machine learning classifier and evaluate its performance on an independent test set. Feature selection and algorithm training are performed offline, while the system is designed to compute features and classify new samples in real time. The time it takes is equivalent to the chosen time bin plus the feature computation and algorithm run, whose execution time is negligible. Our code is written in Python and uses the scikit-learn library [150] for machine learning. Our methodology is readily amenable to parallelization, as all processing is done on a per-flow basis – i.e., feature extraction and classification only need to obtain data from a single stream. Therefore, a multi-core parallel approach is fully feasible, and we do not expect any bottlenecks in high-speed deployments, provided packet capture is adequate. In case of deployment with off-the-shelf hardware, high-speed packet capture libraries (e.g., DPDK[9]) together

[9] https://www.dpdk.org/

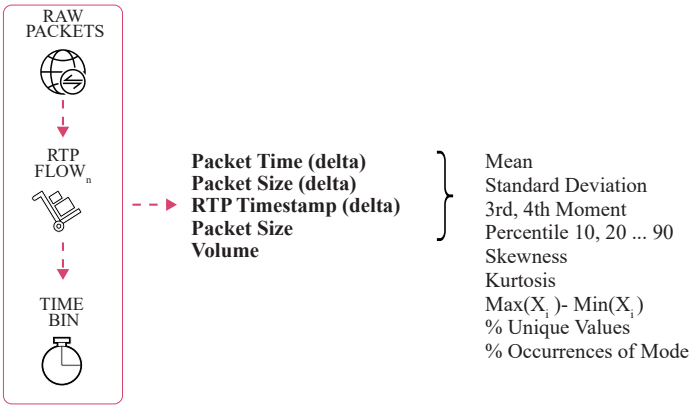


Fig. 7.5 Features derived from packets.

with Network Cards natively supporting load balancing (e.g., Receive-Side Scaling on Intel cards) would perfectly serve at this goal.

Train/test Methodology. We split the call dataset into a training and a test set to prevent overfitting and obtain robust results. We perform feature selection and algorithm hyper-parameter tuning on the training set, and we evaluate classification performance on the test set (which we never use at training). Note that the streams of a single call are used either at training or testing time to keep the two sets completely independent. For Webex, out of 27 calls, we use data from 22 calls for training and data from the remaining 5 calls for testing. For Jitsi we use data from 41 for training and 9 for testing. With this split, we obtain roughly 80% of samples (1-second bins) for training and 20% for testing (see Table 7.2). We also verify that each class is well-represented in both sets. As a global performance indicator, we use the macro-average (a simple mean) of the F1-scores of each class.[10] For some analyses, we also consider accuracy as a concise index of overall performance, since classes are not strongly imbalanced.[11]

Feature Extraction. We extract features from the packets separately by RTP stream and time bin. The features are based on the fields of the RTP protocol and take into consideration its operation. We outline our approach in Figure 7.5. We consider five groups of features, reported in the middle column of the figure, in bold. These include packet characteristics (size, time, volume) and the RTP timestamp field, which indicates the time at which the content was generated at the source. RTP has a

[10]The *F1-Score* is the harmonic mean between *Precision* and *Recall* of a class.
[11]The *accuracy* is the share of correct predictions over the total.

few other fields that essentially indicate header extensions, which we do not include because they are very application and client-specific. Since two of the selected fields (packet time and RTP timestamp) represent time instants, we only consider their relative variation across packets, since the absolute values are useless in our context. For packet size, we use both absolute and relative values. We extract these five values for all packets and compute various statistical indices to create the final features, such as range, mean, standard deviation, percentiles, third and fourth moments, etc. Since we find that the same values recur frequently in the packets, we also add features that measure the number of unique values, the percentage of occurrence of the most frequent value (mode), and the ratio between the minimum value and the range. We report the complete list of statistical indicators on the last column of Figure 7.5. Finally, we consider the traffic volume in terms of the number of packets and bitrate observed in the time bin. We also use the number of packets with the RTP marker flag set as a separate feature.

Since our goal is to design a real-time classification system, we create features that can be computed on the fly by considering only the packets observed in a time bin. Intuitively, the smaller the time bin is, the faster the stream is classified. However, features are more representative with larger time bins since they are computed over a more extensive set of packets. In Section 7.6, we explore this trade-off and evaluate how the temporal granularity affects the classification of an entire stream. Finally, note that we also avoid features that require linking multiple streams to keep our design simple and easily parallelizable.

Feature Selection. In total, we extract 96 features derived from the four empirical distributions mentioned above, plus volume. We publish the full list of features on our research center website.[12] To remove those that are redundant and shrink the overall number of features, we perform a two-step selection process.

1. *Correlation analysis*: We perform an initial feature selection by measuring the correlation between each pair of features. We evaluate all possible pairs in a random order, and whenever we find a Pearson correlation coefficient greater than 0.9 (in absolute terms), we keep only one of the two features at random. With this step, we roughly eliminate half of the features.

[12] https://smartdata.polito.it/rtc-classification/

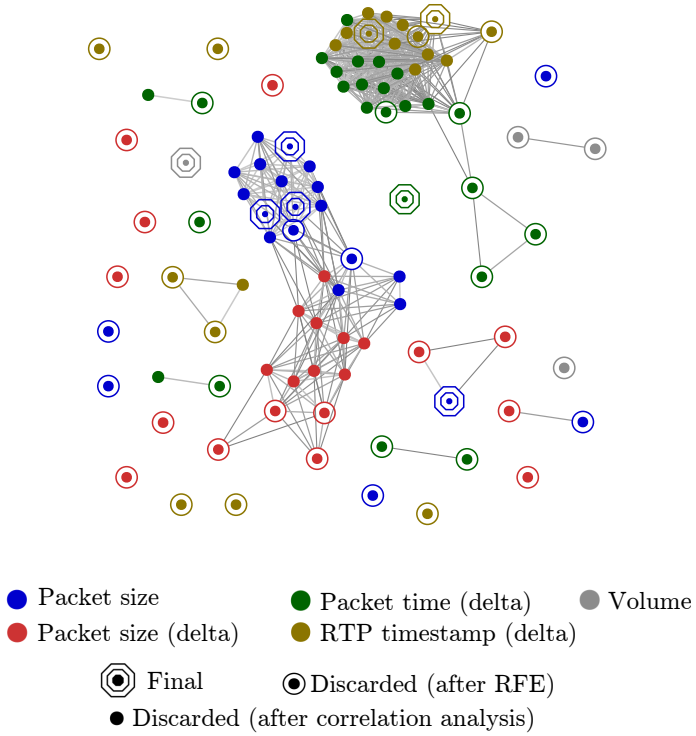


Fig. 7.6 Graph representing the correlation between features. The color indicates the feature set, the shape whether the feature is kept after feature selection and the distance represents the correlation.

2. *Recursive Feature Elimination using the ExtraTree algorithm*: We use the Recursive Feature Elimination (RFE) approach [151] to refine our list of features, maintaining only those that are most useful for our classification problem. Using RFE, we train an ExtraTree classifier on the training set and rank the features by their *feature importance* as provided by the algorithm.[13] We then eliminate the one with the least importance. We recursively repeat this procedure until we reach the minimum number of features and the best performance, which we evaluate using 5-fold cross-validation. Note that tree-based feature ranking is known to be biased in the case of groups of correlated features [152]. Thus, our first step (correlation analysis) is essential for RFE to work correctly.

We graphically illustrate the entire feature selection process for Webex with Figure 7.6, which shows the initial 96 features in the form of a graph. Each node

[13]The ExtraTree classifier natively exposes the feature importance after training.

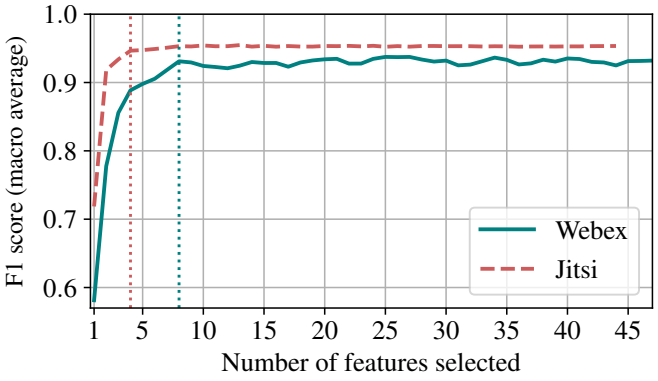


Fig. 7.7 Mean F1 score when varying the number of features. The vertical lines indicate the final number of features.

represents a feature, and the length of edges is (roughly) inversely proportional to the correlation among pairs in absolute value – i.e., highly correlated features remain close to each other. For illustration purposes, we only show the edges where the correlation is higher than 0.5 (in absolute value). Different colours represent the feature sets, while the shape of each node indicates whether a feature is maintained or discarded at one of the selection steps: a circle means that the feature was discarded after correlation analysis, a double circle means that the feature was discarded with RFE, and an octagon means that it passed both steps and is included in the final list.

We first notice that the correlation analysis step maintains all features which are poorly correlated with other ones: all nodes without edges are either double circles or octagons. On the contrary, among groups of highly correlated features, only a few samples are retained. For example, the dense community in the top right of the figure includes the percentiles of packet time inter-arrival time and RTP timestamp, which are intuitively highly correlated. We retain only two of them.

Continuing with the running example of Webex, the first step of the feature selection shrinks our set from 96 to 47 features. We then perform RFE to obtain only those that are useful for our classification problem. We train an ExtraTree classifier on the remaining 47 features, running a 5-fold cross-validation to evaluate how accurate the obtained model is. We then eliminate the feature ranked as least important and repeat this process until we find that the classification performance starts to decrease. In Figure 7.7, we show how the average F1 score varies when

removing an increasing number of features. The figure shows our results for both Webex (solid blue line) and Jitsi (red dashed line).

Considering Webex, when we use all 47 features, we get an F1-score of 0.91. The performance is almost stable (with minimal variations) until we use 8 features only – i.e., we eliminate 39. Then, the accuracy starts decreasing consistently. After analysing the curve, we decide to set the final number of features to 8. Interestingly, we notice that every feature group (except the packet size) appears in the set of the final features (there is an octagon of every colour except red in Figure 7.6). Among the final features, we find the packet size (mode, 25^{th}, 70^{th} and 75^{th} percentile), the 30^{th} percentile and mode of the RTP timestamp delta, the mode of the inter-arrival time and the number of packets with the RTP marker flag set. Intuitively, for each characteristic of the packets, we keep a few statistical properties of its distribution.

The process is similar for Jitsi (red dashed line in Figure 7.7). Note that the curve ends at 43 features, since for Jitsi the first step of feature selection eliminates a slightly larger number of features. The knee in the line shows that we already achieve good performance with as little as 4 features. Among them, we find three representatives of the packet size feature group and the mode of the RTP timestamp delta. This indicates that the packet length is a vital factor for this classification problem.

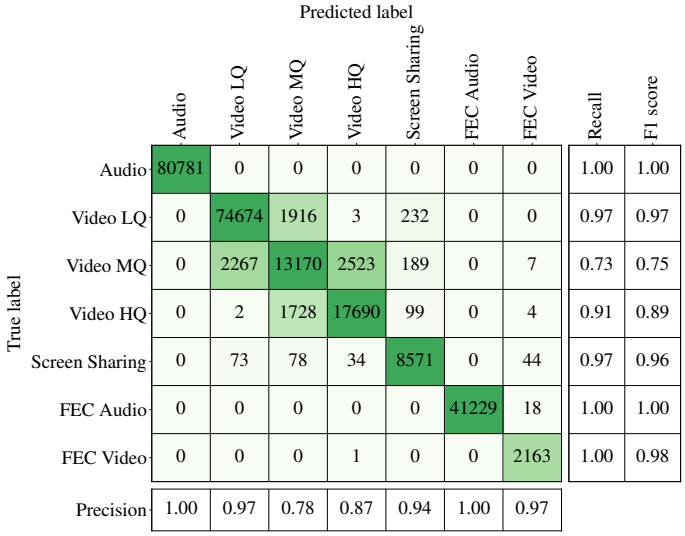
Multi-class Classification. Using the features that we obtain after the feature selection, we try different classification algorithms to find the one that yields a proper trade-off between performance and simplicity. The algorithms we consider are: tree-based classifiers [Decision Tree (DT) and Random Forest (RF)], k-Nearest Neighbors (k-NN), which classifies points based on proximity to other data points, and Gaussian Naïve Bayes (GNB) as a generative probability model. We perform hyper-parameter tuning with 5-fold cross-validation for each of these models, using the training set uniquely. We then evaluate their performance on the separate test set, using the macro-averaged F1-score as a performance indicator. In Section 7.6, we show that the algorithm choice has a moderate impact on classification performance.

7.6 Experimental Results

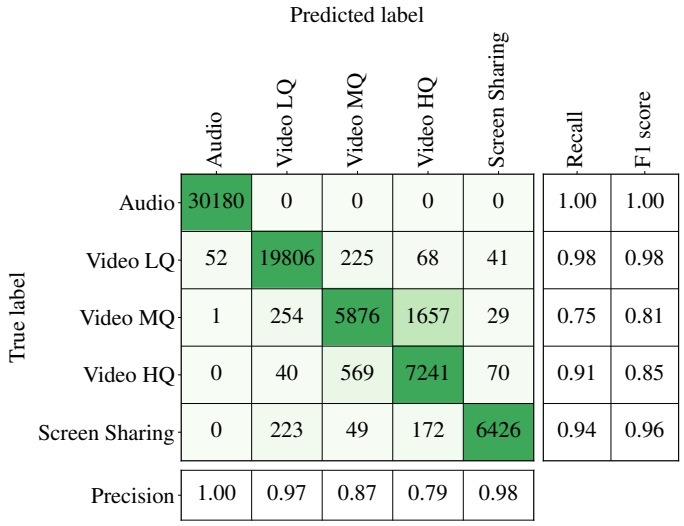
In this section, we present our experimental results for the entire classification problem. First, we discuss the overall classification performance and quantify the impact of the time bin duration, classification algorithm and training set size. We then discuss the importance of the features and analyze how classification errors arise. Finally, we investigate the possibility of transferring a model trained for one RTC application to another. All results are obtained by training classification models on the training set and evaluating their performance on the independent test set.

7.6.1 Classification Performance

We first report and discuss the performance we obtain for both RTC applications when using the best models. Indeed, we try different classification algorithms and finally opt to use a Decision Tree classifier, which provides good performance and a simple model. Running hyper-parameter tuning, we obtain the best results when using the Gini index as a purity measure. In Figure 7.8, we show the confusion matrices for both Webex and Jitsi using a 1s time bin. By definition, a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i and predicted to be in group j. Thus, the main diagonal represents the number of correctly classified samples. We also show the per-class recall and F1-score in the last two columns and precision in the bottom row. We note that, for both applications, all classes except Video MQ and HQ exhibit an F1-Score above 0.96, and thus high precision and recall. Audio is the best performing class for both RTC applications, together with FEC audio for Webex. Here only a handful of samples are misclassified, suggesting that audio streams are generally easy to isolate. Indeed, for Jitsi especially, audio streams tend to use smaller packets than video (see Figure 7.3), making their identification simpler. The worst performing class is video MQ, with F1 scores of 0.73 and 0.75 for Webex and Jitsi, respectively. The confusion matrices reveal that the three different video qualities are, in some cases, confused with each other. Although this is a flaw of our classification model, we tolerate this behaviour given the similar nature of the three classes. Also, keep in mind that applications (especially Webex) use video codecs with variable bitrates that result in different network traffic (see Section 7.4). Overall, for Webex, 96.3% of the samples are



(a) Webex

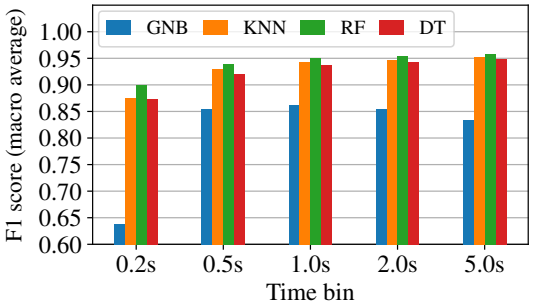


(b) Jitsi

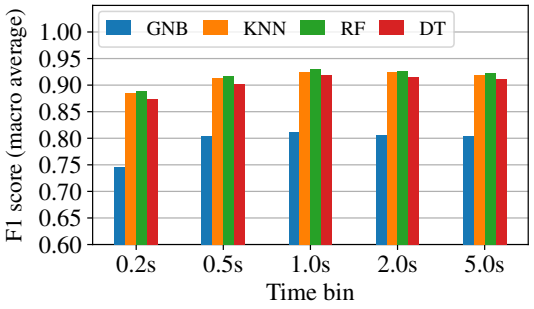
Fig. 7.8 Confusion matrices when using a Decision Tree classifier and 1s time bins.

classified correctly (i.e., accuracy), and the average F1-score is 0.94. For Jitsi, we obtain an accuracy of 95.3% and an average F1-score of 0.92.

Considering computational time, our system needs to perform 3 consecutive steps before providing the final classification label: (i) Wait for the time bin to gather traffic information, (ii) Calculate the features and (iii) Apply the classification model. Step (i) obviously takes most of the time. Step (ii) depends on the class,



(a) Webex



(b) Jitsi

Fig. 7.9 Performance of the four algorithms for different time bins.

with Video HQ being the most expensive as it sends the highest number of packets, thus increasing the number of samples in the calculation. On average, this step takes few milliseconds with our Python code on commodity servers. Finally, step (iii) is even faster, requiring the use of a light-weight decision tree model, that takes tens of microseconds. For high-speed deployments, we envision the use of a parallel multi-core architecture to scale the processing. Such an approach is completely feasible since the classification relies on features extracted on a per-UDP flow basis.

7.6.2 Parameter Sensitivity

We now discuss the impact of the time bin duration on the classification performance. Indeed, we are interested in classifying a stream as fast as possible without sacrificing accuracy. Figure 7.9 shows how performance varies with different time bin durations, from 200ms to 5s. We provide results for 4 classification algorithms, and the y-axis reports the average F1-score we obtain. We find that we generally get better results with larger time bins. This is no surprise since the features are computed over

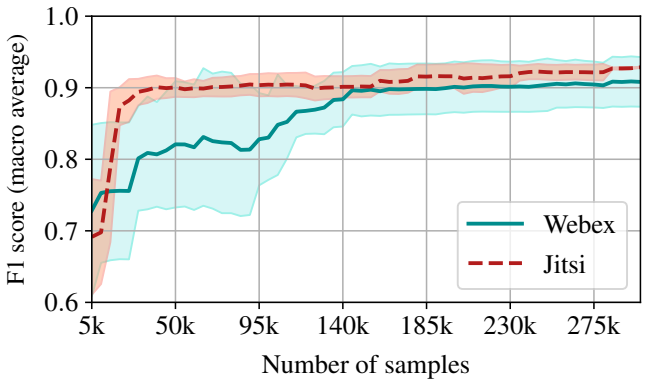


Fig. 7.10 Learning curve: Relationship between the number of training samples and the F1-score.

more extensive sets of packets. For example, in 200ms of a typical audio stream, only 10 packets are generated. The performance flattens for values larger than 1s for both applications, implying that such a time frame is large enough to capture representative features about a stream. We believe that a delay of 1s is not critical, since RTC calls typically last minutes.

Looking at Figure 7.9, we can also compare the performance of different classification algorithms. We observe no large differences, except for Gaussian Naïve Bayes, which exhibits somewhat worse performance, probably due to the simplicity of the model. Note that the lowest F1-score is 0.62 for Webex and 0.73 for Jitsi. This confirms that our careful feature engineering and selection make the results robust to the choice of algorithm. We finally opt to use a Decision Tree for its simplicity, interpretability and speed. Random Forest produces similar results, but is more computationally intensive as it uses trees in parallel, 100 in our case. k-NN also performs well, but requires the model to store the entire training set in the main memory, resulting in significant memory consumption. Using a decision tree instead, the model is only a few *kB* in size. Comparing the two applications confirms that they exhibit very similar performance, with Jitsi having a lower F1-score by about 0.02 in most cases.

7.6.3 Training Set Size

We now investigate how much training data is necessary to achieve good classification performance. To this end, we train many classification models, gradually increasing

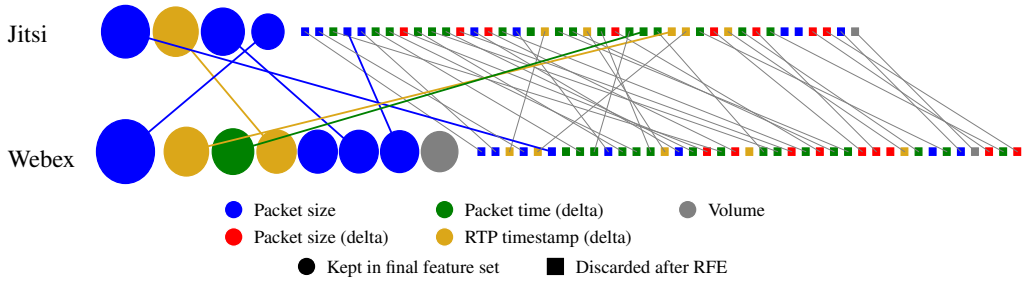


Fig. 7.11 Feature importance comparison between Webex and Jitsi.

the size of the training set. We vary the number of training set samples selecting them from the least possible number of calls. In other words, we entirely consume the samples from one call before drawing them from a second. In this way, we indirectly observe how many calls are required. Note that randomly selecting training data from all calls would likely sample the diversity of the entire dataset, which is unfair for our analysis. In this experiment, we use Decision Tree classifiers with 1s time bins.

Figure 7.10 shows the classification performance versus the training set size. Again, we measure the performance using macro-averaged F1- score on the test set. We repeat each experiment 5 times, shuffling the order of the calls but still drawing samples from one call altogether. The solid blue and red dashed lines indicate the mean score of the experiments for Webex and Jitsi, respectively. The areas represent the standard deviation across the runs. Starting from Jitsi, we notice that the performance improves very quickly with the training set size– with only 20k samples, the F1- score is already above 0.86. Such an amount of time corresponds to 5 hours of audio and video call. After that, it increases very gradually, reaching a local maximum of 0.92 F1 score at 200k samples (55 hours of calls). The standard deviation is generally small and stable. This result suggests that the features we extract and the nature of the problem do not require a large dataset to obtain a reliable model. Conversely, Webex requires a larger training set for accurate classification, exhibiting a slow growth and a larger standard deviation, stabilizing at 145k samples (40 hours of calls). This is likely due to the higher number of classes (with the additional audio and video FEC classes) and a variegated behaviour of the application within a call. Indeed, we observe that there is an abundance of audio and video LQ in various calls and a deficiency of the other classes. Consequently, additional calls are necessary to bridge the gap. To test this conjecture, we perform an additional

experiment where we balance the number of samples per class and find that the performance converges faster.

7.6.4 Feature Analysis

We now discuss the outcomes of the feature selection phase. Our goal is to investigate whether we can recommend a fixed set of features for any RTC application or they are specific for each one. As described in Section 7.5, we carry out a two-fold feature selection: we first remove highly correlated features, and then we perform recursive feature elimination using an ExtraTree classifier. In Figure 7.11 we compare the results of the second step for Webex and Jitsi. Each symbol represents a feature that we retain after the correlation analysis – 43 for Jitsi (upper row) and 47 for Webex (lower row). Circles represent the features that are finally selected, and their size is proportional to the relative importance given by the ExtraTree classifier. The squares represent the remaining features, that were discarded using RFE. We arrange them in the order in which they were discarded. The colours indicate the feature group, similar to Figure 7.6. The edges connect the same feature on the two RTC applications so we can compare Jitsi and Webex.

As anticipated in Figure 7.7, with Jitsi, 4 features are enough to achieve good performance, while Webex needs 8. Looking at Figure 7.11, we observe a large presence of features related to the packet size (blue) – 3 out of 4 for Jitsi and 4 out of 8 for Webex. This is expected, as the packet size is instrumental for distinguishing audio and video streams (see Figure 7.3). We note that 3 of the Jitsi features also appear in Webex, albeit with different importance. Overall, the features are ranked similarly for the two applications, and the Spearman's rank correlation coefficient between the two ranks (including all features shown in Figure 7.11) is 0.70. Interestingly, two features chosen for Webex have been discarded in the first feature selection phase for Jitsi – two circles on the bottom row are not connected to any of the above shapes. A notable one is the number of packets with the RTP packets with the marker flag set (the gray circle). We note that this feature correlates strongly with frame rate in video streams, and speculate it helps identify the screen sharing class, typically with a low frame rate.

7.6.5 Error Analysis

We now analyze misclassification cases to understand (i) how they are spread among streams and (ii) whether they can affect the prompt classification of streams.

Overall, we obtain an accuracy of 96.3% for Webex and 95.3% for Jitsi, as detailed in Section 7.6.1. Here, we want to measure whether these errors are concentrated on a few RTP streams or are scattered between all. To this end, in Figure 7.12, we plot the complementary cumulative distribution function (CCDF) of the percentage of errors per RTP stream. In other words, for each stream in the test set, we compute the percentage of misclassified samples and then show the distribution over all streams. The test set includes 508 streams for Webex and 101 for Jitsi. We observe that most of them present a rather low error rate. For Webex (solid blue curve), we notice that the probability of misclassifying more than 10% of the samples of a stream is $\approx 10\%$. Moreover, the probability of misclassifying more than 50% is less than 2%. This result suggests that, in general, mistakes span through many different streams rather than all originating from a few, and our classifier typically does not commit systematic errors. Similar considerations hold for Jitsi. There are only a handful of streams for which most samples are assigned to the wrong class – see the right-most side of the plot. These are usually short-lived streams (shorter than $10s$), except two long Webex video MQ streams where 68% of samples are misclassified and one long Jitsi video MQ stream with 73%. As reported in Section 7.6.1, video MQ is the hardest class to discern. In conclusion, these results show that the misclassification of an entire flow is very unlikely to happen.

We now investigate the possibility of classifying an entire stream just by looking at the first few samples. It might be beneficial in some real deployments when the network must react quickly to new streams to – e.g., prioritize particular traffic classes (see Section 7.3 for possible deployment scenarios). To this end, we suppose to classify a new *stream* based on the first N samples, using a majority vote scheme on the labels we obtain for those samples. In other words, given the first N samples of a stream, we assign it entirely to the class most samples have been assigned to. In Figure 7.13, we show the macro-averaged F1-score we obtain, varying N between 1 and 30 seconds. In this case, the classification goal is a *stream* rather than a *sample*, and, as such, we compute performance metrics over the streams in the test set. When classifying the stream based solely on the first second, we obtain 0.92 macro-averaged F1-score for Webex (solid blue line) and 0.82 for Jitsi (red

dashed line), as sporadic errors have the maximum impact. Increasing the number of samples N, we obtain better results, reaching macro-averaged F1-Score of 0.99 and 0.93 for Webex and Jitsi, respectively. Indeed, our classifier hardly perpetrates systematic errors (see the previous paragraph), making the majority voting scheme very robust to misclassification. We conclude that our approach is fully appropriate in contexts where the network is required to quickly make decisions on an entire flow, e.g., installing appropriate SDN rules on the network switches.

7.6.6 Model Transfer to other Applications

In our previous results, we train a classifier with labelled data belonging to the same RTC application that we aim at classifying. This might not always be possible, as labelled data are hard and expensive to obtain. Moreover, new RTC applications may spread rapidly without controlled experiments being possible. In this section, we explore to what extent a classifier trained for RTC application A can be used to classify streams of the application B.

For our goal, we investigate the use of *transfer learning* techniques [153], whose goal is to transfer knowledge from one *domain* (i.e., one RTC application) to another. These techniques are useful when we cannot collect labelled data in the second domain. In this case, we can try to use the knowledge from domain A to solve the same problem in domain B. In general, the rationale behind transfer learning techniques is to modify and adapt an ML classifier trained in domain A to classify samples in domain B.

Here, we employ the domain adaptation technique called CORrelation ALignement (CORAL) [120]. As the name suggests, given the feature distributions from two domains (A and B), CORAL tries to align the covariance matrix (matrix of second-order moments) of distribution B to the one of distribution A. Due to the nature of our problem, we hypothesize this approach suitable since we target two similar RTC applications that use the same network protocols. Necessary for our goal, CORAL is an unsupervised technique, as it assumes data for domain B are available, but without class labels.

We here investigate the performance we obtain when using a classifier trained on application A (e.g., Webex) for classifying data of application B (e.g., Jitsi). We perform experiments (i) using the classifier directly on application B and (ii) using

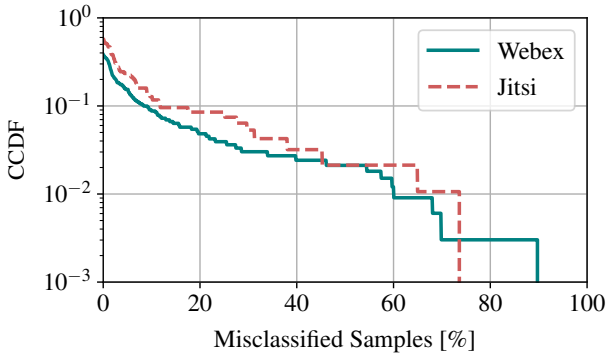


Fig. 7.12 CCDF of percentage of errors per stream.

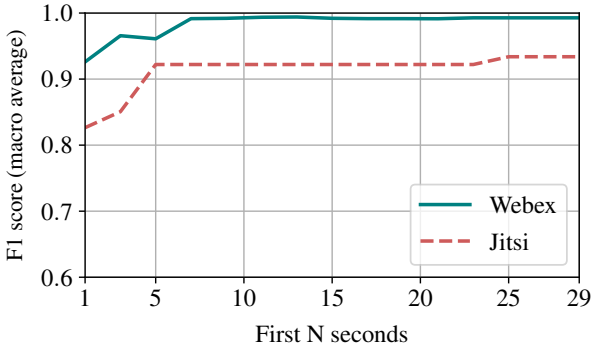


Fig. 7.13 Classification performance using first N samples per stream.

CORAL to align domains A and B. Case (i) corresponds to using a classifier directly outside of the training context. In case (ii), we assume that non-labelled data for application B are available, allowing the use of CORAL to align the two domains. We show the results in Figure 7.14, again measuring performance in terms of macro-averaged F1-Score. The x-axis reports the domain on which the classifier is trained, while the colour of the bars indicates the domain on which we use it. We provide a reference using the green bars, indicating the performance we obtain when we use the classifier in its domain –i.e., the approach we used in the previous sections. For this experiment, we remove the FEC streams from the Webex traffic, since we need the same number of classes for the two applications for a fair comparison. The red bars represent case (i), while the blue bars case (ii).

We first notice how using a classifier directly on a different RTC application entails a certain performance drop (red bars). Indeed, using a classifier trained on Webex to classify Jitsi streams leads to a 0.67macro-averaged F1 score. In the

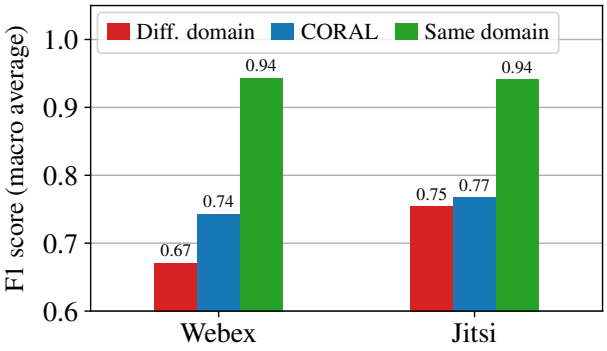


Fig. 7.14 Classification performance varying the target domain.

opposite direction (training on Jitsi and testing on Webex), the performance is slightly better (0.75). The use of CORAL improves the performance in both directions, yielding similar results in both directions (blue bars). We get an F1-Score of 0.74 when training on Webex and using Jitsi and 0.77 vice-versa. Interestingly, the benefit of CORAL is higher in the former case (+0.07), while minimal in the second (+0.02). Nevertheless, it is still far from the performance obtained by training a model on the same domain, which then soars to an F1-score of 0.94 for both applications (green bars). This might originate from the different shapes of traffic distributions between the two RTC applications, as discussed in Section 7.4. In summary, our results suggest that it is possible to use a classifier for a different application if lower performance can be tolerated. If non-labelled data for the target RTC application are available, CORAL is instrumental in increasing the performance.

7.7 Takeaways

In this chapter, we presented an extensive study on the classification of Real-Time Communication (RTC) streams using machine learning techniques, with a focus on RTP traffic in applications like Webex and Jitsi. The key findings from our study can be summarized as follows:

- **Methodology:** We developed a methodology involving RTP stream identification, feature extraction, and a machine learning pipeline for real-time classification of media streams.

- **Classification Performance:** Using a Decision Tree classifier, we achieved high classification accuracy for both Webex and Jitsi, with most classes exhibiting F1-Scores above 0.96. The approach was particularly effective in isolating audio streams.
- **Parameter Sensitivity:** Our results indicated that classification performance generally improved with larger time bins, with a 1s time bin providing a good balance between speed and accuracy.
- **Feature Analysis:** The study highlighted the importance of packet size features in classification. The feature selection process, including correlation analysis and recursive feature elimination, was crucial in identifying the most relevant features for each application.
- **Error Analysis:** Misclassifications were generally distributed across streams rather than concentrated, suggesting that our classifier does not commit systematic errors.
- **Model Transferability:** We explored the concept of transfer learning for RTC stream classification, finding that while direct application of a model trained on one application to another led to performance drops, techniques like CORrelation ALignement (CORAL) could improve cross-application performance.
- **Deployment Scenarios:** We demonstrated the potential of our classification system in RTC-aware network management, showing how it can improve Quality of Experience (QoE) by intelligently managing traffic based on the type of media stream.

In conclusion, our research successfully demonstrates a versatile and efficient approach for real-time classification of media streams in RTC applications, setting a new standard in network management and QoE optimization. By making our dataset and code publicly available, we aim to inspire further advancements in this promising field.

Chapter 8

Machine Learning for QoE in Satellite Communication

8.1 Motivation

The content presented in this Chapter is primarily derived from our paper *Monitoring Web QoE in Satellite Networks from Passive Measurements*, which, as of the writing of this thesis, has been accepted for presentation at the *IEEE Consumer Communications & Networking Conference*.

Satellite Communication (SatCom) offers Internet connectivity where traditional infrastructures are too expensive to deploy, including rural areas and the territory of underdeveloped countries. Here we focus on the GEO SatCom technology, which relies on satellites positioned in geostationary orbits. Propagation delay makes the Round Trip time (RTT) higher than 550ms [154, 92] so that the substantial hurdle of high link latency significantly impairs traditional interactive browsing experiences [155–157]. For instance, the download of a webpage can take several seconds.

In this context, monitoring the Quality of Experience (QoE) subscribers obtain becomes a crucial factor [158–160], and it would allow the SatCom provider to detect anomalies, plan network upgrades, and optimize management policies. Because quantifying Web QoE remains a formidable challenge due to its subjective nature, proxy quality metrics like “onLoad” (or Page Load Time) and “Speed Index” are

used as indirect measures within web browsers [161]. Recent research efforts have focused on proposing unsupervised [162, 163] and supervised [164–166] approaches for measuring web browsing QoE in traditional networks. However, these approaches cannot be directly used for SatCom scenarios. This is especially true because operators deploy middleboxes to mitigate the impact of latency, i.e., Performance Enhancing Proxies (PEPs) [94]. The impact of PEPs on QoE is not clear, and the modification to the traffic they cause challenges traditional QoE estimators.

This Chapter aims to fill this gap. We propose a system to monitor Web QoE specifically designed to address the challenges unique to SatCom. We employ a supervised Machine Learning (ML) approach to predict Web QoE from in-network passive measurements and explore the different factors that affect prediction accuracy. Our system gathers training data through Test Agents, which automatically visit the website to monitor. Ingenuity is needed to design such Test Agents, as they need to generate traffic patterns similar to real subscribers' traffic and we illustrate the most significant challenges to achieve this goal. In addition, the complex nature of Web traffic requires ad-hoc approaches to construct meaningful features. After designing a proper ML pipeline to predict the QoE of a given website, we explore the possibility of training if a single ML model extracts the QoE of multiple websites. Our results are negative and show that a website-specific model must be considered. At last, we explore the model drift with time and propose a continuous learning solution to address this problem.

Our findings demonstrate the feasibility of monitoring Web QoE in SatCom environments, albeit with some limitations. Notably, this approach works well with the subset of websites whose pages include objects from multiple domains. On the contrary, for websites hosted on a single infrastructure, the prediction accuracy remains limited. Periodic generation of training data becomes essential to adapt to evolving web dynamics, and it remains an open challenge to build a general ML model able to predict the QoE of websites unseen at training time.

The rest of the Chapter is organized as follows: in Section 8.2 we present the related work while in Section 8.3 we describe our methodology for feature engineering and classification. Section 8.4 shows our experimental results, and, finally, Section 8.5 concludes the chapter and discusses important takeaways.

8.2 Related Work

The QoE of Internet users is a wide and complex topic as it involves the subjectivity of users and has been extensively studied in the literature [167, 168]. Several works address the challenges of gathering meaningful metrics and the impact of different network conditions [169, 158–160, 170]. Commonly, Web QoE is measured through proxy metrics, which have been shown to be correlated with users' subjective experience. Notably, Page Load Time (also called OnLoad) and Speed Index are the most widely adopted, although Bocchi et al. [161] have shown that they do not necessarily model all factors influencing users' perceptions. More recently, "Above-The-Fold" metrics have been proposed as a more accurate estimation of users' QoE [56].

Regarding SatCom environments, the literature extensively explores the role of the physical layer on the seamless network operation and Quality of Service [171, 68]. Several works proposed approaches to enhance browsing performance in SatCom environments using the most diverse techniques, including Performance Enhancing Proxies and HTTP caching [155–157]. However, there is still a dearth of studies evaluating or measuring QoE in these kinds of networks. Recent efforts regard estimating video QoE [172] or measuring the performance of the QUIC novel protocol in the SatCom environment [89, 173]

Measuring QoE using passive network measurements poses additional challenges, as it is not trivial to derive meaningful metrics or features from network traffic, especially when encryption is in place. Recent research efforts have focused on proposing unsupervised [162, 163] and supervised [164–166] approaches to monitor browsing QoE. These techniques have in common careful feature engineering and the use of machine learning models to derive measures which correlate to users' perceived QoE or to well-known proxy metrics. However, none of them has been studied specifically for SatCom environments or even tested in such a scenario, and no existing literature comprehensively studies Web QoE metrics in SatCom networks. Our work aims to bridge this gap. We engineer a system based on supervised learning and exploit some of the intuitions proposed in [162, 164], specifically the link of different flows to the same user visit. We tailor our system to be deployed at the SatCom operator's premises, where the presence of complex middleboxes (i.e., PEPs) challenges the harvest of truthful metrics.

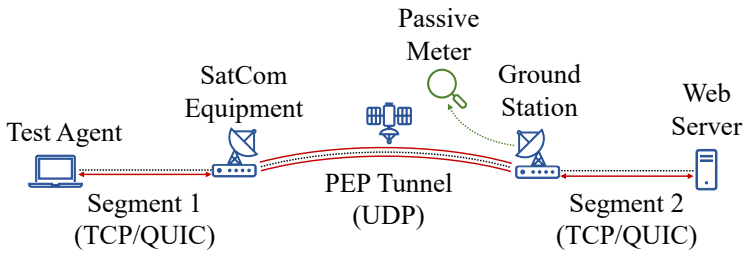


Fig. 8.1 Test Agent and Passive Meter deployment scenario.

8.3 System Design

8.3.1 Problem Statement

Our goal is to design and implement a monitoring system that uses passively-collected measurements to estimate the Web QoE of SatCom users. We target GEO operators, which rely on one or few satellites positioned at a fixed distance from Earth, providing continuous coverage over large geographic areas. In GEO SatCom, operators typically deploy PEPs to improve performance on the satellite segment. In particular, PEPs are designed to overcome TCP limitations in high-latency scenarios. For this, the PEP transparently manipulates TCP connections. Referring to Figure 8.1, the user's SatCom home gateway impersonates the server for all TCP connections initiated by the end-user devices (Segment 1 in the Figure). Acting as TCP proxy, the home gateway buffers the TCP data stream and forwards it to the operator's ground station via a bidirectional UDP tunnel (PEP Tunnel in the Figure). Complex reservation and scheduling algorithms decide how to share the SatCom link capacity among active users. The ground station receives the tunnelled traffic and acts as a second TCP proxy. It opens a new TCP connection toward the actual server to download and store the responses before forwarding them to the home gateway through the shared satellite link (Segment 2 in the Figure). The PEP complicates the collection of passive in-network measurement, as it splits each TCP flow into two sections. A passive probe installed on the ground station will then only observe Segment 2 – i.e., the TCP connection the ground station PEP opens –, and not the actual traffic as received by the end-user device. As such, any QoS metric collected for TCP (such as packet loss, throughput or RTT) might not be representative of the end-user experience. In the case of QUIC (over UDP), PEPs operate in a slightly different fashion:

Packets are tunnelled via UDP, but the home gateway cannot impersonate the server as QUIC header encryption prevents middleboxes from manipulating connections.

We depict our deployment scenario in Figure 8.1. We control a number of Test Agents that are connected as regular end-user devices. At the same time, a passive probe collects measurements at the ground station where it observes the traffic of all subscribers. The passive monitor collects per TCP and per UDP flow summaries with rich statistics. Here we rely on Tstat [38] which exports more than 100 features per each TCP/UDP flow, including the server name (as recovered from Server Name Indication (SNI) in TLS Header), the RTT, the total number of packets and bytes exchanged from the server to the client and vice-versa, the size and timing of each flow first packets, etc. Notice that the observed timings radically differ from the time at which packets are transmitted by/delivered to the subscriber's devices as packets are collected after/before they travel the satellite segment. We assume that each subscriber is uniquely identified by their subscriber IP address.

Using passive measurements, we design a system to estimate the Web QoE of subscribers using ML models that map flow-level and packet-level features to quantitative metrics. As target metrics, we consider SpeedIndex and OnLoad, as explained in section 2.4.

We use *Test Agents* to periodically collect QoE measurements resulting from automated visits to a list of monitored websites. The test agents access the internet via the Sat modem offered by the operator with a standard subscription.

8.3.2 Test Agent Design

A Test Agent aims to emulate the behaviour of a regular user by visiting a set of websites to gather the necessary QoE metrics for training the ML models. The operator chooses the list of websites to include websites of particular interest, such as business-related portals.

A Test Agent consists of dedicated physical machines connected as a regular end-user device. It uses a browsing automation suite (the dockerized version of Browsertime[1] in our case) to automatically visit webpages. We assume Test Agents continuously operate and seek to maximize (i) the diversity within collected data

[1] https://www.sitespeed.io/documentation/browsertime/

and (ii) the size of the data available to train ML models. Creating an accurate and realistic training set requires ingenuity and involves overcoming various pitfalls, here detailed.

Our Test Agent takes into account three aspects that have been recently shown to largely impact automatic web experimentation: The need to include internal pages of a target website; The need to accept the cookie policies; The need to visit pages with warm browser caches.

In fact, recent literature [174, 166] has shown that it is necessary to include websites' internal pages when running any kind of web testing. Thus, for each website, Test Agents visit at least 10 internal pages chosen to maximize diversity. Notice that several approaches can be used to automate this task. Here we opt for manually defining such a list.

Second, the presence of Privacy Banners (also known as Cookie Walls) impairs automated navigation[175]. If not specifically instrumented, the Test Agents will always visit a website as a "first visit" so that the website will show the privacy banner and will not download any third-party elements that require the user to accept the cookie policy first. Given 95% of users simply click on "accept all cookies and policies" [175], here we create a custom Javascript script to accept the Privacy Banner and continue the navigation just like a normal user would do.

Third, the presence of a browser cache radically impacts the resulting network traffic. Thus, we run repeated visits to webpages so that subsequent visits occur with a populated browser cache.

For the experiments in this Chapter, we use two Test Agents, focusing on 10 websites. We chose them among the most visited ones by the operator's subscribers. These include top websites for e-commerce, news, search engines and adult videos. In building such a list, we exclude those websites for which user login is needed to consume most of the website's content (e.g., online social networks). The website list can be derived from Figure 8.2. For each website, we select p=5 representative internal pages. We run a first measurement campaign in September 2022 and a second one from December 2022 to February 2023. Each campaign includes about 2000 visits to each website and page. In Figure 8.2 we show the distribution of QoE metrics in the form of boxplots. The boxes represent the Inter-Quartile range, while whiskers span from the 5^{th} to the 95^{th} percentile. The black stroke represents the median. Observe how different are the metrics on each website, and how the

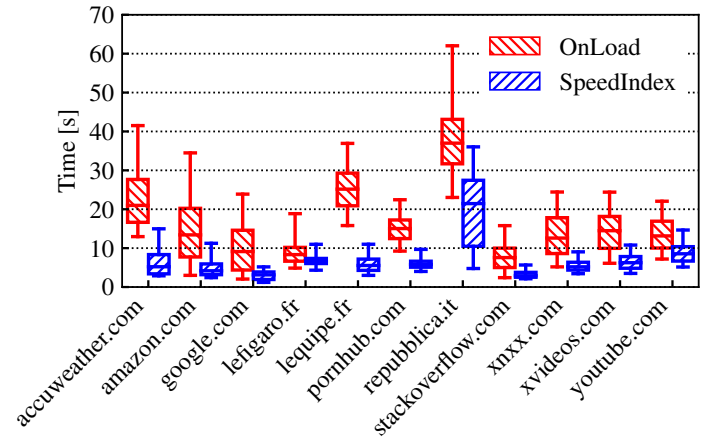


Fig. 8.2 BoxPlot OnLoad vs SpeedIndex

onLoad and SpeedIndex are in the order of tens of seconds and radically higher than those seen on wired or mobile networks [176]. This testifies to the peculiar SatCom extreme latency which impairs the browsing experience.

8.3.3 Feature Engineering

Here we propose to follow a domain expert-driven approach to engineer specific features that are highly correlated with the target QoE metrics. As done in the previous works [177, 165], we focus on time-related and volume-related metrics that we extract from the flow-level measurements. For this, to gather passive measurements, we deploy a Tstat passive probe in between the operator's ground station and the internet access where we continuously collect flow-level data.

To build the training set for ML models we first join the passive data (i.e., flow records collected at the operator's ground station) with Test Agent data (i.e., the target metrics onLoad and Speed Index). We join the Test Agent and passive data records using subscriber IP as key addresses.

Referring to Figure 8.3, we identify the beginning of a new webpage visit when a client contacts the domain of one of the target websites. To this end, we exploit the observation of a flow with the contacted domain name. We call this event *Trigger*. It serves as the observation initial point for building the features. We mark those with thick arrows in Figure 8.3. We refer to the *Trigger Flow* as f_0.

Given a *Trigger*, we open an *Observation Window* of $\Delta T = 30$ seconds during which we extract the information on the first $n = 10$ packets of the first $k = 5$ flows. The rationale behind this choice is twofold. First, the webpage rendering process entails contacting different servers to download all webpage objects. Thus, we want ML models to leverage information from the group of *Related Flows* immediately subsequent to the visit start. Second, we want to minimize the impact of caching and persistent HTTP connection that allows the same flow to download multiple objects (i.e., when a user scrolls a page, the browser downloads new images and material using the same previously opened TCP connections). For this, we extract features only from the first $n = 10$ packets of a flow to include uniquely the first instants of the communication. In the following, we refer to Related Flows as $f_1, \ldots, f_k$.

We engineer features using the information available in the first n packets of the *Trigger* flow f_0 and *Related Flows* $f_1, \ldots, f_k$. Given a flow f, we denote with $c_{i,f}$ and $s_{i,f}$ the i^{th} packet on the client and server side, respectively. Each packet $c_{i,f}$ (or $s_{i,f}$) is characterized by its size $|c_{i,f}|$, its time $t(c_{i,f})$ and the value of its Time-To-Live field $TTL(c_{i,f})$. From them, we build our features. Specifically, each dataset entry represents a webpage visit and is described by the following features:

- We build two sets including, respectively, the size of all client packets $|c_{i,f}|$ and the size of all server packets $|s_{i,f}|$ for *Trigger* and *Related* flows. Out of each set, we extract the 25^{th}, 50^{th}, 75^{th} and 90^{th} percentiles, as well as the mean, standard deviation, maximum, minimum values, and the occurrences of the missing samples[2].

- We compute the inter-arrival time of each packet as the time elapsed since the previous packet in the same direction. Given $c_{i,f}$ (or $s_{i,f}$), its inter-arrival time is defined as $t(c_{i,f}) - t(c_{i-1,f})$, with $i \geq 2$. We build two sets including all inter-arrival times for client and server packets, respectively. Out of them, we extract the same statistics as for packet sizes.

- For each *Related Flow* f_i, we compute its relative starting time with respect to the *Trigger* as $t(c_{1,f_i}) - t(c_{1,f_0})$ with $i \in \{1 \ldots k\}$

[2]Missing samples can occur if a flow has less then n packets, or if a website opens less than k flow in ΔT

- For each flow f_i, we compute the time between the first server and client packet $t(s_{1,f_i}) - t(c_{1,f_i})$. This time measures, in fact, the Round Trip Time between the operator's ground station and the actual server.

- For each flow, we compute the size of the first flight in each direction. A flight is the total size of the application payload contained in packets sent without any confirmation received from the counterpart. It provides an estimate of Client and Server Hello TLS messages. For the client side, formally, given flow f, we compute it as $\sum_{j=1}^{j_{max}} |c_{j,f}|$ with j_{max} set so that $t(c_{j,f}) < t(s_{1,f})$. For the server side, we compute the same measure specularly.[3]

- For each flow, we extract the Time-To-Live (TTL) value from the IP header of server packets. This is a coarse indication of the server's distance (in number of hops). As the TTL could vary within a single flow, we take the minimum value. For each flow f_i, we extract $\min_{\forall i \in 1...n} TTL(s_{i,f_i})$.

In total, we have 52 features. By design, they are L4-agnostic, meaning that even if the communication is carried over UDP/QUIC, the system continues to operate seamlessly. At last, observe that, once the Starting Point is triggered, the first k Related Flows may include some flows generated by background traffic and applications. This may result in occasional uncorrelated flows that can confuse the classifier. For instance, if the user accesses multiple web pages at the same time, the flows each page generates gets multiplexed in the network, possibly impacting the feature extraction process.

8.3.4 ML Pipeline

To train the ML models, we adopt the typical pipeline for supervised tasks. We formulate the problem as a regression task, the goal being the prediction of the value of the target QoE-related metrics. Given a data point y_i, we aim at building a model $\hat{y}_i = f(\mathbf{x}_i)$, where $\mathbf{x}$ is the array of the input features described in the previous section.

To measure prediction performance, we use two established metrics: R^2 Score and Mean Absolute Percentage Error (MAPE). The R^2 coefficient serves to determine whether a linear regression can be used to describe the target variable. An R^2 score

[3]For the server side, we clearly neglect client packets related to the first flight while computing j_{max}.

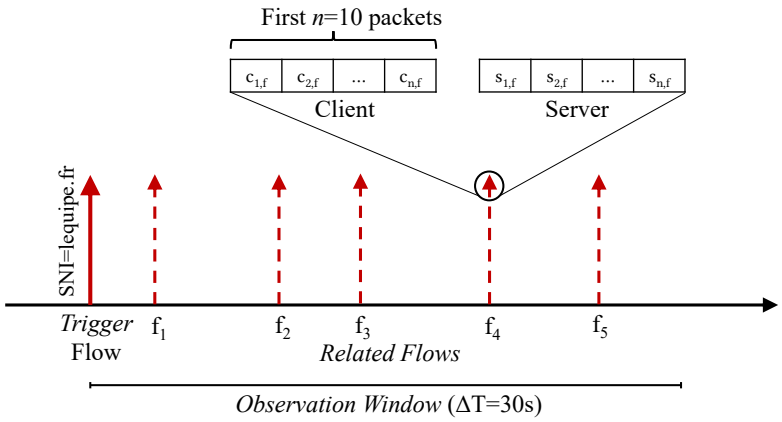


Fig. 8.3 Detection of *Related Flows* and corresponding features.

of 0 signifies a model $\mathbf{x}$ that is not able to predict correctly the actual values (i.e., $\mathbf{x}$ cannot explain y). Conversely, $R^2 = 1$ means $f(\mathbf{x})$ is a perfect predictor. In mathematical terms:

$$R^2 = 1 - \frac{\sum_{i=1}^{z}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{z}(\hat{y}_i - \bar{y})^2}$$

where $\hat{y}_i$ represents the predicted values of the actual sample y_i, $\bar{y}$ being the mean y_i and z the number of data points.

The MAPE is a metric used to assess the accuracy of a predictive model in percentage terms. It measures the average percentage error between the predicted and actual values:

$$MAPE = 100\frac{1}{z}\sum_{i=1}^{z}\left|\frac{\hat{y}_i - y_i}{y_i}\right|$$

We apply a feature selection stage to identify the most relevant features from an initial array $\mathbf{x}$ of 52 variables. This reduces the complexity of the model by discarding either features that are redundant or uncorrelated with the target variable. To tackle this task, we utilize the Recursive Feature Elimination (RFE) algorithm [151] in conjunction with a Random Forest Regressor-based model. This approach involves systematically training the algorithm and iteratively discarding features deemed least important. The output $\mathbf{x}$' is then used to build the model $f(\mathbf{x}')$

We train a specific ML model for each website using the selected features and adopting the standard Stratified K-Fold Cross-Validation methodology to mitigate the risk of overfitting, with 70% and 30% of data used for training and validation. As regression models $f(\mathbf{x}')$, we test Decision Tree Regressor (DTR), Random Forest

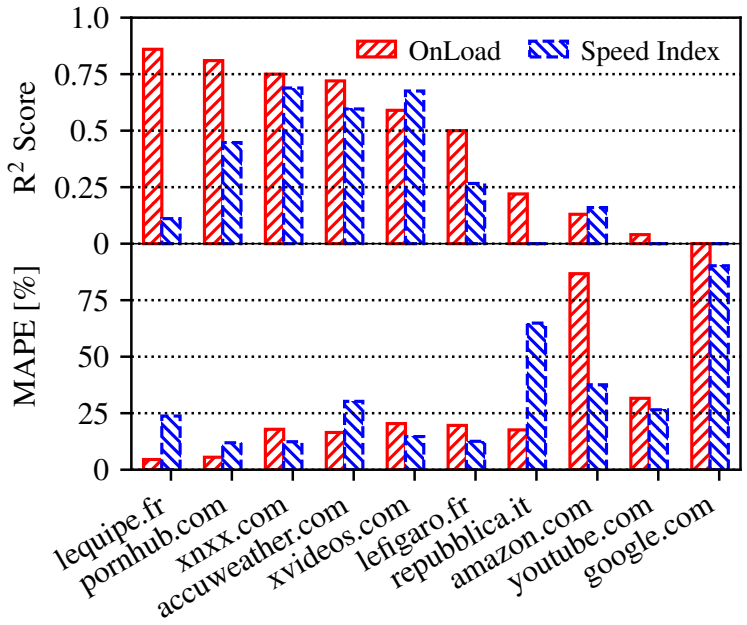


Fig. 8.4 Performance on different websites, measured using R^2 Score and MAPE.

Regressors (RFR), Linear Regressors (LR) and K-Nearest Neighbors Regression (KNNR).

8.4 Experimental Results

In this section, we show and discuss the experimental results, dissecting the regressor performance first in different scenarios and then delving into feature and ML model choice. To determine the most suitable ML model and training strategy, we undertake a systematic exploration of several ML models and alternatives on how to build them.

8.4.1 Per-Website Model Performance

We first consider building a website-specific model, an option suitable for our target deployment where the operator can select the target websites. Here we consider the Random Forest Regressor, which provides the best results (see Sec. 8.4.4).

In Figure 8.4, we show per-website performance in terms of R^2 score (top plot) and MAPE (bottom plot). As expected, values of the R^2 score and MAPE are negatively correlated. Prediction accuracy radically varies across websites, with some exhibiting very good performance while others have unsatisfactory results. For the onLoad, 6 websites have R^2 score > 0.5 and MAPE $< 25\%$, hinting that ML models are capable of providing a reliable prediction for them. However, for the remaining 4 websites, the predictive power of the model is very poor, with an R^2 score below 0.25 or negative (not visible as the scale represents positive values only). For *amazon.com*, *youtube.com* and *google.com*, we link bad prediction to the fact that those websites include objects that are served by a few domains hosted in the same infrastructure owned by the same company. This limits the number and the diversity of related flows that are often below $k = 5$ impacting the set of meaningful features. Conversely, for *repubblica.it* we observe that all pages include a very large number of third-party objects. Some of them are advertisement banners extremely slow to load. This impairs the OnLoad time as observed in Figure 8.2, which exceeds 35 seconds most of the time. This makes the prediction for this website very unreliable. Similar considerations hold for Speed Index, even if prediction accuracy is overall lower – only 3 websites present R3 score above 0.5. This is somewhat expected, as the Speed Index value depends on how the webpage is rendered by the browser [161, 56] and, thus, network traffic has a more indirect impact.

We provide two examples to qualitatively illustrate predictions for a website with a high/poor R^2 score in Figure 8.5. We show the predicted and real values for the OnLoad metric using a scatter plot. Each point represents a different visit, and different colours indicate the density of points. Ideally, in the case of a perfect regressor, all points should lie on the main diagonal. This is what happens for *lequipe.fr* ($R^2 = 0.8$) where points lie on the diagonal stretched between 16 and 30 seconds, hinting that predictions are accurate both for slow and fast visits. Different is the picture for *google.com* ($R^2 < 0$). The model exhibits almost no prediction power, with predictions that are massed in a circular shape centred along 6 seconds regardless of the real visit OnLoad time.

In summary, the different performance indicates that the operator shall carefully select the target websites and test the prediction performance of the regressor before deploying it in operations.

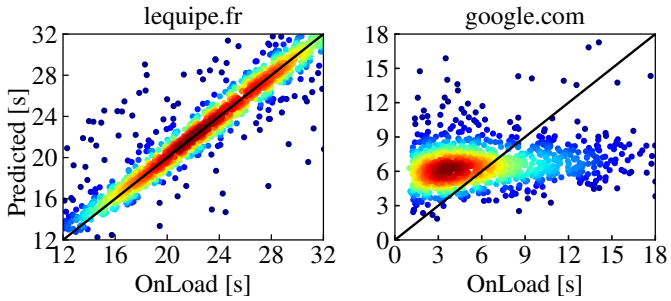


Fig. 8.5 Scaterplots representing predicted and real OnLoad values for two websites.

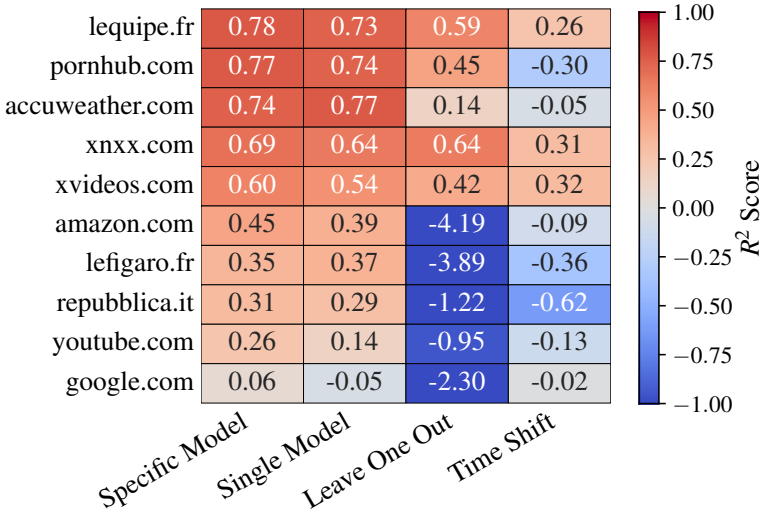


Fig. 8.6 Prediction performance for onLoad in different scenarios.

8.4.2 One vs Many Models

We now discuss the possibility of using a single model to predict the QoE for multiple websites.

First, we consider the case of building a single model trained on all websites. The intuition is to create a model that generalizes the complexities and nuances present across multiple domains, resulting in a more robust and adaptable solution. For this, we create a single set containing all points for all websites. We use it for training (70%) and testing (30% split) a *single* model. We normalize the target metrics in a website-wise fashion to obtain values in the same order of magnitude. In Figure 8.6, we compare the R^2 score we obtain for different scenarios. The first

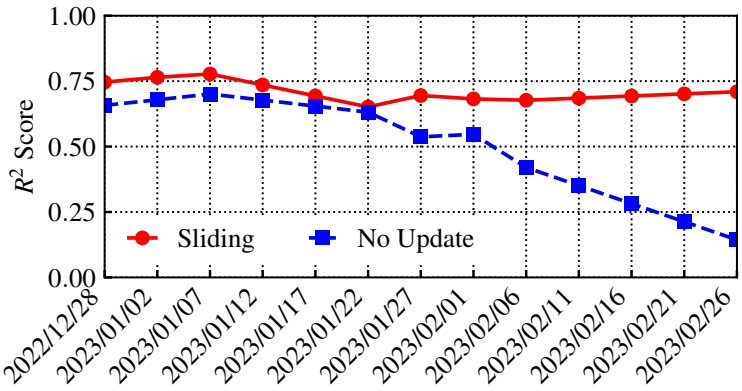


Fig. 8.7 Prediction performance for OnLoad of *pornhub.com* with different training strategies.

two columns compare the prediction performance with a Specific Model for each website (first column) with the performance of the Single Model (second column). Overall, prediction accuracy decreases with the R^2 score that never decreases more than 0.2. For some websites, the drop is negligible (see for example *pornhub.com*). Again, some websites present awful performance in general

We now evaluate the scenario in which the ML model is used to predict the QoE for websites unseen at training time. The intuition is to have a global single model that works for any website. To this end, we adopt a leave-one-out approach: we train a model using all data from all websites except the one under consideration for testing, i.e., systematically excluding data about one target domain at a time during the training phase. The resulting performance is depicted in the third column of Figure 8.6. It shows that almost no prediction capability is offered in such a scenario. In most cases, the R^2 score is negative, and above 0.5 only in two cases. This clearly demonstrates that we need to include samples during training for the specific target website the operator wants to monitor. In a nutshell, it is not possible to generalize a single model applicable to any website. This reflects the specificity of the traffic generated by each website and underlines the need to collect site-specific data for training.

We conclude that the best results are obtained by considering a site-specific model. It would be possible to train a single model on multiple websites, even with a moderate penalty. However, it is not possible to train a generic model to predict QuE of a website unseen at training time.

8.4.3 Temporal Stability

We now evaluate the impact of training data and model freshness. We want to quantify to what extent a model trained on a dataset collected at a given point in time can correctly predict the QoE later in time. To this end, we use the data collected on September 2022 to train models, while we use data from February 2023 to evaluate their prediction performance. We use a Random Forest regressor trained for each website. We show results in the last column of Figure 8.6. With few exceptions, the predictive capabilities of the models are completely lost. For instance, the *lefigaro.com* R^2 drops from 0.78 to 0.26. For *xnxx.com* and *xvideos.com*, some prediction capability survives after three months. In the other cases, the prediction power is completely lost, with negative R^2 scores. This is due to changes in the website aspect, in the infrastructure that serves them, or/and in the network properties. For instance, manual inspection of available snapshots on the Wayback Machine[4] reveals that some websites incurred a graphical restyle in the last weeks of 2022. This clearly makes the previously collected training set totally useless.

We simulate such a system using the data in the December 2022 - February 2023 dataset. In Figure 8.7, we show the evolution over time of the R^2 score for *pornhub.com*. The blue dashed line reports performance when using always the same model trained using only the first week of data (the penultimate week of December 2022), and then testing its performance on all visits occurring in each period of 5 days. Clearly, the frozen model becomes quickly obsolete over time. From the beginning of 2022, its prediction power vanishes. In a nutshell, the regression model is becoming too outdated. We observe the same behaviour for all websites: in the long term, any model must be updated.

To overcome the model ageing, we propose to use the *walk-forward* approach typically used for time series prediction. Here, we assume test agents continuously collect data. For each target website, at the end of a period (e.g., a few days), we use the data collected from the past w days to train a new model. We follow a *sliding-windows* approach, i.e., we keep constant the size of the training set. We then use this freshly trained model to predict the performance during the next period of time.

We simulate a system that implements the walk-forward policy previously described. We show results with the red solid line in Figure 8.7. Here, we consider a

[4] https://web.archive.org/

Table 8.1 List of 25 most important features according to RFE.

Measure	Direction	Statistics
Packet Inter-arrival Time	Client	Mean, Standard Deviation, Maximum, #(Zero Value), Percentiles 25, 50, 75, 90
Packet Inter-arrival Time	Server	Standard Deviation, Maximum, Percentiles 50, 75
Time-To-Live	Server	Minimum value of the first 3 flows
Time between first Client and Server packet	-	First 5 flows
Relative time of Related Flows	-	All Related Flows

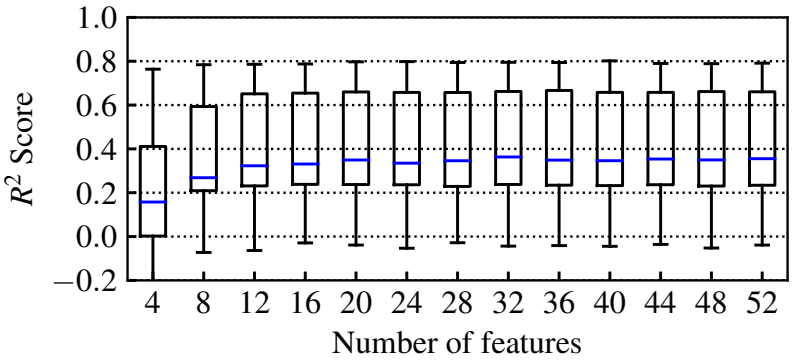


Fig. 8.8 Prediction performance for OnLoad with a different number of features.

period of w equal to 5 days for training a model. We then evaluate its performance for the following 5 days. Since we use a model that is trained always on the most recent dataset, the R^2 score remains stable at around 0.75.

We conclude that it is mandatory to continuously operate Test Agents to ensure the freshness of the training data which results the key to maintaining model performance consistent over time.

8.4.4 Feature and Algorithm Impact

We finally discuss the impact of the feature selection and the choice of regression algorithm. As detailed in Section 8.3.4, we adopt the RFE feature selection method to find the most meaningful features among the 52 we extract. In Figure 8.8, we show how R^2 Score varies with models that consider only the top-k ranked features for the onLoad prediction. Boxplots represent the distribution across the 10 websites. We observe that the median R^2 score significantly increases up to 20 features. Including more than 25 features brings negligible improvement.

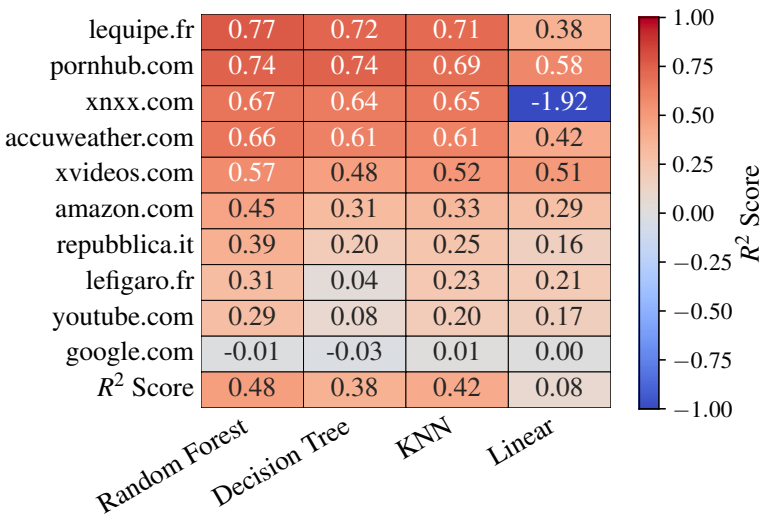


Fig. 8.9 Comparision of prediction performance for onLoad, using one model per site.

Looking at which are the most relevant features, we observe those related to packet inter-arrival time and other flow timings (relative starting time in particular), as well as some on the server Time-To-Live. For completeness, we list in Table 8.1 the set of 25 features that we use in all experiments.

We finally compare the performance of different classification algorithms, as anticipated in Section 8.3. For each algorithm, we run a hyper-parameter tuning step using a coarse grid search. A Random Forest Regressor with 100 estimators provides the best performance in all cases, for both OnLoad and SpeedIndex and for both the R^2 and MAPE performance metrics. In Figure 8.9, we detail the average per-site R^2 scores of all algorithms in predicting the OnLoad. The last row presents the column-wise average. As said, the Random Forest model provides the best performance for all websites, followed by KNN. However, the latter entails large models as they must contain the entire training set. Thus, we believe the Random Forest model represents the best choice. A Decision Tree provides modest performance for all websites, while the Linear regressors perform largely worse than other methods, hinting they are not well-suited for this kind of problem.

At last, we test the performance of Neural Network (NN) models. However, the need for large amounts and always recent data limits their applicability in this scenario. Our dataset in fact resulted too small to let the NN converge. In practice,

the cost of obtaining thousands of samples for each target website increases the cost of data collection with a limited payoff.

8.5 Takeaways

In this chapter, we have explored the challenges and successes of designing a Web Quality of Experience (QoE) estimation system for satellite communications (SatCom). Through extensive experimentation and analysis, several critical insights have been gained. These insights not only reflect the current state of our research but also chart the course for future explorations in this domain. Key findings from this study are summarized as follows:

- **Monitoring System Design and Implementation**: Successfully designed and implemented a system for estimating Web Quality of Experience (QoE) for satellite communication (SatCom) users, targeting GEO operators using passive measurements.
- **Role and Challenges of PEPs**: Performance Enhancing Proxies (PEPs) are crucial in SatCom networks for TCP performance enhancement but introduce complexities in QoE measurement due to the splitting of TCP flows.
- **Machine Learning for QoE Estimation**: The application of machine learning, particularly random forest regressors, proved effective in mapping flow-level and packet-level features to QoE metrics, utilizing data from Test Agents.
- **Web Content Dynamics**: The variability and architectural differences across websites hinder the development of a general model for predicting performance on new, unseen websites.
- **Continuous Model Training Necessity**: Regular model updates and training are essential due to the dynamic nature of web content and infrastructure, with the walk-forward approach being crucial for maintaining model relevance.
- **Future Research Directions**: Expanding the scope to include a broader range of websites and considering user habit diversity, subscriber setup, and backbone connectivity are vital. Further exploration into transfer learning and

domain adaptation is needed to enhance model generalization across different regions.

This chapter demonstrates the feasibility of using machine learning to predict Web QoE in SatCom scenarios and emphasizes the continuous adaptation and refinement needed for the models in response to the dynamic web environment.

Chapter 9

Conclusions

The research presented in this thesis revolves around the measurement of network traffic utilizing both active and passive techniques (Chapter 3, Chapter 4, and Chapter 5). Additionally, it encompasses the development of Machine Learning models that leverage these measurements as features, enabling the prediction of QoS and QoE metrics (Chapter 7 and Chapter 8).

The case study is founded on a diverse array of traffic protocols, ranging from video-conferencing applications employing RTP to the utilization of HTTP/3 with QUIC, as well as more generalized protocols such as DNS, TCP or UDP traffic consumption. A comprehensive overview of the network protocols and mechanisms employed by these applications is presented, with a prominent emphasis on RTP and QUIC.

This thesis also provides guidance on establishing a robust testbed for measurement purposes, including a discussion on the most valuable open-source software tools to facilitate this task. Additionally, a significant portion of the code and data collected over the course of the study is shared (see Table 1.1).

9.1 Machine Learning for Networks: Personal Considerations

We are witnessing the true potential of Machine Learning unfolding before our eyes. A prime example is the advent of Large Language Models (LLM), exemplified by

ChatGPT[1] and Google Bard[2], which revolutionized the market in early 2022. Only a couple of years ago, the idea of such advances might have seemed plausible, but with predictions of much longer development times.

Artificial intelligence has indeed made significant inroads into the market: many companies have shifted their operations towards utilizing these tools to enhance productivity, while others have even made artificial intelligence the cornerstone of their business strategies. Large Language Models, in particular, rely heavily on vast amounts of data to function optimally. The internet, with its extensive reservoir of textual, visual, and multimedia content, has provided an ideal environment for their evolution by providing a rich data pool. However, it's essential to recognize that AI requires not only data but also substantial computing power for training. In recent years, the surge in GPU capabilities has been propelled in tandem with the remarkable growth of the gaming industry, likely fueled by the cryptocurrency mining boom as well. Consequently, demand for GPUs has skyrocketed to unprecedented levels, driven by both gamers seeking high-performance hardware and cryptocurrency miners leveraging GPUs for their computationally intensive tasks. This convergence of factors has provided fertile ground for the rapid growth and flourishing of AI, thanks to the investment received in those sectors.

At this point, we must address a crucial question: what drives the convergence of AI and networking? Machine Learning is often used to identify patterns in data, with the aim of understanding complex structures through examples. Network protocols, being a set of standardised rules, naturally lend themselves to the application of AI. While our focus has been on the abundance of internet data, it's vital to acknowledge that its effective use hinges on network infrastructure traversal. This involves two key aspects: monitoring network traffic consumption and developing scalable, energy-efficient, and secure infrastructures. Network protocols are essential for transmitting data online, forming the backbone of the internet. They structure data into packets for transmission, highlighting the intrinsic link between data and protocols. These protocols act as containers for data, ensuring its efficient exchange. This dualism between protocol and data is the basis of the Internet. In other words, data on the Internet cannot exist without network protocols, just as network protocols cannot exist without the need to exchange data. Optimizing network performance, such as reducing energy consumption and minimizing delays, leads to significant economic

[1] https://chat.openai.com/
[2] https://bard.google.com/chat?hl=it

and environmental benefits due to the internet's vast scale. Hence, every enhancement contributes to substantial impacts in financial and environmental realms, making this topic pertinent and intriguing.

Looking to the future, envisioning a world where Artificial Intelligence (AI) becomes increasingly intertwined with our daily lives, it's evident that significant changes are on the horizon across multiple domains that currently seem commonplace. In this evolving landscape, researchers are tasked not only with the development of AI models but also with the responsibility of fostering a deeper understanding of these technologies within society. Our role extends beyond mere innovation; it involves disseminating knowledge about the capabilities and safety measures associated with AI, thereby promoting an ethical and secure advancement of these tools. It is imperative that we engage in discussions surrounding the implications of AI technologies, ensuring that their integration into our lives is guided by ethical principles and societal well-being. Through collaborative efforts and transparent communication, we can pave the way for a future where AI enriches our lives while safeguarding our values and security.

References

[1] Vijay K Adhikari, Yang Guo, Fang Hao, Volker Hilt, Zhi-Li Zhang, Matteo Varvello, and Moritz Steiner. Measurement study of netflix, hulu, and a tale of three cdns. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 23(6), 2015.

[2] Trinh Viet Doan, Vaibhav Bajpai, and Sam Crawford. A longitudinal view of netflix: Content delivery over ipv6 and content cache deployments. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 1073–1082. IEEE, 2020.

[3] Ron Frederick, Stephen L. Casner, Van Jacobson, and Henning Schulzrinne. RTP: A Transport Protocol for Real-Time Applications. RFC 1889, January 1996.

[4] Jonathan Rosenberg, Christian Huitema, Rohan Mahy, and Joel Weinberger. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489, March 2003.

[5] Philip Matthews, Jonathan Rosenberg, and Rohan Mahy. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766, April 2010.

[6] Jonathan Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245, April 2010.

[7] Marc Petit-Huguenin and Gonzalo Salgueiro. Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS). RFC 7983, 2016.

[8] Christer Holmberg, Stefan Hakansson, and Goran Eriksson. Web Real-Time Communication Use Cases and Requirements. RFC 7478, 2015.

[9] Magnus Westerlund and Stephan Wenger. RTP Topologies. RFC 7667, November 2015.

[10] Roy T. Fielding, Henrik Nielsen, Jeffrey Mogul, Jim Gettys, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068, January 1997.

[11] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, May 2015.

[12] Matteo Varvello, Kyle Schomp, David Naylor, Jeremy Blackburn, Alessandro Finamore, and Konstantina Papagiannaki. Is the web http/2 yet? In *International Conference on Passive and Active Network Measurement*, pages 218–232. Springer, 2016.

[13] Eric Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446, August 2018.

[14] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio, and Danilo Giordano. A network analysis on cloud gaming: Stadia, geforce now and psnow. *Network*, 1(3):247–260, 2021.

[15] Ryan Shea, Jiangchuan Liu, Edith C-H Ngai, and Yong Cui. Cloud gaming: architecture and performance. *IEEE network*, 27(4):16–21, 2013.

[16] Chun-Ying Huang, Cheng-Hsin Hsu, Yu-Chun Chang, and Kuan-Ta Chen. Gaminganywhere: an open cloud gaming system. In *Proceedings of the 4th ACM multimedia systems conference*, pages 36–47, 2013.

[17] Henning Schulzrinne, Stephen Casner, Ron Frederick, and Van Jacobson. Rtp: A transport protocol for real-time applications. Technical report, 2003.

[18] Branislav Sredojev, Dragan Samardzija, and Dragan Posarac. Webrtc technology overview and signaling solution design and implementation. In *2015 38th international convention on information and communication technology, electronics and microelectronics (MIPRO)*, pages 1006–1009. IEEE, 2015.

[19] Martino Trevisan, Ali Safari Khatouni, and Danilo Giordano. Errant: Realistic emulation of radio access networks. *Computer Networks*, 176:107289, 2020.

[20] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio, and Danilo Giordano. A network analysis on cloud gaming: Stadia, GeForce Now and PSNow. Zenodo, 2021.

[21] Marc Carrascosa and Boris Bellalta. Cloud-gaming: Analysis of google stadia traffic. *Computer Communications*, 188:99–116, 2022.

[22] Philippe Graff, Xavier Marchal, Thibault Cholez, Stéphane Tuffin, Bertrand Mathieu, and Olivier Festor. An analysis of cloud gaming platforms behavior under different network constraints. In *2021 17th International Conference on Network and Service Management (CNSM)*, pages 551–557. IEEE, 2021.

[23] Debargha Mukherjee, Jim Bankoski, Adrian Grange, Jingning Han, John Koleszar, Paul Wilkins, Yaowu Xu, and Ronald Bultje. The latest open-source video codec vp9-an overview and preliminary results. In *2013 Picture Coding Symposium (PCS)*, pages 390–393. IEEE, 2013.

[24] Wei Cai, Ryan Shea, Chun-Ying Huang, Kuan-Ta Chen, Jiangchuan Liu, Victor CM Leung, and Cheng-Hsin Hsu. A survey on cloud gaming: Future of computer games. *IEEE Access*, 4:7605–7620, 2016.

[25] Hua-Jun Hong, De-Yu Chen, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. Placing virtual machines to optimize cloud gaming experience. *IEEE Transactions on Cloud Computing*, 3(1):42–53, 2014.

[26] Arto Ojala and Pasi Tyrvainen. Developing cloud business models: A case study on cloud gaming. *IEEE software*, 28(4):42–47, 2011.

[27] Florian Metzger, Stefan Geißler, Alexej Grigorjew, Frank Loh, Christian Moldovan, Michael Seufert, and Tobias Hoßfeld. An introduction to online video game qos and qoe influencing factors. *IEEE Communications Surveys & Tutorials*, 24(3):1894–1925, 2022.

[28] Yanjiao Chen, Kaishun Wu, and Qian Zhang. From qos to qoe: A tutorial on video quality assessment. *IEEE Communications Surveys & Tutorials*, 17(2):1126–1165, 2014.

[29] Zili Meng, Yaning Guo, Chen Sun, Bo Wang, Justine Sherry, Hongqiang Harry Liu, and Mingwei Xu. Achieving consistent low latency for wireless real-time communications with the shortest control loop. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 193–206, 2022.

[30] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. Pruning edge research with latency shears. In *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*, HotNets '20, page 182–189, New York, NY, USA, 2020. Association for Computing Machinery.

[31] Saeed Shafiee Sabet, Steven Schmidt, Saman Zadtootaghaj, Babak Naderi, Carsten Griwodz, and Sebastian Möller. A latency compensation technique based on game characteristics to mitigate the influence of delay on cloud gaming quality of experience. In *Proceedings of the 11th ACM Multimedia Systems Conference*, pages 15–25, 2020.

[32] Kuan-Ta Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang, and Chin-Laung Lei. Measuring the latency of cloud gaming systems. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 1269–1272, 2011.

[33] Mark Claypool and David Finkel. The effects of latency on player performance in cloud-based games. In *2014 13th Annual Workshop on Network and Systems Support for Games*, pages 1–6. IEEE, 2014.

[34] Xavier Marchal, Philippe Graff, Joël Roman Ky, Thibault Cholez, Stéphane Tuffin, Bertrand Mathieu, and Olivier Festor. An analysis of cloud gaming platforms behaviour under synthetic network constraints and real cellular

networks conditions. *Journal of Network and Systems Management*, 31(2):39, 2023.

[35] Philippe Graff, Xavier Marchal, Thibault Cholez, Bertrand Mathieu, and Olivier Festor. Efficient identification of cloud gaming traffic at the edge. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–10, 2023.

[36] Joël Roman Ky, Bertrand Mathieu, Abdelkader Lahmadi, and Raouf Boutaba. Assessing unsupervised machine learning solutions for anomaly detection in cloud gaming sessions. In *2022 18th International Conference on Network and Service Management (CNSM)*, pages 367–373, 2022.

[37] M. Trevisan, A. S. Khatouni, and D. Giordano. ERRANT: Realistic emulation of radio access networks. *Computer Networks*, 176:107289, 2020.

[38] Martino Trevisan, Alessandro Finamore, Marco Mellia, Maurizio Munafo, and Dario Rossi. Traffic analysis with off-the-shelf hardware: Challenges and lessons learned. *IEEE Communications Magazine*, 55(3):163–169, 2017.

[39] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. Rfc, RFC Editor, 07 2003.

[40] Carsten Burmeister, Jose Rey, Noriyuki Sato, Joerg Ott, and Stephan Wenger. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF). RFC 4585, 2006.

[41] Luca Vassio, Hassan Metwalley, and Danilo Giordano. The exploitation of web navigation data: Ethical issues and alternative scenarios. In *Blurring the Boundaries Through Digital Innovation*, pages 119–129. Springer, 2016.

[42] Gianluca Perna, Martino Trevisan, Danilo Giordano, and Idilio Drago. A first look at http/3 adoption and performance. *Computer Communications*, 187:115–124, 2022.

[43] Mike Bishop. Hypertext Transfer Protocol Version 3 (HTTP/3). Internet-Draft draft-ietf-quic-http-34, Internet Engineering Task Force, 2021. Work in Progress.

[44] J. Iyengar and M. Thomson. Quic: A udp-based multiplexed and secure transport. RFC 9000, RFC Editor, May 2021.

[45] Darius Saif, Chung-Horng Lung, and Ashraf Matrawy. An early benchmark of quality of experience between http/2 and http/3 using lighthouse. *arXiv preprint arXiv:2004.01978*, 2020.

[46] Robin Marx, Joris Herbots, Wim Lamotte, and Peter Quax. Same standards, different decisions: A study of quic and http/3 implementation diversity. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, pages 14–20, 2020.

[47] Sreeni Tellakula. Comparing HTTP/3 vs. HTTP/2 Performance. https://blog.cloudflare.com/http-3-vs-http-2/, April 2020.

[48] Luis Guillen, Satoru Izumi, Toru Abe, and Takuo Suganuma. Sand/3: Sdn-assisted novel qoe control method for dynamic adaptive streaming over http/3. *Electronics*, 8(8):864, 2019.

[49] Darius Saif and Ashraf Matrawy. A pure http/3 alternative to mqtt-over-quic in resource-constrained iot. *arXiv preprint arXiv:2106.12684*, 2021.

[50] Dominic Lovell, Barry Pollard, Robin Marx, and Shaina Hantsis. Part IV Chapter 24, HTTP), 2021.

[51] Konrad Wolsing, Jan Rüth, Klaus Wehrle, and Oliver Hohlfeld. A performance perspective on web optimized protocol stacks: Tcp+ tls+ http/2 vs. quic. In *Proceedings of the Applied Networking Research Workshop*, pages 1–7, 2019.

[52] Jawad Manzoor, Llorenç Cerdà-Alabern, Ramin Sadre, and Idilio Drago. On the performance of quic over wireless mesh networks. *Journal of Network and Systems Management*, 28(4):1872–1901, 2020.

[53] Gaetano Carlucci, Luca De Cicco, and Saverio Mascolo. Http over udp: An experimental investigation of quic. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, page 609–614, New York, NY, USA, 2015. Association for Computing Machinery.

[54] Arash Molavi Kakhki, Samuel Jero, David Choffnes, Cristina Nita-Rotaru, and Alan Mislove. Taking a long look at quic: an approach for rigorous evaluation of rapidly evolving transport protocols. In *Proceedings of the 2017 Internet Measurement Conference*, pages 290–303, 2017.

[55] Mike Kosek, Tanya Shreedhar, and Vaibhav Bajpai. Beyond quic v1–a first look at recent transport layer ietf standardization efforts. *arXiv preprint arXiv:2102.07527*, 2021.

[56] Diego Neves da Hora, Alemnew Sheferaw Asrese, Vassilis Christophides, Renata Teixeira, and Dario Rossi. Narrowing the gap between qos metrics and web qoe using above-the-fold metrics. In *International Conference on Passive and Active Network Measurement*, pages 31–43. Springer, 2018.

[57] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. A survey on quality of experience of http adaptive streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492, 2014.

[58] Thiago Guarnieri, Idilio Drago, Alex B. Vieira, Italo Cunha, and Jussara Almeida. Characterizing qoe in large-scale live streaming. In *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, pages 1–7, 2017.

[59] Daniel Perdices, Gianluca Perna, Martino Trevisan, Danilo Giordano, and Marco Mellia. When satellite is all you have: Watching the internet from 550 ms. IMC '22, page 137–150, New York, NY, USA, 2022. Association for Computing Machinery.

[60] José Luis García-Dorado, Alessandro Finamore, Marco Mellia, Michela Meo, and Maurizio Munafo. Characterization of isp traffic: Trends, user habits, and access technology impact. *IEEE Transactions on Network and Service Management*, 9(2):142–155, 2012.

[61] Matthew Sargent and Mark Allman. Performance within a fiber-to-the-home network. *ACM SIGCOMM Computer Communication Review*, 44(3), 2014.

[62] Dongzhu Xu, Anfu Zhou, Xinyu Zhang, Guixian Wang, Xi Liu, Congkai An, Yiming Shi, Liang Liu, and Huadong Ma. Understanding operational 5g: A first measurement study on its coverage, performance and energy consumption. 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*, pages 479–494, 2020.

[63] Arvind Narayanan, Eman Ramadan, Jason Carpenter, Qingxu Liu, Yu Liu, Feng Qian, and Zhi-Li Zhang. A first look at commercial 5g performance on smartphones. In *Proceedings of The Web Conference 2020*, pages 894–905, 2020.

[64] Martino Trevisan, Danilo Giordano, Idilio Drago, Maurizio Matteo Munafò, and Marco Mellia. Five years at the edge: Watching internet from the isp network. *IEEE/ACM Transactions on Networking*, 28(2):561–574, 2020.

[65] Thomas Koch, Ethan Katz-Bassett, John Heidemann, Matt Calder, Calvin Ardi, and Ke Li. Anycast in context: A tale of two systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, SIGCOMM '21, page 398–417, New York, NY, USA, 2021. Association for Computing Machinery.

[66] (how much) can edge computing change network latency? In *2021 IFIP Networking Conference (IFIP Networking)*, pages 1–9, 2021.

[67] Yurong Hu and V.O.K. Li. Satellite-based internet: a tutorial. *IEEE Communications Magazine*, 39(3):154–162, 2001.

[68] Oltjon Kodheli, Eva Lagunas, Nicola Maturo, Shree Krishna Sharma, Bhavani Shankar, Jesus Fabian Mendoza Montoya, Juan Carlos Merlano Duncan, Danilo Spano, Symeon Chatzinotas, Steven Kisseleff, et al. Satellite communications in the new space era: A survey and future challenges. *IEEE Communications Surveys & Tutorials*, 23(1):70–109, 2020.

[69] J Border. Enhancing proxies intended to mitigate link-related degradations. *RFC 3135*, 2001.

[70] J. Zhu, S. Roy, and J.H. Kim. Performance modelling of tcp enhancements in terrestrial–satellite hybrid networks. *IEEE/ACM Transactions on Networking*, 14(4):753–766, 2006.

[71] Alain Pirovano and Fabien Garcia. A new survey on improving tcp performances over geostationary satellite link. *Network and Communication Technologies*, 2(1):1, 2013.

[72] François Michel, Martino Trevisan, Danilo Giordano, and Olivier Bonaventure. A first look at starlink performance. In *Proceedings of the 2022 Internet Measurement Conference*, 2022.

[73] Chuck Fraleigh, Sue Moon, Bryan Lyles, Chase Cotton, Mujahid Khan, Deb Moll, Robert Rockell, Ted Seely, and S Christophe Diot. Packet-level traffic measurements from the sprint ip backbone. *IEEE network*, 17(6):6–16, 2003.

[74] Marina Fomenkov, Ken Keys, David Moore, and KC Claffy. Longitudinal study of internet traffic in 1998-2003. In *Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6, 2004.

[75] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 90–102, 2009.

[76] Zachary S Bischof, Fabián E Bustamante, and Rade Stanojevic. Need, want, can afford: Broadband markets and the behavior of users. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 73–86, 2014.

[77] Pierre Borgnat, Guillaume Dewaele, Kensuke Fukuda, Patrice Abry, and Kenjiro Cho. Seven years and one day: Sketching the evolution of internet traffic. In *IEEE INFOCOM 2009*, pages 711–719. IEEE, 2009.

[78] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. *ACM SIGCOMM Computer Communication Review*, 40(4):75–86, 2010.

[79] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Enric Pujol, Ingmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, Oliver Hohlfeld, and Georgios Smaragdakis. The lockdown effect: Implications of the covid-19 pandemic on internet traffic. In *Proceedings of the ACM Internet Measurement Conference*, IMC '20, page 1–18, New York, NY, USA, 2020. Association for Computing Machinery.

[80] Thomas Favale, Francesca Soro, Martino Trevisan, Idilio Drago, and Marco Mellia. Campus traffic and e-Learning during COVID-19 pandemic. *Computer Networks*, 176:107290, 2020.

[81] Timm Böttger, Ghida Ibrahim, and Ben Vallis. How the internet reacted to covid-19: A perspective from facebook's edge network. In *Proceedings of the ACM Internet Measurement Conference*, pages 34–41, 2020.

[82] David L Johnson, Veljko Pejovic, Elizabeth M Belding, and Gertjan Van Stam. Traffic characterization and internet usage in rural africa. In *Proceedings of the 20th international conference companion on World wide web*, pages 493–502, 2011.

[83] Daniel Minoli. *Innovations in satellite communications and satellite technology: the industry implications of DVB-S2X, high throughput satellites, Ultra HD, M2M, and IP*. John Wiley & Sons, 2015.

[84] Joerg Deutschmann, Thomas Heyn, Christian Rohde, Kai-Steffen Hielscher, and Reinhard German. Broadband internet access via satellite: State-of-the-art and future directions. In *Broadband Coverage in Germany; 15th ITG-Symposium*, 2021.

[85] M Tropea and P Fazio. Evaluation of tcp versions over geo satellite links. In *2013 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*, pages 86–90. IEEE, 2013.

[86] Fei Peng, Ángel Salamanca Cardona, Kaveh Shafiee, and Victor CM Leung. Tcp performance evaluation over geo and leo satellite links between performance enhancement proxies. In *2012 IEEE Vehicular Technology Conference (VTC Fall)*, 2012.

[87] Muhammad Muhammad, Matteo Berioli, and Tomaso De Cola. A simulation study of network-coding-enhanced pep for tcp flows in geo satellite networks. In *2014 IEEE International Conference on Communications (ICC)*, pages 3588–3593. IEEE, 2014.

[88] A Abdelsalam, Michele Luglio, Mattia Quadrini, Cesare Roseti, and Francesco Zampognaro. Quic-proxy based architecture for satellite communication to enhance a 5g scenario. In *2019 International Symposium on Networks, Computers and Communications (ISNCC)*, pages 1–6. IEEE, 2019.

[89] Ludovic Thomas, Emmanuel Dubois, Nicolas Kuhn, and Emmanuel Lochin. Google quic performance over a public satcom access. *International Journal of Satellite Communications and Networking*, 37(6):601–611, 2019.

[90] Nicolas Kuhn, François Michel, Ludovic Thomas, Emmanuel Dubois, and Emmanuel Lochin. Quic: Opportunities and threats in satcom. In *2020 10th Advanced Satellite Multimedia Systems Conference and the 16th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, pages 1–7. IEEE, 2020.

[91] Jörg Deutschmann, Kai-Steffen Hielscher, and Reinhard German. Satellite internet performance measurements. In *2019 International Conference on Networked Systems (NetSys)*, pages 1–4. IEEE, 2019.

[92] Aravindh Raman, Matteo Varvello, Hyunseok Chang, Nishanth Sastry, and Yasir Zaki. Dissecting the performance of satellite network operators. *Proceedings of the ACM on Networking*, 1(CoNEXT3):1–25, 2023.

[93] Mohamed M. Kassem, Aravindh Raman, Diego Perino, and Nishanth Sastry. A browser-side view of starlink connectivity. In *Proceedings of the 2022 Internet Measurement Conference*, 2022.

[94] Jim Griner, John Border, Markku Kojo, Zach D. Shelby, and Gabriel Montenegro. Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations. RFC 3135, June 2001.

[95] DPDK Intel. Data plane development kit, 2014.

[96] Jinliang Fan, Jun Xu, and Mostafa H Ammar. Crypto-pan: Cryptography-based prefix-preserving anonymization. *Computer Networks*, 46(2), 2004.

[97] Luca Schumann, Trinh Viet Doan, Tanya Shreedhar, Ricky Mok, and Vaibhav Bajpai. Impact of evolving protocols and covid-19 on internet traffic shares. *arXiv preprint arXiv:2201.00142*, 2022.

[98] Christoph Stork, Enrico Calandro, and Alison Gillwald. Internet going mobile: internet access and use in 11 african countries. *info*, 2013.

[99] Samuel Maredi Mojapelo. The internet access and use in public libraries in limpopo province, south africa. *Public Library Quarterly*, 39(3):265–282, 2020.

[100] Laura Silver and Courtney Johnson. Internet connectivity seen as having positive impact on life in sub-saharan africa. 2018.

[101] Digital TV Research. Africa svod forecasts. https://digitaltvresearch.com/product/africa-svod-forecasts/, 2022.

[102] Conviva. Conviva's state of streaming. https://pages.conviva.com/rs/138-XJA-134/images/RPT_Conviva_State_of_Streaming_Q3_2021.pdf, 2021.

[103] Gianluca Perna, Dena Markudova, Martino Trevisan, Paolo Garza, Michela Meo, and Maurizio M Munafò. Retina: An open-source tool for flexible analysis of rtc traffic. *Computer Networks*, 202:108637, 2022.

[104] Antonio Nistico, Dena Markudova, Martino Trevisan, Michela Meo, and Giovanna Carofiglio. A comparative study of rtc applications. In *2020 IEEE International Symposium on Multimedia (ISM)*, pages 1–8. IEEE, 2020.

[105] Rick Hofstede, Pavel Čeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*, 16(4):2037–2064, 2014.

[106] Benoît Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, October 2004.

[107] Luca Deri and NETikos SpA. nprobe: an open source netflow probe for gigabit networks. In *TERENA Networking Conference*, pages 1–4, 2003.

[108] Eric Rescorla and Nagendra Modadugu. Datagram Transport Layer Security. RFC 4347, April 2006.

[109] Andrea Di Domenico, Gianluca Perna, Martino Trevisan, Luca Vassio, and Danilo Giordano. A network analysis on cloud gaming: Stadia, GeForce Now and PSNow, 2021.

[110] Gianluca Perna, Dena Markudova, Martino Trevisan, Paolo Garza, Michela Meo, Maurizio M Munafò, and Giovanna Carofiglio. Online classification of rtc traffic. In *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–6. IEEE, 2021.

[111] Gianluca Perna, Dena Markudova, Martino Trevisan, Paolo Garza, Michela Meo, Maurizio M Munafò, and Giovanna Carofiglio. Real-time classification of real-time communications. *IEEE Transactions on Network and Service Management*, 2022.

[112] Dena Markudova, Martino Trevisan, Paolo Garza, Michela Meo, Maurizio M Munafo, and Giovanna Carofiglio. What's my App?: ML-based classification of RTC applications. *ACM SIGMETRICS Performance Evaluation Review*, 48(4):41–44, 2021.

[113] M. Trevisan, D. Giordano, I. Drago, M. M. Munafò, and M. Mellia. Five years at the edge: Watching internet from the isp network. *IEEE/ACM Trans. on Networking*, 28(2):561–574, 2020.

[114] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. The cost of the" s" in https. In *Proc. of the 10th ACM International on Conf. on emerging Networking Experiments and Technologies*, pages 133–140, 2014.

[115] M. Finsterbusch, C. Richter, E. Rocha, J. Muller, and K. Hanssgen. A survey of payload-based traffic classification approaches. *IEEE Communications Surveys Tutorials*, 16(2):1135–1156, 2014.

[116] E. Baştuğ, M. Bennis, and M. Debbah. A transfer learning approach for cache-enabled wireless networks. In *2015 13th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pages 161–166, 2015.

[117] Z. Xu, D. Yang, J. Tang, Y. Tang, T. Yuan, Y. Wang, and G. Xue. An actor-critic-based transfer learning framework for experience-driven networking. *IEEE/ACM Transactions on Networking*, 29(1):360–371, 2021.

[118] Selim Ickin, Markus Fiedler, and Konstantinos Vandikas. Customized video qoe estimation with algorithm-agnostic transfer learning. *arXiv preprint arXiv:2003.08730*, 2020.

[119] Y. Hao, J. Yang, M. Chen, M. S. Hossain, and M. F. Alhamid. Emotion-aware video qoe assessment via transfer learning. *IEEE MultiMedia*, 26(1):31–40, 2019.

[120] Baochen Sun, Jiashi Feng, and Kate Saenko. Correlation alignment for unsupervised domain adaptation. In *Domain Adaptation in Computer Vision Applications*, pages 153–171. Springer, 2017.

[121] Cristina Rottondi, Riccardo di Marino, Mirko Nava, Alessandro Giusti, and Andrea Bianco. On the benefits of domain adaptation techniques for quality of transmission estimation in optical networks. *Journal of Optical Communications and Networking*, 13(1):A34–A43, 2021.

[122] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE communications surveys & tutorials*, 10(4):56–76, 2008.

[123] Martino Trevisan, Idilio Drago, Marco Mellia, Han Hee Song, and Mario Baldi. What: A big data approach for accounting of modern web services. In *2016 IEEE Int. Conf. on Big Data (Big Data)*, pages 2740–2745. IEEE, 2016.

[124] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–8. IEEE, 2018.

[125] Athula Balachandran, Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Srinivasan Seshan, Shobha Venkataraman, and He Yan. Modeling web quality-of-experience on cellular networks. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 213–224, 2014.

[126] Irena Orsolic, Dario Pevec, Mirko Suznjevic, and Lea Skorin-Kapov. A machine learning approach to classifying youtube qoe based on encrypted network traffic. *Multimedia tools and applications*, 76(21):22267–22301, 2017.

[127] Pedro Casas, Alessandro D'Alconzo, Florian Wamser, Michael Seufert, Bruno Gardlo, Anika Schwind, Phuoc Tran-Gia, and Raimund Schatz. Predicting qoe in cellular networks using machine learning and in-smartphone measurements. In *Ninth International Conf. on Quality of Multimedia Experience*, pages 1–6. IEEE, 2017.

[128] Dario Bonfiglio, Marco Mellia, Michela Meo, Dario Rossi, and Paolo Tofanelli. Revealing skype traffic: when randomness plays with you. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 37–48, 2007.

[129] A. Finamore, M. Mellia, M. Meo, and D. Rossi. Kiss: Stochastic packet inspection classifier for udp traffic. *IEEE/ACM Transactions on Networking*, 18(5):1505–1515, 2010.

[130] A. S. Buyukkayhan, A. Kavak, and E. Yaprak. Differentiating voice and data traffic using statistical properties. In *2013 International Conference on Electronics, Computer and Computation (ICECCO)*, pages 76–79, 2013.

[131] M. Di Mauro and M. Longo. Revealing encrypted webrtc traffic via machine learning tools. In *2015 12th International Joint Conference on e-Business and Telecommunications*, volume 04, pages 259–266, 2015.

[132] T. Sinam, I. T. Singh, P. Lamabam, N. N. Devi, and S. Nandi. A technique for classification of voip flows in udp media streams using voip signalling traffic. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 354–359, 2014.

[133] Nanditha Rao, A Maleki, F Chen, Wenjun Chen, C Zhang, Navneet Kaur, and Anwar Haque. Analysis of the effect of qos on video conferencing qoe. In *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1267–1272. IEEE, 2019.

[134] Boni Garcia, Micael Gallego, Francisco Gortazar, and Antonia Bertolino. Understanding and estimating quality of experience in webrtc applications. *Computing*, 101(11):1585–1607, 2019.

[135] Manuela Vaser and Sonia Forconi. Qos kpi and qoe kqi relationship for lte video streaming and volte services. In *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*, pages 318–323. IEEE, 2015.

[136] Jan Badshah, Majed Mohaia Alhaisoni, Nadir Shah, and Muhammad Kamran. Cache servers placement based on important switches for sdn-based icn. *Electronics*, 9(1):39, 2020.

[137] Jan Badshah, Muhammad Kamran, Nadir Shah, and Shahbaz Akhtar Abid. An improved method to deploy cache servers in software defined network-based information centric networking for big data. *Journal of Grid Computing*, 17(2):255–277, 2019.

[138] Dohyung Kim and Younghoon Kim. Enhancing ndn feasibility via dedicated routing and caching. *Computer networks*, 126:218–228, 2017.

[139] Stuart Clayman, Reza Shokri Kalan, and Müge Sayit. Virtualized cache placement in an sdn/nfv assisted sand architecture. In *2018 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*, pages 1–5. IEEE, 2018.

[140] Muhammad Shafiq, Xiangzhan Yu, and Asif Ali Laghari. Wechat traffic classification using machine learning algorithms and comparative analysis of datasets. *International Journal of Information and Computer Security*, 10(2-3):109–128, 2018.

[141] Petr Matousek, Ondrej Rysavy, and Martin Kmet. Fast rtp detection and codecs classification in internet traffic. *Journal of Digital Forensics, Security and Law*, 01 2014.

[142] M. C. S, S. H, and T. E. Somu. Network traffic classification by packet length signature extraction. In *2019 IEEE International WIE Conference on Electrical and Computer Engineering*, pages 1–4, 2019.

[143] P. Choudhury, K. R. Prasanna Kumar, G. Athithan, and S. Nandi. Analysis of vbr coded voip for traffic classification. In *2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 90–95, 2013.

[144] Giovanna Carofiglio, Giulio Grassi, Enrico Loparco, Luca Muscariello, Michele Papalini, and Jacques Samain. Characterizing the relationship between application qoe and network qos for real-time services. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Network-Application Integration*, pages 20–25, 2021.

[145] Robert C Streijl, Stefan Winkler, and David S Hands. Mean opinion score (mos) revisited: methods and applications, limitations and alternatives. *Multimedia Systems*, 22(2):213–227, 2016.

[146] Dunja Vucic and Lea Skorin-Kapov. The impact of packet loss and google congestion control on qoe for webrtc-based mobile multiparty audiovisual telemeetings. In *International Conference on Multimedia Modeling*, pages 459–470. Springer, 2019.

[147] International Telecommunication Union – Telecommunication Standardization Bureau. Recommendation ITU-T G.1070 – Opinion model for videotelephony applications. 2018.

[148] International Telecommunication Union – Telecommunication Standardization Bureau. Recommendation ITU-T G.107.1 – Wideband E-model. 2019.

[149] Martino Trevisan, Alessandro Finamore, Marco Mellia, Maurizio Munafo, and Dario Rossi. Traffic Analysis with Off-the-Shelf Hardware: Challenges and Lessons Learned. *IEEE Commun. Mag.*, 55(3):163–169, 2017.

[150] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[151] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[152] Laura Toloşi and Thomas Lengauer. Classification with correlated features: unreliability of feature ranking and solutions. *Bioinformatics*, 27(14):1986–1994, 2011.

[153] Lorien Y Pratt. Discriminability-based transfer between neural networks. *Advances in neural information processing systems*, pages 204–204, 1993.

[154] Daniel Perdices, Gianluca Perna, Martino Trevisan, Danilo Giordano, and Marco Mellia. When satellite is all you have: Watching the internet from 550 ms. In *Proceedings of the 22nd ACM Internet Measurement Conference*, IMC '22, page 137–150, New York, NY, USA, 2022. Association for Computing Machinery.

[155] Paul Davern, Noor Nashid, Cormac J Sreenan, and Ahmed Zahran. Httpep: A http performance enhancing proxy for satellite systems. *International Journal of Next Generation Computing (IJNGC)*, 2:242–256, 2011.

[156] Igor Bisio, Stefano Delucchi, Fabio Lavagetto, and Mario Marchese. Transmission rate allocation over satellite networks with quality of experience-based performance metrics. In *2014 7th advanced satellite multimedia systems conference and the 13th signal processing for space communications workshop (ASMS/SPSC)*, pages 419–423. IEEE, 2014.

[157] Adrien Thibaud, Julien Fasson, Fabrice Arnal, David Pradas, Emmanuel Dubois, and Emmanuel Chaput. Qoe enhancements on satellite networks through the use of caches. *International Journal of Satellite Communications and Networking*, 36(6):553–565, 2018.

[158] Andreas Sackl, Pedro Casas, Raimund Schatz, Lucjan Janowski, and Ralf Irmer. Quantifying the impact of network bandwidth fluctuations and outages on web qoe. In *2015 Seventh International Workshop on Quality of Multimedia Experience (QoMEX)*, pages 1–6, 2015.

[159] Tobias Hoßfeld, Florian Metzger, and Dario Rossi. Speed index: Relating the industrial standard for user perceived web performance to web qoe. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6, 2018.

[160] Diego Neves da Hora, Alemnew Sheferaw Asrese, Vassilis Christophides, Renata Teixeira, and Dario Rossi. Narrowing the gap between qos metrics and web qoe using above-the-fold metrics. In Robert Beverly, Georgios Smaragdakis, and Anja Feldmann, editors, *Passive and Active Measurement*, pages 31–43, Cham, 2018. Springer International Publishing.

[161] Enrico Bocchi, Luca De Cicco, and Dario Rossi. Measuring the quality of experience of web users. *ACM SIGCOMM Computer Communication Review*, 46(4):8–13, 2016.

[162] Martino Trevisan, Idilio Drago, and Marco Mellia. Pain: A passive web performance indicator for isps. *Computer Networks*, 149:115–126, 2019.

[163] Luis Roberto Jiménez, Marta Solera, Matías Toril, Carolina Gijón, and Pedro Casas. Content matters: Clustering web pages for qoe analysis with webclust. *IEEE Access*, 9:123873–123888, 2021.

[164] Alexis Huet, Antoine Saverimoutou, Zied Ben Houidi, Hao Shi, Shengming Cai, Jinchun Xu, Bertrand Mathieu, and Dario Rossi. Revealing qoe of web users from encrypted network traffic. In *2020 IFIP Networking Conference (Networking)*, pages 28–36. IEEE, 2020.

[165] Pedro Casas, Sarah Wassermann, Nikolas Wehner, Michael Seufert, Joshua Schüler, and Tobias Hossfeld. Mobile web and app qoe monitoring for isps-from encrypted traffic to speed index through machine learning. In *2021 13th IFIP Wireless and Mobile Networking Conference (WMNC)*, pages 40–47. IEEE, 2021.

[166] Pedro Casas, Sarah Wassermann, Nikolas Wehner, Michael Seufert, and Tobias Hossfeld. Not all web pages are born the same content tailored learning for web qoe inference. In *2022 IEEE International Symposium on Measurements & Networking (M&N)*, pages 1–6. IEEE, 2022.

[167] Olga Kondratyeva, Natalia Kushik, Ana Cavalli, and Nina Yevtushenko. Evaluating quality of web services: A short survey. In *2013 IEEE 20th International Conference on Web Services*, pages 587–594. IEEE, 2013.

[168] Sabina Baraković and Lea Skorin-Kapov. Survey of research on quality of experience modelling for web browsing. *Quality and User Experience*, 2:1–31, 2017.

[169] Tobias Hoßfeld, Sebastian Biedermann, Raimund Schatz, Alexander Platzer, Sebastian Egger, and Markus Fiedler. The memory effect and its implications on web qoe modeling. In *2011 23rd International Teletraffic Congress (ITC)*, pages 103–110, 2011.

[170] Pengfei Wang, Matteo Varvello, and Aleksandar Kuzmanovic. Kaleidoscope: A crowdsourcing testing tool for web quality of experience.

[171] Christian Niephaus, Mathias Kretschmer, and Gheorghita Ghinea. Qos provisioning in converged satellite and terrestrial networks: A survey of the state-of-the-art. *IEEE Communications Surveys & Tutorials*, 18(4):2415–2441, 2016.

[172] Matthieu Petrou, David Pradas, Mickaël Royer, and Emmanuel Lochin. Forecasting youtube qoe over satcom. In *The IEEE 97th Vehicular Technology Conference*, 2023.

[173] Nicolas Kuhn, François Michel, Ludovic Thomas, Emmanuel Dubois, Emmanuel Lochin, Francklin Simo, and David Pradas. Quic: Opportunities and threats in satcom. *International Journal of Satellite Communications and Networking*, 40(6):379–391, 2022.

[174] Waqar Aqeel, Balakrishnan Chandrasekaran, Anja Feldmann, and Bruce M Maggs. On landing and internal web pages: The strange case of jekyll and hyde in web performance measurement. In *Proceedings of the ACM Internet Measurement Conference*, pages 680–695, 2020.

[175] Nikhil Jha, Martino Trevisan, Luca Vassio, and Marco Mellia. The internet with privacy policies: Measuring the web upon consent. *ACM Transactions on the Web (TWEB)*, 16(3):1–24, 2022.

[176] Mohammad Rajiullah, Andra Lutu, Ali Safari Khatouni, Mah-Rukh Fida, Marco Mellia, Anna Brunstrom, Ozgu Alay, Stefan Alfredsson, and Vincenzo Mancuso. Web experience in mobile networks: Lessons from two million page visits. In *The World Wide Web Conference*, pages 1532–1543, 2019.

[177] Sarah Wassermann, Pedro Casas, Zied Ben Houidi, Alexis Huet, Michael Seufert, Nikolas Wehner, Joshua Schüler, Shengming Cai, Hao Shi, Jinchun Xu, et al. Are you on mobile or desktop? on the impact of end-user device on web qoe inference from encrypted traffic. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2020.