Capítulo

1

Content Steering: Leveraging the Computing Continuum to Support Adaptive Video Streaming

Roberto Rodrigues-Filho (UFSC), Eduardo S. Gama (UNICAMP), Marcio Miranda Assis (UNICAMP), Roger Immich (UFRN), Edmundo Madeira (UNICAMP), Luiz F. Bittencourt (UNICAMP)

Abstract

Video streaming has become one of the most used Internet applications nowadays, with numerous leading technology companies competing for dominance in a market valued in the billions. The delivery of high-quality streaming services requires strategic utilization of computing resources near end-users. Emerging technologies like 6G and edge-cloud continuum infrastructures are being explored to meet the growing demands. These technologies promise to enable fast and reliable data transfer for large data volumes, with the edge-cloud continuum facilitating service placement mobility from central cloud data centers to edge devices near users. However, managing seamless service mobility and precise computing resource allocation for quality services remains complex. In the light of this, the zero-touch network concept was born. It eliminates the need for manual network configuration and it is becoming popular in this context. Specifically, in video streaming, the integration of the Content Steering architecture from the Dynamic Adaptive Streaming over HTTP (DASH) protocol with container orchestration technologies could allow for autonomous video streaming service placement across the computing continuum, reducing human involvement and optimizing computing resource use. This short course provides a hands-on experience with the latest technology in this domain, teaching participants about cutting-edge architectures and tools for creating and managing adaptive video streaming applications using the latest content steering architecture introduced in the DASH protocol. Participants will build a small edge-cloud virtual and local testbed to explore request steering strategies for video content across the computing continuum. The course also addresses current challenges and future research opportunities in this evolving field.

1.1. Introduction

Video streaming content is one of the most popular forms of content on the Internet nowadays, with big techs fighting for a market share in the growing video streaming application market. This scenario has motivated the research and development of new techniques and methods for supporting the demands for video content streaming in today's networking infrastructures.

Over the years, many technologies have been developed to better support the growing demands for video streaming content [51, 33, 53]. 5G networks and the expected 6G networks are recent technologies supporting fast access to this type of content [66]. Furthermore, Zero-touch Network and Service Management [40] is being investigated to provide closed-loop autonomous control over the configuration and optimization of such infrastructures and enable them to support resource-intensive applications such as video streaming.

Aligned with these technological advances to create and manage better infrastructures to support the demands of contemporary services, the Edge-cloud Continuum is an emerging infrastructure that has gained popularity in recent years. These are hierarchical infrastructures of highly heterogeneous devices that comprehend the end-user and IoT devices (e.g., laptops, smartphones, drones, sensors, actuators, etc.), a supporting network of edge/fog computing that brings the capability of cloud services closer to end users and extends itself to cloud datacenters [10].

This resulting computing infrastructure enables and supports a plethora of applications that generate and process high volumes of data. It enables the strategic deployment and allocation of video streaming services, bringing them closer to end-users and thereby minimizing network latency, as noted by Gama et al. [27]. Within this continuum, services and data are strategically positioned to optimize the end-user experience, particularly in scenarios marked by high user mobility and escalating demands for video quality. Consequently, video streaming applications running in such infrastructures can effectively avoid network congestion and other challenges, ensuring superior quality of experience for end-users.

This chapter, therefore, is dedicated to providing an overview of the aforementioned technology and concepts, focusing on the edge-cloud continuum infrastructure as the bedrock for supporting video streaming applications and tackling its associated challenges. We also provide an introduction and overview of the content steering architecture defined in the Dynamic Adaptive Streaming over HTTP (DASH) [1] protocol as a core mechanism for steering incoming video requests to appropriate services placed closer to end users. Finally, we present the technologies that are often used at the platform level to manage the platform's underlying devices and support service placement.

Furthermore, this chapter presents a GitHub [1] repository with a fully functioning video streaming application that leverages the content steering architecture to better exploit resources in the edge-cloud continuum. We provide all the necessary code for the interested reader to perform experiments with video streaming and content steering architecture in their machines.

[1]https://github.com/robertovrf/content-steering-tutorial

The remainder of this chapter is organized as follows: Sec. 1.3 presents a definition of the Edge-cloud Continuum and discusses the challenges and opportunities of such distributed infrastructures; Sec. 1.3 details the video streaming protocols and introduces the content steering architecture; Sec. 1.4 introduces the main technologies and tools used to manage clusters on the Edge-cloud continuum; Sec. 1.5 presents the available repository with a fully functioning video streaming application; Sec. 1.6 discusses the challenges and research opportunities for supporting video streaming applications on the edge-cloud continuum; and finally, Sec. 1.7 concludes the chapter.

1.2. Exploiting the Edge-cloud Continuum Resources

In this section, we introduce the concept of edge-cloud computing continuum and contextualize how new technologies are needed to exploit the resources in the computing continuum enabled by such infrastructures. To conclude, we define how these infrastructures support video streaming applications and the challenges involved at the platform and service level in exploiting the underlying resources available at the infrastructure level.

1.2.1. Defining the Edge-cloud Continuum

The Edge-cloud Continuum is a hierarchical and highly heterogeneous infrastructure with devices spread throughout a wide geographical location. It ranges from centralized cloud computing platforms located at the core of the network extending to end-user devices located at the edge of the network, resulting in a computing continuum.

Bittencourt et al. [10] and Peng et al. [46] have described the computing continuum stemming from the development of the fog/edge computing technology created to aggregate and process data generated by the Internet of Things (IoT) devices. A different and possible definition stems from the perspective of service placement from the cloud platform to devices executing closer to end users. Fig 1.1 illustrates the Edge-cloud Continuum concept, giving an overview of the infrastructure and its main characteristics.

The hierarchical edge-cloud infrastructure is represented in the aforementioned figure, where end-user devices are placed at the bottom of the hierarchy. These devices have an important characteristic that mainly characterizes the computing continuum. End-user devices often have constrained resources and small network latency [49]. Examples of these devices are sensors, household appliances (e.g., smart TVs), laptops and desktop computers, smartphones, autonomous vehicles, drones, and many other computing devices.

In the second layer of the infrastructure (LAYER 2..N), we have the edge computing platform [57]. These are clusters of computers located in close proximity to end-users. These clusters of computers have more available computing resources (e.g., CPU and storage) than the user devices but add a bit more network latency. They are also often used as the base for Content Delivery Networks (CDN) [70, 39, 12] due to their strategic location close to the end user and thus presenting small network latency while presenting reasonable and available computing resources. The infrastructure of servers presented as the second layer can extend further as a set of clusters that connects one geographic region to the cloud. This second layer can be extended to many layers up to the cloud.

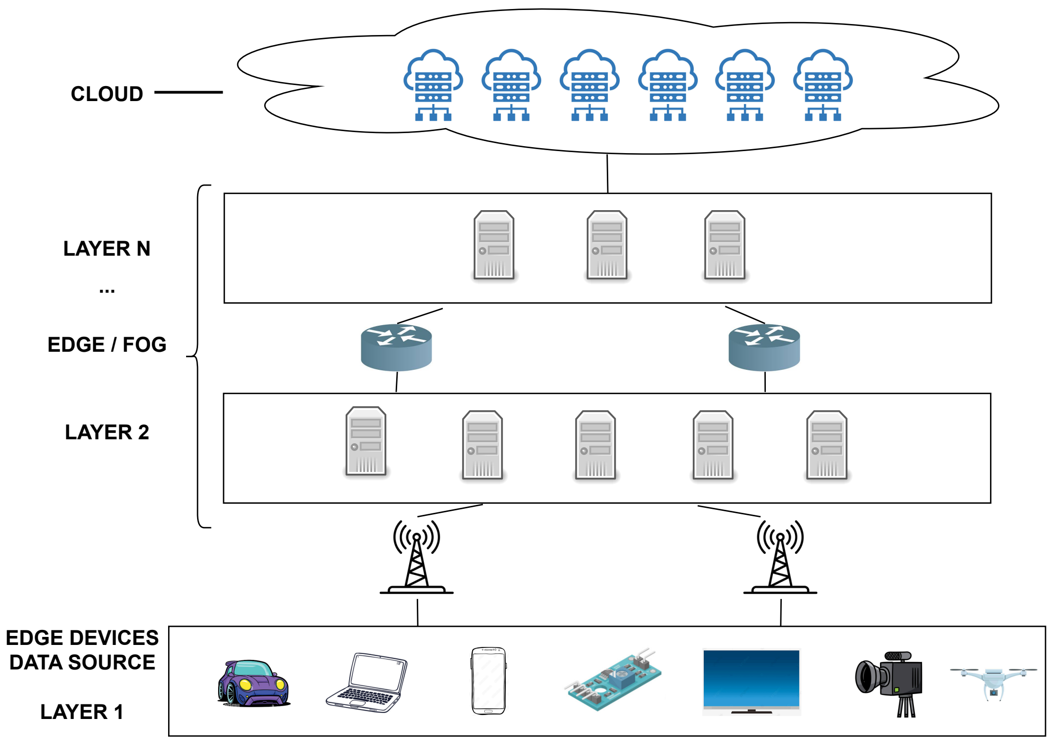

Figure 1.1. Overview of Edge-Cloud Continuum Infrastructure.

The cloud computing platform is located at the top layer. At this layer, the infrastructure has the characteristic of having the illusion of unlimited computing resources while maintaining a high network latency. At this level, CPU and storage-intensive services are better accommodated, but due to the distance between end users and the services, there is an added network latency in response time.

The resulting edge-cloud infrastructure has been explored to support services for systems that range from machine learning-based, virtual and augmented reality applications to video streaming, autonomous vehicles and others in recent years. The growing popularity of such infrastructures makes deploying and managing applications in such infrastructures crucial for supporting contemporary systems.

The development of systems to exploit resources in the edge-cloud continuum is marked with many challenges. These challenges mainly involve monitoring services throughout the continuum, supporting strategic service and data placement across edge and cloud to better exploit computing resources as the demands of the application and users change over time.

1.2.2. Exploiting the Hierarchy of Computing Resources

In order to explore the computing continuum infrastructure, systems face many challenges [10, 41]. One of the main challenges is to equip software deployed on the continuum to be able to move around the infrastructure to better exploit the available resources.

This is a challenging task that involves anticipating the resource demands of applications and the unexpected changes of those needs as changes occur in the operating environment. For instance, if the systems' workload volume suddenly spikes or the resources suddenly disconnect from the network, the executing software is required to adapt in order to properly exploit the underlying resources.

Moreover, besides adapting in the face of unexpected changes, software executing on the continuum should also adapt in anticipation of events. For instance, in a scenario where users are moving (i.e., user mobility) we often can apply forecasting strategies to predetermine the route where the users may take [2]. Especially if we consider a user's daily routine. In such cases, we can anticipate where to place services and data to better exploit resources in the continuum to increase users' Quality of Experience (QoE) as they move.

In general, building applications to exploit the resources in an edge-cloud continuum infrastructure involves many important challenges. We now present the main ones and discuss possible directions for future research. The main challenges we present in this section are the actual challenge of managing and properly allocating resources in the continuum, the challenge of abstracting the complexity of programming and deploying services capable of moving from cloud to edge to end-user devices and vice-versa, and finally, the challenges of monitoring services across the continuum.

1.2.2.1. Resource Allocation and Optimization

Exploiting computing resources in the continuum is crucial for optimizing systems performance. Although resource allocation is a classical distributed systems problem [30], it gains new dimensions when considering the scale of edge-cloud platforms.

The resource allocation problem usually requires schedulers that implement models to receive the set of available resources as input, and the application demands and outputs a plan for when and where to execute the application. In the edge-cloud continuum setting, resource allocation becomes not only difficult due to the number of constraints and requirements from applications and the sheer number of heterogeneous resource capabilities but also due to the volatility of its operating environment.

In the edge-cloud continuum, it is not sufficient to have schedulers implementing fixed decision-making models, but instead to have models that can adapt and evolve to consider changes in the operating environment. Theses changes includes the addition and removal of computing resources, end user mobility, incoming workload pattern changes, and so on.

Furthermore, to tackle the scalability in terms of the number of devices present in such deployment environments, we divide resource allocation tasks among controllers. This allows schedulers to act on a specific set of devices, considering requirements and constraints from a specific set of services, reducing the complexity of the model. This leads to a problem of where to place such controllers and how they should cooperate to optimize the global system's goal instead of optimizing isolated parts of the system.

Relevant strategies have been discussed in the literature to tackle the volatility issue when making resource allocation decisions in volatile environments [61, 15]. Moreover, different coordinating strategies have been explored to make joint decisions in multiple controllers over distributed infrastructures [19, 11, 45].

1.2.2.2. Service and Data Placement

Placing services and data on the different devices that compose the edge-cloud continuum is essential to maintaining a high Quality of Service (QoS) for certain systems. Some applications require services that are commonly executed on the cloud to be moved and executed at the edge closer to users or move services from end-user devices to the cloud when the service requires more resources.

Thus, strategically placing services and data in specific parts of the continuum is an important task to optimize applications running on the continuum. Regarding service mobility, many supporting technologies have gained popularity. Container technologies and container orchestration technologies provide a platform to enable general adaptation, including mobility. For instance, services can be wrapped inside Docker [2] containers and deployed in any cluster on the continuum managed by Kubernetes [3].

Furthermore, it is possible to see a trend in the industry to create stateless services to facilitate the exploitation of containers and container orchestration as a supporting platform for adaptation. A stateless service is a component that retains no information on previous requests or users' sessions and thus can be moved and replicated throughout the infrastructure with no changes to its implementation. Due to being stateless and self-contained, microservice-based architectures [31, 25, 26] and Function-as-a-Service [56, 58] have gained popularity in contemporary software systems and are widely used in the edge-cloud continuum.

However, not every service can be developed with no state. Many services require access to state information to compose a functioning system. In this context, engineers carefully design services that carry no state but are highly dependent on data stored in a cloud database. In these settings, the service is free to exploit the underlying platform to be freely replicated without maintaining state consistency across its replicas, but it suffers from performance degradation when placed far from the database on which it depends.

These data-dependent services can still be created as microservices or using a programming model such as Function-as-a-Service. They can also still leverage the container-orchestration platform underneath it to be placed in different devices across the distributed infrastructure. In a setting where the edge and cloud seamlessly integrate, we observe a significant decline in performance when these services are deployed on an edge node to facilitate rapid access from user devices. This is because the service keeps constantly accessing (updating or fetching) data in the cloud, adding network latency to transfer data to/from the cloud, defeating the purpose of placing the service closer to end users.

As a response to this problem, recent research has focused on the implementation of the concept now known as stateful Function-as-a-service (FaaS) [48]. Using this, the

[2]Docker: Container Technology - `https://docker.com`

[3]Kuberbetes: Container Orchestrator - `https://kubernetes.io`

functions can move to the edge, and access replicated data that is now copied to the edge. Likewise, similar strategies also support data access on the edge to microservice-based solutions [50].

1.2.2.3. Monitoring in the Edge-Cloud Continuum

In order to make precise service placement decisions and allocate resources on the continuum, a key challenge is to collect the correct metric based on which placement decisions will be based. In an ever-changing environment such as the edge-cloud continuum, an important task is monitoring the correct metric at the correct location and reporting the collected information to the decision-making component in time.

The edge-cloud continuum operating environment demands a highly adaptive and flexible solution for monitoring metrics. This is due to the constant changes that affect different aspects of the environment. Changes in the workload pattern and volume dictated by the way users interact with the system, changes in the devices on the infrastructure – new devices are integrated into the platform or functioning devices become unavailable, changes regarding the mobility of users, and so on.

These changes require the system to accommodate the new operating environment conditions to maintain a high-quality user experience. As the system changes, the components responsible for monitoring the system must change accordingly. For instance, if services are replicated or placed in different parts of the infrastructure, the monitoring system should adapt accordingly to collect data to support further decision-making. Therefore, edge-cloud continuum infrastructures require monitoring solutions that can adapt to the location where they collect metrics, the type of metric they collect, and how they process and make the collected information available to the decision-making component.

Adaptive monitoring solutions have been the focus of many research over the years [22, 63, 71, 4]. These research papers explore the requirements for creating adaptive monitoring systems, the mechanisms to support such adaptation, and the dimensions of the monitoring that need adaptation according to the scenario and environment in which the system is deployed. However, few initiatives tackle the end-to-end engineering challenges for deploying and managing adaptive monitoring solutions at the scale of the edge-cloud continuum, the most relevant presented by Colombo et al. [16].

Despite the few initiatives exploring an end-to-end solution, many technologies have been developed for monitoring services and data in the edge-cloud continuum. The review conducted by Verginadis [67] presents an interesting overview of many of such technologies describing Prometheus [4], Datadog [5], Zabbix [6] and Elastic Stack [7].

In the end, besides the scale on which these monitoring solutions are required to operate when deployed over edge-cloud continuum infrastructures, these tools should also consider the sheer levels of heterogeneity for monitoring the highly diverse devices

[4]

[5]`https://www.datadoghq.`

[6]`https://www.zabbix.com.`

[7]`https://www.elastic.co/elastic-stack`

that compose the continuum. That entails the application of different techniques to probe different metrics from different devices.

1.2.3. Computing Resource Demands for Video Streaming Applications

In the remainder of this chapter, we will focus on exploring service placement across the edge-cloud continuum to support video streaming applications. As the foundation of the explored approach, we leverage the Content Steering [1] architecture to exploit resources on an edge-cloud continuum infrastructure for video streaming.

Particularly for video streaming applications, the edge-cloud continuum infrastructure offers a great range of computing resources to accommodate the ever-increasing demands of end users [28]. Considering the data volume, Internet connection quality, and mobility of end users, systems are required to adapt to new operating conditions.

Current architectures to support video streaming solutions consider the placement of video content in CDNs. This data placement closer to where the end users are connected reduces the network latency for the users to access the desired content. This helps increase the quality of experience for the end user. Many video streaming solutions use multi-CDNs hosting copies of video content and have user players directly fetch video content from specific CDNs closer to the end user using specific URLs.

HSL [8] and DASH-IF [9] [59] have proposed and implemented a content steering architecture to determine which CDN the video player is fetching data during video playing. This gives the player more flexibility in adapting where the video is being fetched. Particularly in scenarios with user mobility or when the number of users increases, content steering can assist in balancing the request load from one location to another, reducing network congestion while increasing the quality of experience for end users.

Content-steering architectures are very useful when considering the placement of video streaming services across the continuum. Over the next sections, we discuss how to leverage such architecture to steer video content requests to specific services placed throughout the continuum. This, in turn, opens the possibility for a more flexible and adaptive infrastructure capable of handling video streaming to millions of users.

Finally, edge-cloud continuum infrastructures are also ideal for scaling content-steering solutions, as we can place different steering controllers across the infrastructure. This strategic placement of content steering controllers can act on a specific network region, reducing the number of users handled per controller and allowing this solution to scale drastically to accommodate millions of users worldwide.

1.3. Video Streaming Architecture Overview

This section investigates the mechanisms and structures that support video data transmission within network infrastructures. Section 1.3.1 presents a vision from encoding video content into digital formats to optimizing data packets for continuous delivery. Each component within these architectures ensures a seamless viewing experience for end users. In Section 1.3.2, we designed an architecture for Content Steering Services. Initially, a cen-

[8]HTTP Live Streaming (RFC 8216)

[9]Dynamic Adaptive Streaming over HTTP Industry Forum

tralized steering server orchestrates operations, followed by an Edge-Cloud Continuum approach. The service's main objective is to show users' directness to a video streaming service in real-time, balancing the load and improving scalability.

1.3.1. Adaptive Video Streaming

Although traditional video delivery methods have their merits, they face certain limitations when it comes to connection-oriented protocols like Real-Time Messaging Protocol (RTMP/TCP) [65] or Real-time Transport Protocol (RTP/UDP) [54] for connectionless protocols. Typically, the media server pushes the media to the client in these protocols. However, these methods require help scaling the infrastructure, depending on specific vendors, and having high maintenance costs necessitating complex and expensive servers. Over time, new technologies have emerged to overcome these limitations and enhance video streaming [8]. One of the most popular solutions is HTTP Adaptive Streaming (HAS), which offers several benefits over traditional methods. Adaptive streaming is a dynamic video delivery technique that adjusts the video playback quality in real time based on the user's network conditions, with the primary goal of providing seamless viewing playback by adapting to changes in available bandwidth.

Dynamic Adaptive Streaming over HTTP (DASH) [60] and Apple's HTTP Live Streaming (HLS) [44] are two of the most popular HAS solutions in use today. These methods, which handle over half of all video streams, are particularly prevalent in on-demand and live streaming platforms. Figure 1.2 depicts the basic workflow concept of a HAS service for video streaming involves the video source transmitting the raw video data to the HAS servers, which then encode it to produce multiple representations of a single video.

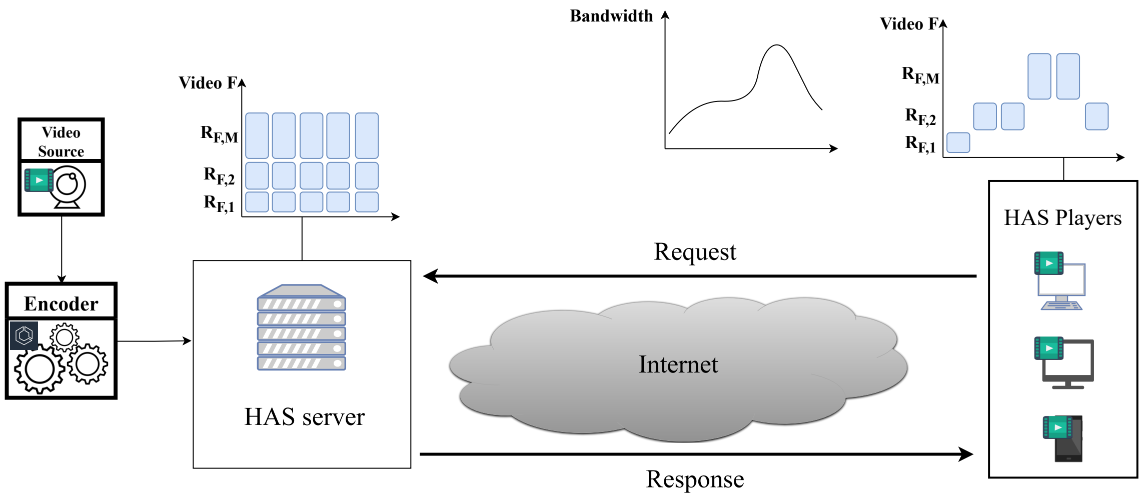

Figure 1.2. Overview of typical HTTP adaptive streaming server. The server provides the media segments in different representations. The client requests media segments in desired representations and measurement of throughput and buffer fill level.

In the context of HAS, videos are divided into short segments, typically lasting one to ten seconds. The encoder then encodes each segment into different versions with varying resolutions or bitrates. The Media Presentation Description (MPD) manifest file keeps track of the storage location of these segments on a server. When a user watches a video, their HAS player utilizes this information, internet speed, and device capabilities to select the best version to stream. This process of segmentation and encoding is a key aspect of HAS, enabling it to adapt to the user's network conditions and provide a seamless viewing experience.

Different coding formats can encode the video, such as Advanced Video Coding (AVC), High Efficiency Video Coding (HEVC), Versatile Video Coding (VVC), VP9, and AOMedia Video 1 (AV1). These codecs generate multiple encoded representations of the video at different bitrates, catering to diverse network bandwidths. Clients requesting the video content can seamlessly download segments from these encoded representations. The client-side player intelligently selects the appropriate bitrate based on real time network conditions and dynamically switches between representations to ensure smooth playback. The downloaded segments are then concatenated within the playback buffer and processed by a standard decoder for video rendering. This process ensures a conforming bitstream compatible with decoders on various client devices.

AVC, commonly known as H.264, is a foundational video compression standard developed by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). Its widespread adoption owes to its versatility and compatibility across various platforms. However, despite its widespread use, AVC must improve to achieve optimal compression efficiency for high-resolution and high-quality video content.

HEVC, also known as H.265, emerges as the successor to AVC, aiming to address its predecessor's shortcomings and meet the increasing demand for higher quality video at lower bitrates. Developed by the same organizations responsible for AVC, HEVC introduces advanced encoding techniques such as larger block sizes, improved motion compensation, and more efficient entropy coding. Consequently, HEVC offers substantially improved compression efficiency, making it an attractive choice for various applications, including ultra-high-definition television (UHDTV), video streaming, and mobile video. Despite its advantages, HEVC adoption has been hindered by licensing issues, particularly in terms of royalties.

VP9 and AV1 are open and royalty-free video compression formats. They offer comparable compression efficiency to HEVC while remaining accessible and free. As a result, open-source implementations find widespread use in web browsers, online video platforms, and streaming services. However, some devices and platforms may limit adoption due to the need for more hardware support.

VVC, known as H.266, emerges as the latest generation video compression standard. VVC represents a significant leap in compression efficiency compared to previous standards, offering enhanced support for high-resolution video, HDR content, and immersive multimedia experiences. While VVC demonstrates remarkable compression gains, its adoption may face challenges due to licensing issues and the need for hardware support. Despite these challenges, VVC holds promise for applications requiring the highest compression efficiency and quality in video delivery.

1.3.1.1. Application-Network Interaction in Video Streaming

Regarding the HAS paradigm, video streaming applications are traditionally built upon the Transmission Control Protocol (TCP) [21]. Within this paradigm, application-layer Adaptive Bitrate (ABR) logic operates independently from the transport layer. While ABR logic dynamically adjusts streaming quality based on available bandwidth, TCP congestion control regulates data flow to avoid network congestion. However, the lack of coordination between these mechanisms often results in inefficiencies and concerns about fairness. ABR logic, which operates independently at the application layer, has a tendency to adjust the streaming quality too frequently without taking into account the network conditions monitored by TCP congestion control. This can lead to unfair bandwidth allocation among different sessions and can also cause instability in network utilization.

Additionally, uncoordinated adjustments by both mechanisms can exacerbate congestion issues instead of alleviating them [47]. To tackle these difficulties, an integrated approach is necessary wherein ABR logic and TCP congestion control cooperate to optimize resource allocation and improve network stability. By synchronizing these control loops and promoting communication between layers, it becomes possible to mitigate inefficiencies and ensure a fair bandwidth distribution among streaming sessions, thereby enhancing an overall users' QoE [42].

The emergence of the Quick UDP Internet Connections (QUIC) transport protocol, as presented in [34], has opened up new avenues for improving the interaction between video streaming applications and transport functionalities. While certain studies advocate for QUIC as a promising solution for video streaming, others engage in debates comparing it to TCP [55]. Furthermore, concerted efforts have been made to enhance QUIC's support tailored to video streaming applications.

A recent study [13] has evaluated QUIC's performance in the context of real time video streaming and found instances where QUIC exhibited excessive reliability, leading to suboptimal performance compared to TCP. QUIC performs better than the TCP protocol, where low-throughput use cases are necessary. However, in some cases, there are degradations compared to TCP and previous generations of HTTP, where the network bandwidth is poor or there is a variation in the bandwidth, as is typically experienced during mobility. The dynamics of application-network interaction for video streaming, particularly concerning integrating emerging transport protocols like QUIC, represent fertile ground for future investigation. Understanding these interactions and developing integrated solutions tailored to the specific needs of video streaming applications present exciting avenues for future research and innovation in this domain.

1.3.1.2. Bitrate Adaptive Schemes

Within the domain of client-server multimedia delivery, Bitrate Adaptation Schemes can be classified into client, server(s), network, and mixed-based adaptation. However, in this work, we limit our scope to client-based adaptation, representing most approaches [8]. These schemes delegate the client player to autonomously select the optimal video segment representation based on the network conditions. This selection process hinges on one or more metrics, with available bandwidth and playback buffer health being the most common. Client-based adaptation schemes aim to achieve a seamless viewing experience by dynamically adjusting the bitrate of the streamed video. They strive to balance maintaining high video quality with mitigating the effects of fluctuating bandwidth, thereby preventing buffering stalls and ensuring smooth playback.

Within the domain of client-server multimedia delivery, Bitrate Adaptation Schemes can be classified into client, server(s), network, and mixed-based adaptation. However, this work limits our scope to client-based adaptation, representing most approaches [8]. These schemes delegate the client player to autonomously select the optimal video segment representation based on the network conditions. This selection process hinges on one or more metrics, with available bandwidth and playback buffer health being the most common. Client-based adaptation schemes aim to achieve a seamless viewing experience by dynamically adjusting the bitrate of the streamed video. They strive to balance maintaining high video quality with mitigating the effects of fluctuating bandwidth, thereby preventing buffering stalls and ensuring smooth playback.

In general, available bandwidth- and buffer-based adaptation suffers from poor QoE due to unreliable bandwidth estimation methods and long-term bandwidth fluctuations. Meanwhile, in mixed-based adaptation, the client selects bitrate using bandwidth, buffer occupancy, segment size, and duration metrics. Thus, many studies can develop complex algorithms to select the following segments. Among the studies in mixed-based adaptation, we highlight some studies that have adopted predictive strategies based on the past and current states of the player context as well as the device context to determine the next segment's bitrate [5, 38]. Meanwhile, in Bentaleb *et al.* [6], a Game Theory Adaptive bitrate scheme was developed. This technique employs a cooperative game in coalition form, enabling the clients to reach a consensus on the ABR selection process by formulating it as a bargaining process. Consequently, this approach enhances the viewer's QoE and ensures video stability.

Another movement noted in academia is the increasing popularity of Learning-based schemes, which benefit from the latest advancements in machine learning techniques [32, 69, 7]. Learning-based schemes can derive effective strategies without making any assumptions about the environment. However, their performance heavily depends on the quality of the training data. Since network environments can be diverse, and their dynamics change over time, predicting future states accurately requires more effort. Additionally, implementing learning-based schemes on constrained devices is impractical due to their high storage and computational costs.

1.3.2. Content Steering Server-based Design Architecture

Designing a video streaming architecture requires a distinct approach, especially when considering the involvement of multiple content delivery networks (CDNs), each operating independently with its multimedia contents. CDNs currently play a vital role in delivering media content to end users. They can replicate their content across numerous locations or deploy distributed servers to retrieve content from a central source. Such features imply that designing a video streaming architecture entails thorough planning and coordination.

However, CDNs have limitations when it comes to the network edge. Some CDNs may only provide coverage in some relevant regions, while others may face internal capacity constraints. Moreover, some CDNs may need more caches to support the effective delivery of extensive video collections to the target audience. As a result, some works are utilizing dynamic "CDN switching" technologies to adjust content delivery paths for streaming dynamically.

To address the growing demand for streaming services, the technical communities responsible for developing prominent streaming standards have devised a system where every end user regularly contacts a central manager, the content steering server. End users receive instructions on which CDN to retrieve the content during each interaction. The steering server implements a load balancing strategy based on user feedback and collected global metrics. This strategy can balance long-term business logic and short-term QoE maximization.

In order to explain how the Content Steering protocol mechanism operates, we provide the manifest files for both HLS and DASH locations for CDNs and the Steering Server during a streaming session. We will use the manifest files in 1.1 and 1.2. In these examples, we will use `https://cdn.com` as the CDN server and `https://steeringserver.com` as the content steering server. The examples here demonstrate only a single CDN server, although multiple tags can implement it.

```
#EXTM3U
#EXT-X-CONTENT-STEERING:SERVER-URI="https://steeringserver.com"
#EXT-X-STREAM-INF:BANDWIDTH=1280000,PATHWAY-ID="cdn"
https://cdn.com/hi/video.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1280000,PATHWAY-ID="cdn-2"
https://cdn-2.com/hi/video.m3u8
```

Listing 1.1. HLS manifest

```
<BaseURL serviceLocation="cdn">https://cdn.com/</BaseURL>
<BaseURL serviceLocation="cdn-2">https://cdn-2.com/</BaseURL>
<ContentSteering defaultServiceLocation="cdn" queryBeforeStart="true">
https://steeringserver.com
</ContentSteering>
```

Listing 1.2. HLS manifest

The manifest file received by the user also contains metadata from steering elements. HLS/DASH streaming players seamlessly detect the presence of steering servers and initiate communication with them throughout the streaming session. They send HTTP GET requests to the specified steering server indicated in the manifest, potentially incorporating additional parameters, such as throughput and the pathway utilized by the user.

In this example, the response contains two servers identified by the PATHWAY-PRIORITY array. These servers are capable of providing the requested video segments. It prioritizes the EdgeCache server, while the CDN server has a lower priority. The specifications of the EdgeCache are detailed in the PATHWAY-CLONES with an original CDN base, and the edge node address rules are given by the URI-REPLACEMENT, with the host replaced by a URL in HOST.

```
{
"VERSION": 1,
"TTL": 10,
"RELOAD-URI": "https://steeringserver.com?session=abc",
"PATHWAY-PRIORITY": ["EdgeCache", "CDN", "CDN-2],
"PATHWAY-CLONES": [
   {
       "BASE-ID": "CDN",
       "ID": "EdgeCache",
       "URI-REPLACEMENT": {
           "HOST": "https://edgecacheserver.com",
       }
   },
    {
        "BASE-ID": "CDN",
        "ID": "EdgeCache",
        "URI-REPLACEMENT": {
           "HOST": "https://edgecacheserver.com",
        }
    }
]
}
```

Listing 1.3. Json response.

The client receives these instructions with a Time-To-Live (TTL) of 10 seconds, indicating the response interval for requesting the next update. Reducing response time is crucial for enabling many additional system utilities. Thus, when TTL becomes shorter than the size of the player's buffer, this automatically enables QoE optimizations, such as buffering prevention or allowing clients to use higher-quality streams.

This callback mechanism ensures timely adjustments to the streaming pathways based on network conditions or nodes' availability. Shorter response times are critical for different network congestion adaptations, such as fault-tolerant, mobility, and many other applications.

This syntax for steering server responses and client-server interactions remains consistent for both HLS and DASH systems. Consequently, a single server can manage content steering operations for both protocols. For detailed specifications regarding the response format within the Steering response, refer to the guidelines outlined in [1].

Within this multimedia service inside the Edge-Cloud Continuum, the architecture of video streaming systems emerges as a dynamic ecosystem comprising different components. These components include client-side elements, CDNs, edge cache servers, origin servers, and steering servers. Together, they form a network of agents working seamlessly to ensure efficient content delivery. Each stage in this multimedia delivery step significantly influences end users' QoE. The concept of Content Steering on the Edge-Cloud Continuum addresses the dynamism of edge computing environments and enables applications to exploit edge-cloud computing resources better. In this Section, our survey focuses on the key aspects of the distribution process, firstly, in a centralized steering server in the cloud. Then, we integrated the service with the Edge-Cloud Continuum.

1.3.2.1. Centralized Steering Server-based Design

In a Centralized Steering Server-based design, a single entity executes the load balancing strategy in content delivery. This design's workflow typically involves achieving some beneficial effect. For example, it may perform fault tolerance control through the video streaming servers, increasing the system's reliability. It may also perform CDN load balancing, enabling broader distribution.

We can note some limitations. The first one is scalability. As the number of clients and concurrent requests increases, the centralized steering server may encounter scalability challenges. Managing a large volume of client requests and dynamically directing them to the optimal CDN in real time requires substantial computational resources and network bandwidth. Scaling up the steering server infrastructure to accommodate growing demand can take time and effort. Let us assume, for example, that we have an event watched by 1M of concurrent viewers. Then, with 10 seconds TTL, the steering server must process at least 100K requests per second. That is a high number, conventional hardware, and some non-trivial logic required for deriving each steering response; it may easily overload a single server or a cluster of servers.

Maintaining and operating a centralized steering server infrastructure incurs significant ongoing costs. These costs include hardware acquisition, network infrastructure expenses, software development, and maintenance. Such expenses can be considerable, as operating costs can escalate rapidly as the system scales to accommodate more clients and higher traffic volumes.

A prolonged TTL diminishes the usefulness and efficiency of content steering mechanisms. Although a long TTL may be suitable for basic load balancing and CDN management, more is needed for other critical objectives such as QoS and QoE optimizations. It may also compromise the speed of fault tolerance responses. For example, when clients experience buffering issues during content delivery, redirecting them to an alternative CDN after a long delay may not effectively mitigate the buffering problem.

1.3.2.2. Content Steering at the Edge-Cloud Continuum

Leveraging edge and cloud resources allows for greater scalability and flexibility in content delivery infrastructure. Edge servers can handle localized spikes in demand and deliver content closer to end-users, while cloud resources provide additional capacity and support for global content distribution. Overall, Content Steering at the Edge-Cloud Continuum offers speed, scalability, reliability, and personalization, enabling organizations to deliver content more efficiently and effectively to end users worldwide.

Figure 1.3 illustrates the proposed Edge-Cloud Steering implementation. The design consists of two stages. In the first stage, the cloud steering master has its load balancing strategy given the list of CDN servers. They also decide which regional edge steering server should take responsibility for the user in the next TTL turn. Once a request arrives, the master node sends the response based on the list of CDN and edge steering server URLs. In the second stage, the system makes subsequent steering decisions at TTL intervals for each streaming session. Edge computing platforms implement all oper-

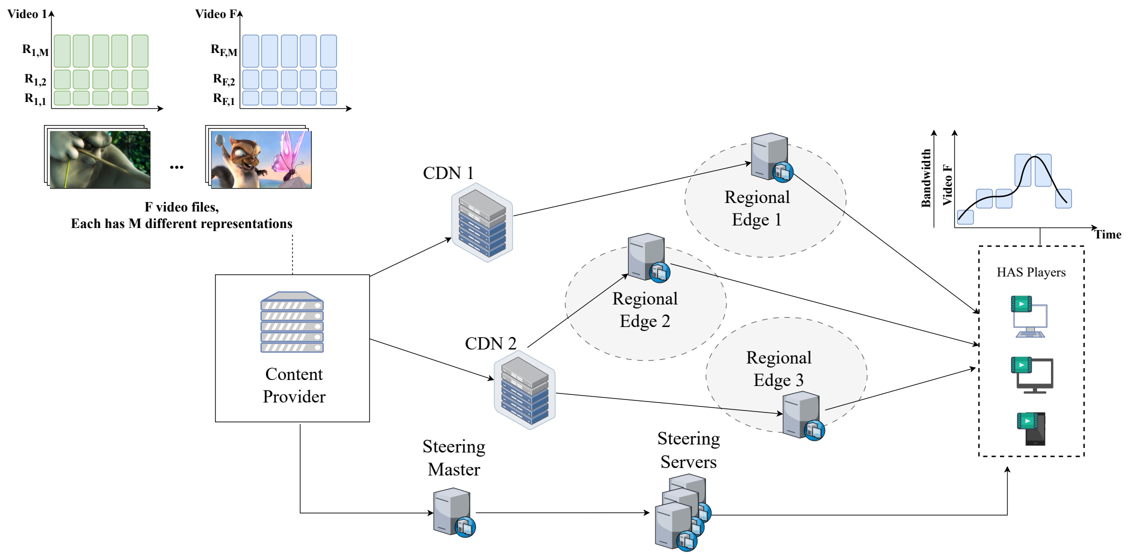

Figure 1.3. Overview of Content Steering at the Edge-Cloud Continuum: Directing user requests to edge-based content delivery systems, leveraging real-time network conditions and user proximity.

ations. This approach shows great potential in addressing the limitations of a centralized approach [24, 20]:

- **Scalability:** This implementation achieves scalability levels comparable to CDNs or platforms responsible for executing edge functions. By distributing processing tasks across multiple edge points, it can seamlessly accommodate growing demands without compromising performance;

- **Cost-Efficient Deployment:** Deploying this implementation becomes significantly more economical due to reduced bandwidth and per-request costs at CDNs or edge platforms compared to the higher egress traffic costs associated with cloud platforms. This cost-effectiveness makes it a more viable option for organizations seeking efficient content delivery solutions;

- **Enhanced Responsiveness:** The two-phase implementation enhances responsiveness by enabling lower TTL between clients and servers. This reduced latency facilitates quicker interactions between users and content servers, leading to smoother content delivery experiences.

Reducing response times is paramount for unlocking additional functionalities within the system. When TTL values become shorter than the size of the player's buffer (e.g., 1-30 seconds), it automatically enables QoS and QoE optimizations. For instance, it helps prevent buffering issues and allows clients to access higher quality streams seamlessly. Moreover, shorter response times are essential for supporting mobile applications, disaster recovery scenarios, and various other use cases requiring swift and efficient content delivery.

1.4. Managing Clusters on the Edge-cloud Continuum

In IoT and Edge Computing environments, resources are limited and heterogeneous. Choosing technologies that provide the best experience in relation to the orchestration and execution of the proposed services is one of the main points of attention in this scenario. This section will cover the technologies used to accomplish this in the context of this short course, highlighting the container approach, its orchestration through Kubernetes and how both were used to implement the Edge Computing infrastructure used.

1.4.1. Introducing Containers

Containerization is a lightweight virtualization technology that enables the encapsulation of services and applications in small standard units. Unlike traditional machine virtualization, where a virtual machine is instantiated with the entire operating system, libraries and other dependencies, the container encompasses only the application (or service) of interest and the dependencies necessary for its execution. Scalability is enhanced using this approach, as containers consume less resources and generate lower overheads when compared to virtual machines. Each container is defined from an image, built on a stacked layer structure, where each change in the subsequent layer creates a new layer. Except for the topmost layer, all the others are read-only and can be reused to compose other images. Containers share the same kernel and isolate application processes from the rest of the operating system.

In this context, Docker[10], CoreOS and other interested parties in the container industry established the Open Container Initiative (OCI) with the purpose of defining standard specifications for container technologies. Currently, OCI proposes 3 standards: runtime, image and container distribution. Therefore, OCI allows compatibility among all artifacts produced in compliance with the proposed standards. Among the use cases related to comply with OCI, we highlight Internet of Things and Edge Computing, which are closely related to this short course.

The container technology stands out for offering distinguished characteristics in relation to other virtualization approaches, namely: portability, volatility, and resource consumption. Portability is evident in the way containers are built, as the application is encapsulated with all the dependencies necessary for its execution, a container will have the same execution behavior in any environment in which it is provisioned. Regarding volatility, due to the ephemeral nature of the container, all possible data persistence is carried outside the container. Therefore, in case of execution issues related to a failed container, it is not necessary to carry out debugging to identify the fundamental problem affecting its functioning, but rather to destroy it and create a new instance without the need to take care of application data, which can be costly. Finally, as previously described, as it is an environment with a small number of additional processes and services, as opposed to virtual machines, the container consumes fewer resources, optimizing the physical resources available on the host system.

Fig.1.4 shows the main differences between containers in relation to virtualization based on virtual machines.

[10]Docker is a container platform: `https://www.docker.com/`

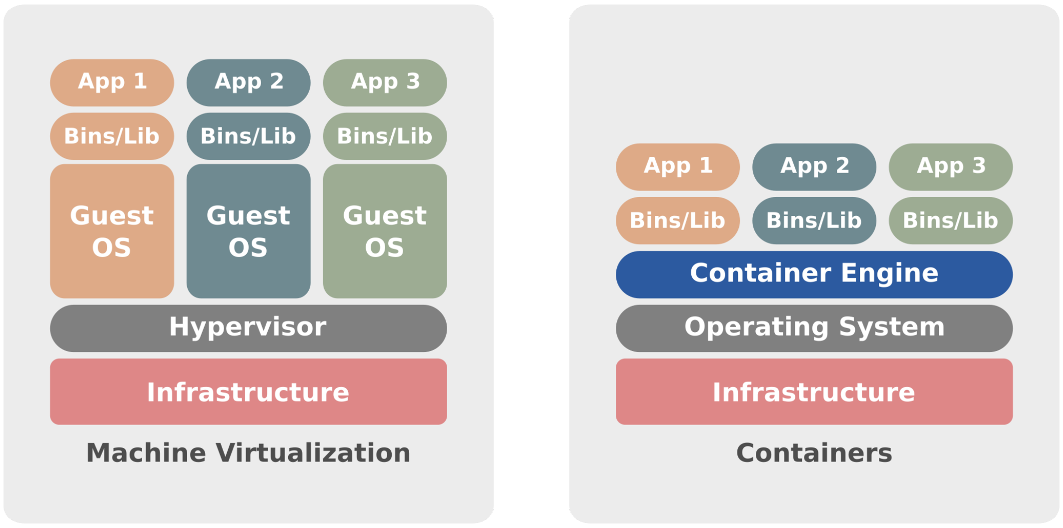

Figure 1.4. Containers vs Virtualization based on Virtual Machines.

It is important to note that running containers also require a runtime, the most famous of which is Docker. Container runtimes can be classified into two types: low-level and high-level. Low-level runtimes adhere to OCI specifications. Among them are runC, crun, runhcs, and containerd. On the other hand, there are runtimes that have an extra layer, because they present more functionalities than the minimum enough for just running containers. Examples of these high-level runtimes are Docker, Podman and CRI-O. For a runtime to execute an image of interest, a container image should be passed to the runtime. In the context of containers, images are stored in repositories that are referenced to the runtime at execution time. These repositories, also known as Registries, are well-structured and can be public or private. In the first case, the stored images are open to the public. In the second case, the images are made available to a specific audience, for example, one company employee; therefore, most of the time, they are available only through access credentials. Container image repository implementations include Red Hat's Harbor and Quay.

Containers enable the creation of a new application development paradigm based on microservices. This paradigm enables splitting application components into a number of relatively small services that, unlike paradigms with monolithic approaches, address the architecture and organization of applications by dividing the whole implementation into small, well-defined, and concise units according to the functions they perform in the application. The microservices communication is performed through well-defined APIs.

1.4.2. Managing Containers Lifecycle with Container Orchestrators

Containers are virtualized environments that offer several possibilities for use. Even though they offer a set of positive characteristics, they also bring challenges in managing their entire lifecycle in a simple way. In addition, containers alone do not deliver all the properties expected in the micro-services paradigm, being just one component that supports the paradigm. Properties such as elasticity and resilience are not delivered by container runtimes alone. To fill these gaps, orchestrators emerged, such as Docker Swarm, for example, to support containers lifecycle management (creation, replicas, garbage collector, removal of containers, etc.), among the orchestrators. A container orchestrator that has established itself as a market standard due to its versatility and effectiveness is Kubernetes[11].

The Kubernetes container orchestrator is an open-source project hosted by the Cloud Native Computing Foundation (CNCF). It is a well-documented orchestrator with an active community of developers and a growing set of funding entities (IBM, Google, Oracle, among others). There are currently several Kubernetes distributions, in which companies insert a set of functionalities into Vanilla Kubernetes (community version hosted at CNCF), and deliver them to interested parties through usage subscriptions. The main available Kubernetes distributions are Red Hat, Rancher, and VMware Tanzu.

When it comes to deployment architectures, Kubernetes is a highly distributed and flexible orchestrator. Composed of *Nodes*, it allows the orchestrator deployment to have specialized Nodes, making the organization as complex as needed; on the other hand, the same Kubernetes can be instantiated in an all-in approach – with just one Node assuming all responsibilities for orchestrating the environment. Among the types of Nodes provided by Kubernetes, two are considered fundamental and are present in most deployment architectures, which are the Worker Nodes and Control Nodes. Worker Nodes contain the computing resources that will be consumed by user workloads, while Control Nodes have the components related to the control of the entire environment and the orchestration services themselves (scheduling, for example). An interesting feature of Kubernetes is that even if the Control Nodes become unavailable, the workloads running on the Worker Nodes will continue to be active (even though with some restrictions). As it is organized into Nodes, the insertion of new Nodes, both workers (more resources) and Control, is a trivial task.

In the context of the workloads, unlike simple container runtime environments, in Kubernetes the smallest processing unit is a POD. A POD comprises one or more containers and a set of metadata that defines parameters related to the containers themselves (labels, the quantity of replicas, the quantity of resources required for execution, etc.), as well as the orchestrator's behavior over the POD. Considering the microservices paradigm, the POD corresponds to a microservice. A notable feature of PODs is that all containers in a POD share the same address space, so they can communicate through the localhost. Furthermore, this behavior allows the creation of some design patterns: the sidecar. This pattern allows the user to provide an auxiliary container that complements or adapts certain functionalities of the main container (the microservice). For example, given a microservice that logs the events of its execution to *syslog*, but the log aggregation

[11]Kubernetes is a container-orchestrator: `https://kubernetes.io/`

software used does not understand this protocol, in this scenario it would be possible to implement a sidecar to receive the syslog protocol on localhost, translating the protocol and sending it to the observability environment.

Kubernetes approaches the maintenance of the desirable state and the theory of controls as the core of its activities on PODs. Through these two approaches, Kubernetes, in control cycles (reconciliation cycle), checks whether the PODs running in the environment are in the desirable state. The respective desired states of each POD are defined through a structure called replica set. This object allows the indication of how many replicas of the POD of interest should exist at a given time. For example, if a replica set of a POD A is indicated as 2, two instances of POD A will be running. If a problem occurs and one of the replicas becomes unavailable, in the next reconciliation cycle a new POD A is automatically instantiated. In addition to the replica set, there are other objects that are defined around the POD and allow controlling the behavior regarding the deployment and execution of PODs, namely Deployment, StatefulSet, and DaemonSet. All of them use the Replica Set in their operation. Deployment operates mainly at the replica layer, and it can indicate the volumes used, number of POD replicas, among other properties. In addition to the properties and resources defined in Deployment, StatefulSet addresses a shortcoming in container environments, namely ephemerality. In StatefulSet, Kubernetes persists both network and volume address (resource location). Therefore, if a StatefulSet POD becomes unavailable, when Kubernetes instantiates it again (reconciliation cycle), it returns to using the network and volumes used by the POD that became unavailable. Finally, the Daemonset also offers the properties and resources defined in Deployment. Added to this, this object guarantees that each Kubernetes Node will receive at least one instance of the POD of interest. Through this approach, a POD can interact directly with the Kubernetes Node environment. A classic use case for Daemonset is to serve as a deployment model for log event collectors.

Kubernetes is intended to deliver complex tasks through trivial definitions. In this brief description of the main features it is clear that the use of Kubernetes is currently essential in the context of the use of containers.

1.4.3. Setting up a Kubernetes-based Cluster for Edge-Cloud Infrastructures

The edge-cloud computing continuum is a structure of heterogeneous devices (large capacity servers, access points, repeaters, among others) that can be arranged hierarchically in layers. The hierarchical distribution can include n-tiers, with a spectrum that goes from the upmost level (where the public or private cloud is located) to the bottommost layer. Considering a bottom-up approach, the bottommost layer contains the devices that end users interact with (smart TVs, smartphones, tablets, laptops, etc.). The intermediary tiers are composed of a collection of servers placed close to end-user devices, running back-end services close to data sources. Finally, the topmost layer are the cloud platforms, where all the resource-intensive back-end services run, exploiting the cloud platform's illusion of infinite computing resources available (e.g. storage, CPU, network , etc.).

A characteristic of the hierarchical structures of the edge-cloud computing continuum is that each layer devices are subject to their respective network latency and computational capacity when communicating with other devices. Lower layer devices, as they are closer to end users, are subject to lower latencies when communicating with end users. Conversely, devices in the highest layer are subject to larger latencies resulting from networking devices processing and queueing as well as link medium propagation delays, as they are further away from the end user. Considering resource capacity, devices closer to end users often have limited computing resources, and as we move upwards through the layers, we have devices with more capacity available. Therefore, properly managing the balance between network latency and resource availability is crucial to supporting applications that demand a high-quality end-user experience.

The balance between network latency and computing resources is often achieved through service allocation and offloading strategies. As already mentioned, as these are heterogeneous layers considering the latency and computational capacity of the devices, to support the allocation of services in the different layers, lightweight and flexible execution environments are often used. In this scenario, containers and Kubernetes play a fundamental role in this type of environment.

1.4.3.1. Reference Infrastructure Model

To demonstrate the flexibility of using containers and the Kubernetes orchestrator in the edge-cloud computing continuum scenario, a reference infrastructure model was created consisting of three layers structured in a binary tree topology. The root of the tree is in the first layer, being composed of a node representing the cloud. In the second, intermediate layer, two nodes are present. And finally, in the lower layer, four nodes are present (Fig. 1.5). In this last layer, network latency is minimal when accessed by users. This topology ensures that nodes can only interact with their child nodes or parent nodes, not having access to other nodes in the cluster. This arrangement makes it possible to adequately simulate the hierarchy formed in real edge-cloud computing continuum infrastructures. It is possible to use virtualized environments to emulate devices in a real environment. In this reference model, infrastructure resources were deployed on a cloud server managed by OpenStack.

Kubernetes is the container orchestrator of choice for the reference model. Each of the tier nodes is managed by a Kubernetes instance. These instances are composed of an all-in model, with the Master and Worker operating in a single unit. Therefore, Kubernetes is responsible for container deployment and container lifecycle management – services deployed in this environment run inside Docker containers. Additionally, tools such as Prometheus are used to monitor consumed and available resources and node service analysis. Surveys collected by Prometheus are made available to application-level services to decide how to adapt application services to better accommodate workload volume, infrastructure resource allocation, and end-user mobility.

The reference infrastructure model reflects the structure presented in the topology diagram presented in the above-mentioned figure. There are seven nodes organized hierarchically in a tree. Some layer 2 nodes interfaces are used only when installing the environment (for example, to download packages and container images). We do not need external access to these nodes during the execution of the experiments. So, the networks interfaces from this nodes (1 and 2 - layer 2) must be removed after installing the environ-

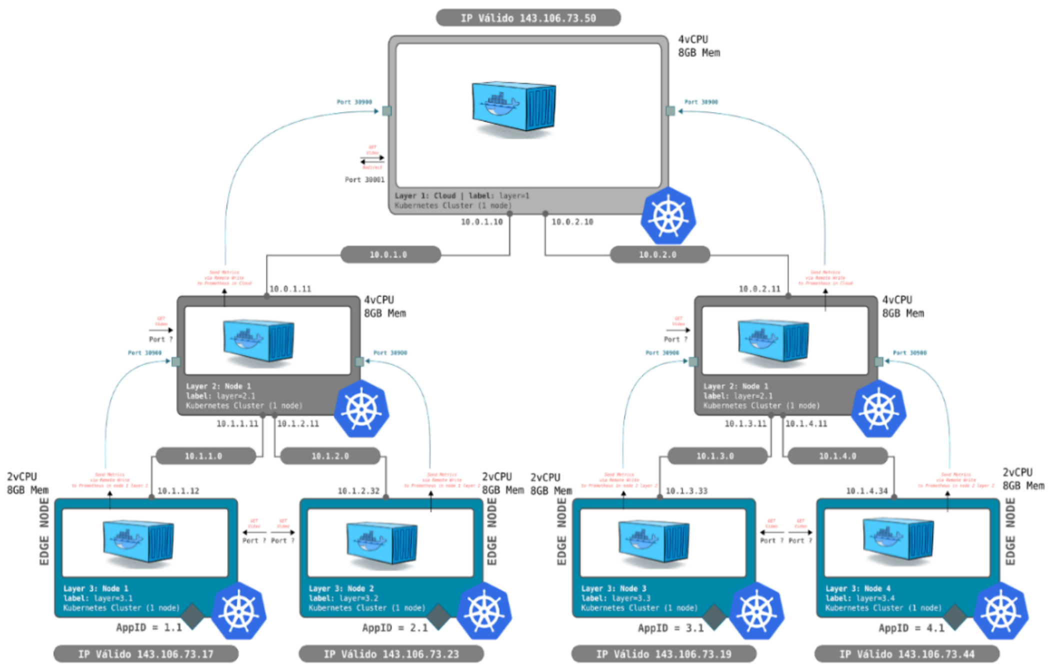

Figure 1.5. Topology diagram.

ment. In this model, two deployment and installation tools were mainly used: Terraform and Kubespray. Terraform was used to prepare virtual machines and networks in the OpenStack environment. Kubespray was used to install a Kubernetes cluster on previously created virtual machines. It forms a total of 7 virtual machines, each of which runs an all-in Kubernetes cluster instance.

Terraform is an open-source tool offered by HashiCorp. This tool allows the implementation of infrastructure environments described as code, i.e. Infrastructure as Code (IaC). Terraform encapsulates all the logic and APIs of the environment of interest and describes it as a provider. Such providers are used to make API calls abstractly to the implemented code. This makes the maintenance, support and migration process to other providers easier with Terraform. In the test scenario, the OpenStack provider and Terraform were used to configure and instantiate all the necessary resources.

To start the Terraform environment, the following steps should be followed where ten files will be created:

```
$ cd terraform/

## Download files related to the OpenStack provider:
$ terraform init

## Create the action plan. All changes to the OpenStack
## environment will be displayed here:
```

```
$ terraform plan

## Execute the previously created action plan:
$ terraform apply
```

- **00-variable.tf**: file with the definitions of the variables that will be used by the other resources. In this file, the value of FLOATING_IP_POOL must be changed with the floating pool of IPs that the infrastructure is using:

```
variable "floating_ip_pool" {
  type    = string
  default = "FLOATING_IP_POOL"
}
```

- **10-ssh-key.tf**: defines the public key (rsa type) that will be used to access the created virtual machines instances. In the entry below, the value of PUBLIC_KEY must be set to a public key.

```
resource "openstack_compute_keypair_v2" "user_key" {
  name       = "access-key"
  public_key = "PUBLIC_KEY"
}
```

- **20-network.tf**: file with definitions about the network aspects used in the rest of the environment.

- **30-security-groups.tf**: security groups used. Initially, all ports are closed. In OpenStack (provider used), opening is carried out through Security Groups.

- **40-flavor.tf**: definition of the set of resources used in virtual machines (flavor). This definition is made in terms of vCPU, memory and disk.

- **50-image.tf**: operating system image used in the environment's virtual machines. Some images presented in OpenStack providers have a limitation regarding network interfaces (1 in this case). This behavior can lead to network availability issues when more than one interface is configured. Neutron (Openstack's network service) tries to configure the presented interfaces, indicated by Terraform, but only the first one is defined successfully. It is recommended to use more modern OS images that allow the configuration of multiple network interfaces (e.g. Almalinux 8, CentOS, etc.).

- **60-instances.tf**: definitions of VM instances and security groups. Floating IPs (previously allocated in the Openstack infrastructure) are also configured. Regarding floating IPs, if they are not present and need to be allocated on demand, it is necessary to uncomment the resource below (inserted in the file).

```
#resource "openstack_networking_floatingip_v2" "fip_pool" {
#   count  = 7
#   pool   = var.floating_ip_pool
#   description = "Floating IP to project Ericson"
#}
```

As previously described, 7 nodes will be provisioned. Therefore, 7 floating IPs are needed for the initial configuration of the environment. Consequently, 7 entries like the one shown below must be present for the respective nodes.

```
resource "openstack_compute_floatingip_associate_v2"
"fip_associate_node11" {
   floating_ip = "177.220.85.224"
   instance_id = openstack_compute_instance_v2.vm_node_11.id
   fixed_ip    = "192.168.200.210"
}
```

- **provider.tf**: defines the provider that will be used (OpenStack). This provider's credentials can be defined in the file itself or in the cloud.yaml file.
- **clouds.yaml**: Settings and access credentials for the Openstack provider, used to provide the reference model infrastructure. This file can be obtained directly from the Openstack graphical interface (Horizon Dashboard).
- **versions.tf**: Defines the Terraform versions that will be used.

Kubespray: is an open source tool intended for deploying and managing Kubernetes clusters. The tool works with public cloud, on-premises, bare metal, and staging solutions, making it ideal for managing highly available clusters across multiple different platforms. It is comprised of Ansible playbooks, inventory, provisioning tools, and domain knowledge for generic OS/Kubernetes cluster configuration management tasks. Kubespray in the context of the reference model is used to provision 7 clusters. For each cluster there is an inventory with its respective configuration.

- **Node Cloud**: cluster-root/
- **Node 1 layer 2**: cluster-node2-1/
- **Node 2 layer 2**: cluster-node2-2/
- **Node 1 layer 3**: cluster-node3-1/
- **Node 2 layer 3**: cluster-node3-2/
- **Node 3 layer 3**: cluster-node3-3/
- **Node 4 layer 3**: cluster-node3-4/

After installing kubespray and its dependencies, each folders from the list above must be copied to *invetory/*. Preparing the environment for running kubespray is as follows:

```
## Clone the repository:
$ git clone https://github.com/kubernetes-sigs/kubespray.git
$ cd kubespray

```

```
## Copy the kubernetes cluster inventory files to the inventory
folder. REPOSITORY is set with this git project folder:
$ cp -r ${REPOSITORY}/artefacts/kubespray/cluster-* ./invetory/

## Create virtual environment:
$ python3 -m venv venv

## Activate the environment:
$ source venv/bin/activate

## Install the dependencies:
$ pip install -r requirements.txt
```

It is necessary to configure the network addresses as defined in Terraform. Below is an example configuration.

```
all:
  hosts:
    node1:
      ansible_host: 177.220.85.224
      ip: 192.168.200.210
      access_ip: 192.168.200.210
  children:
    kube_control_plane:
      hosts:
        node1:
    kube_node:
      hosts:
        node1:
    etcd:
      hosts:
        node1:
    k8s_cluster:
      children:
        kube_control_plane:
        kube_node:
    calico_rr:
      hosts: {}
```

Ansible_host must be defined with the floating IP address of the virtual machine that Kubespray (Ansible components) will access to perform the installation of the kubernetes node. The other two addresses (*ip* and *accessip*) must be of the internal interface, in this case the interface that is associated with the floating IP. Another necessary configuration is to define the docker repositories where the images will be obtained. This must be done in *inventory/CLUSTER_NODE/group_vars/all/containerd.yml*.

The following entries must be inserted at the end of the file:

```
(...)
containerd_registries:
  "docker.io":
    - "https://mirror.gcr.io"
    - "https://registry-1.docker.io"
\end{verbatim}

```

```
Once all configurations have been made, Kubespray must be run for each
   node of interest.
\begin{verbatim}
$ ansible-playbook -i inventory/cluster-root/hosts.yaml -u
almalinux -b -v --private-key="${KEYS_FOLDER}/id_rsa" cluster.yml

-i: inventory with de node cluster address, quantity of nodes,
kubernetes version and other configurations.
-u: user used by OS to execute the activities.
-b: become a root if necessary.
-v: verbose mode.
--private-key: file with private key to access the VM
```

After running Kubespray for each new node, it is necessary to verify that the environment was correctly installed. This must be performed for each of the Kubernetes clusters created. Within each virtual machine in the reference model, it is initially necessary to obtain the Kubernetes credentials, copy them to an appropriate location, and verify that the cluster is operational (1 node and the operational services PODs should be ready).

```
## Create the Kubernetes config file:
$ mkdir .kube
$ sudo cp /etc/kubernetes/admin.conf .kube/config
$ sudo chmod 666 .kube/config

## Check the kubernetes master node:
$ kubectl get nodes
NAME    STATUS   ROLES           AGE   VERSION
node1   Ready    control-plane   18h   v1.25.5

## Check the kubernetes system PODs:
$ kubectl get pods -n kube-system
NAME                                       READY   STATUS    REST-
calico-kube-controllers-75748cc9fd-ttdm8   1/1     Running   0
calico-node-z8cg9                          1/1     Running   0
coredns-588bb58b94-k2qrs                   1/1     Running   0
dns-autoscaler-d8bd87bcc-djvm8             1/1     Running   0
kube-apiserver-node1                       1/1     Running   1
kube-controller-manager-node1              1/1     Running   2
kube-proxy-t4b97                           1/1     Running   0
kube-scheduler-node1                       1/1     Running   2
nodelocaldns-9vk68                         1/1     Running   0
```

Some other settings may be needed, such as checking and setting host names and inserting them in */etc/hosts* since, there is no DNS server for the environment. It may also require some OS tuning. These details are beyond the scope of this document.

1.5. Adaptive Video Streaming Application for the Edge-cloud Continuum

To enhance the readers' understanding of the framework behind the content steering service distributed across the Edge-Cloud Continuum, we present a detailed case study focusing on a video streaming service hosted on a CDN platform within the Institute of Computing (IC) at UNICAMP. We aim to illustrate alternative video provisioning points for users. To achieve this, we examine a scenario where the CDN, hosted in the Cloud, redirects users to an edge server on the local machine. Various factors could prompt this redirection, including recurrent congestion issues within users' home networks.

Fig. 1.6 illustrates the basic elements that a fully functioning video streaming application running on the edge-cloud continuum contemplates. We provide all necessary software with detailed instructions for the reader to execute the system on their machine. Moreover, the interested reader may use the available system to conduct their own experiments and analysis.

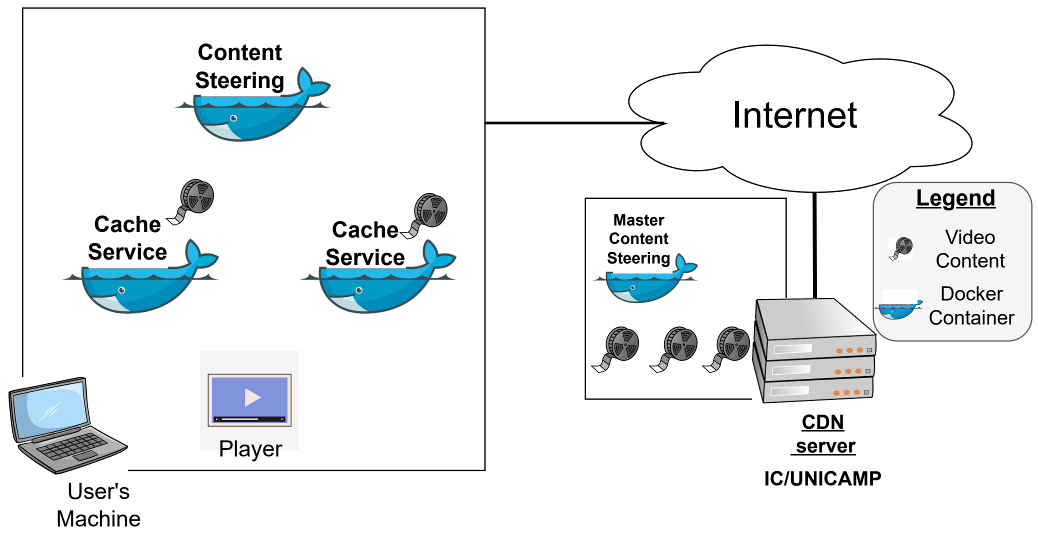

Figure 1.6. Main elements of the video streaming application. A local machine running a video player, two cache services and a content steering, connected to a CDN with video content and a master content steering.

The video streaming application we provide has four main basic components: i) the CDN servers where the main content and the master content steering are hosted, ii) the cache servers running on the reader's machine, as an illustration of cache servers deployed in multiple locations over the edge of the network, iii) the edge content steering module responsible for steering video content in a specific location of the edge-cloud continuum, and iv) the user's video player, that in this case also runs on the user's machine.

The CDN servers are placed in the IC-Unicamp dependencies, and their content should be accessible to the reader. Note that any CDN can be used as a replacement for the one we have set up to demonstrate our video streaming application. We provide a detailed description of how to set up and configure a CDN in case the reader wants to use their own settings. The CDN hosts both the video content and an executing master content steering service. The client's video player always contacts the master content steering and collects content from the CDN in the first round, later it connects to the local content steering and starts fetching content from the local cache service.

The cache services are provided in the form of Docker container images to facilitate the system's execution. The cache services in our scenario are different points for content provision. Thus, the video player can fetch video content directly from the cache services in the same network instead of fetching content from a distant CDN server.

We also make available a content steering service that should be executed on the reader's local machine. We provide both the Docker image and the source code. This content steering service is responsible for steering the user's video player to specific cache services in the reader's machine. In our video streaming systems example, this content steering represents the module responsible for adapting and handling content steering in a specific geographic location in a real-life deployment in a full-scale edge-cloud continuum infrastructure.

Moreover, the content steering module is also responsible for monitoring the cache service metrics and detecting when a specific cache service is down or not responding. When this occurs, the content steering redirects the video player to fetch information from another cache service or the closest CDN server. In our practical experiment, we can observe such behavior after deploying the system and stopping the Docker container running the cache service that provides the content to the video player.

Finally, we provide a fully functioning video player that implements the content steering protocol. The video player accesses the CDN server to fetch the manifest file with the information on where to fetch the video content. After the first interaction with the CDN node, the video player receives a URL for a local cache service and fetches the video content from the local cache service.

1.5.1. Repository Description

The complete materials, algorithms, tools, and tutorials for installing and running the application can be accessed on Github [12]. To obtain the code and materials, you must perform a process named clone, which downloads the source code. We recommend using Ubuntu version 22.04 since the setup instructions are tailored for it.

After the cloning process, a directory named content-steering-tutorial should have been created. From the repository's root directory, folders for the video streaming and steering services and a dataset folder containing the multimedia content should be created. To streamline the execution of experiments, we provide some shell scripts to automate the process.

1.5.2. Running the Streaming Services

Now we are initiating video streaming and steering microservices. It's assumed that the reader is familiar with Docker. To start the video streaming containers, we create two replicas of the service using the command:

```
$ docker compose -f steering-service/docker-compose.yml up -d
```

Listing 1.4. Command to create the streaming services.

If you wish to configure the number of replicas, modify the replica count within the docker-compose.yml file in the streaming-service directory. Now, ascertain the IPs assigned to each service and associate them with the domains created in /etc/hosts.

You can then verify the features of local video streaming. Remember that our focus is on Content Steering features. Therefore, our experiments use a straightforward

[12] `https://github.com/robertovrf/content-steering-tutorial`

```
$ docker compose -f steering-service/docker-compose.yml up -d
```

Listing 1.6. Command to create the steering service.

HTTP server to serve static multimedia content. This approach allows us to observe the services functioning autonomously, independent of others. Then, you can access the *dash.js* player and try to load the manifest using the URL format:

```
https://<streaming-domain>/<streaming-path>/manifest.mpd
```

Listing 1.5. URL to watch the video in any Dash player.

Replace <*streaming-domain*> with your streaming domain and <*streaming-path*> with the path to your video.

1.5.3. Running the Steering Service

To create the Content Steering service, we follow similar commands to those used for the streaming service setup. Within the steering-service folder, you find the *docker-compose.yml* file containing the default configurations for local steering creation.

The Content Steering mechanism involves real-time monitoring of deployed containers. Various tools for resource monitoring within containers have been proposed, including docker stats, Prometheus, and Google's cAdvisor. In this tutorial, we utilize the docker package to collect resource usage metrics from containers and determine the availability of video streaming services. The steering collects the Docker stats metrics from the containers with running stats every two seconds. It is recommended that the reader investigate the *steering-service/src/monitoring.py* code to understand metrics capture.

The communication to the steering service occurs via an HTTP GET method, as discussed in Section 1.3. We have set up a basic Flask application with a single route handling GET requests at the root URL, running on port 30500. Listing 1.7 provides the code snippet responsible for managing these requests (*steering-service/src/app.py*). When a user initiates a request to the content steering, the service identifies active edge servers using the *getNodes()* method from monitoring in line 7. Subsequently, the algorithm updates the list with the selected edge servers and the target server, presenting them as the output for the user's request. The parser method constructs the message (lines 9-14). The return incorporates essential components such as the TTL and the RELOAD-URI, which specify the URI for reloading the steering service's configuration as needed. Finally, we use the jsonify function to convert the message into JSON.

```
1  @app.route('/', methods=['GET'])
2  def do_get(name):
3
4  trg = request.args.get('_DASH_pathway', default='', type=str)
5  thr = request.args.get('_DASH_throughput', default=0.0, type=float)
6
7  nodes = _monitoring.getNodes()
8
9  message = _dash.parser(
10     target  = trg,
$ ./create_certs.sh <streaming-domain>
```

Listing 1.8. Command to generate certificates.

```
    nodes   = nodes,
    uri     = BASE_URI,
    request = request
)

return jsonify(message), 200
```

Listing 1.7. Main method from the steering service.

1.5.4. Running the Testbed

In the initial experiment, the localhost environment was set up using Docker tools. Start by modifying your local host's file, found at */etc/hosts*, to assign custom domain names for the streaming services you intend to create. After that, execute the script *create_certs.sh* to generate certificates and activate HTTPS for each streaming service in localhost:

Next, retrieve a DASH video from the dataset provided by [64] and store it in the specified folder named "dataset". The dataset offers a range of codec choices, including AV1, AVC, HEVC, and VVC. You may utilize wget or any other suitable method to download the video. We used the "Eldorado" video encoded in AVC with 4-second segments for our experiments.

1.5.5. Running the Video Player

We can test our experiment once we have deployed the necessary components mentioned earlier. Our next step would be to request a video from a scenario where the CDN, located in the Cloud, serves as the initial source. As the user watches the video, they can switch between servers autonomously. This enables us to evaluate our edge servers' efficiency and effectiveness in delivering high-quality content.

To gain a better understanding of how the transition from the cloud-based content source to the edge servers works, we can utilize the player available at [13]. This player lets us visualize how the user's playback experience changes as they switch between servers.

1.6. Challenges and Opportunities

As we can now create and configure a Kubernetes-based cluster over the edge-cloud continuum, deploy and perform (re)placement of video streaming services over the infrastructure to improve the quality of experience, we start facing new and interesting challenges.

In this section, we discuss three of the main challenges we consider the most important: content steering optimization, the concept of zero-touch networks and their related challenges, and the issues that arise when considering user mobility.

[13]Content Steering - `https://reference.dashif.org/dash.js/latest/samples/advanced/content-steering.html`

1.6.1. Content-Steering Optimization

The content steering problem is an interesting and challenging problem that needs to be addressed. Especially in the edge-cloud continuum settings, where content steering has to handle high volumes of video request content. Thus, the content steering optimization problem can be formulated as follows:

Problem: *Given a set of users requesting video content and a set of CDN servers, how to best steer user requests to load balance them among the available servers considering user and server location, network congestion, video quality, and network bandwidth to maximize user quality of experience.*

A version of this problem has been tackled in the literature. Gama et al. [29] has presented the challenge and provided an Integer Linear Programming (ILP)-based solution. In the context of video streaming over edge-cloud continuum infrastructures, however, it is crucial to consider the volatile and dynamic nature of edge-cloud continuum infrastructures, where changes constantly and unexpectedly occur at the infrastructure, platform, and service levels and based on user behavior and request patterns. In such scenarios, the content steering policy needs to be constantly updated.

Therefore, static optimization solutions for content steering that are defined pre-deployment are not sufficient. The adoption of models to drive content steering is required to evolve over time to include changes in content popularity, user behavior, varying request volumes, and sudden network congestion, to name a few sources of unexpected events.

The application of machine learning, specifically reinforcement learning strategies, is desired in such volatile settings. This is because reinforcement learning techniques define their models as the agent interacts with the environment; in this case, the content steering orchestrator defines which CDN server the users will fetch the content from. This is crucial in situations where the content steering orchestrator encounters unexpected scenarios where it is required to optimally load balance the incoming requests to the available CDN servers, for instance, a CDN server becomes unavailable or there is a sudden spike in requests in a specific geographic location.

An interesting line of research is to enable autonomous learning for both content steering policy and content steering orchestrator placement. This would enable the definition of where in the edge-cloud continuum to place a content steering orchestrator to content steering for a set of users while, at the same time, enabling the recently placed orchestrator to learn the best policy to steer incoming requests.

Autonomous learning solutions, such as the one outlined here, could help tame the increasing complexity of exploiting the underlying edge-cloud continuum resources while supporting the executing services for video streaming applications.

1.6.2. Zero-touch Network and Service Management (ZSM)

The application of closed-loop automation to bootstrap, configure, and optimize networking systems led to the definition of the concept of Zero-touch Networks and Service Management (ZSM) [40, 62]. Many research papers that address the concept of the zero-touch network come from the 5G/6G communities [9, 62, 18], given the rising complexity in setting up, configuring, and optimizing such networks.

As an important line of research, we have the application of closed-loop automation to set up, configure, and optimize video streaming services, platforms, and the underlying networking infrastructure to support video streaming applications in edge-cloud continuum settings. Besides the application of machine learning techniques to solve the content steering problem (Sec. 1.6.1), video streaming could benefit from autonomous (self-adaptive, self-optimizing, self-healing) techniques. They can be used to autonomously set up, perform placement, and optimize cache services throughout the edge-cloud continuum, even autonomously choosing the cache replacement strategy that best suits the cache service based on the handled requests.

Furthermore, the application of such techniques can autonomously define parameters in the content steering architecture. For example, adjust the parameter that defines how often players provide information to content steering orchestrators to make decisions on which server the player should fetch information. These parameters are currently set manually, and it has an impact on how fast the system adapts to avoid networking congestion as well as on the amount of information content steering orchestrator receives in a timespan.

A complementary line of research is to apply Intent-based Networks (IBN) [37] and Intent-driven Networks (IDN) [43, 23] concepts to guide zero-touch network optimization and service management. As we apply autonomous mechanisms to optimize the network, the use of high-level network intents defined by both application developers and network administrators can be used as network goals for the closed-loop automation mechanism to satisfy with minimum to no human intervention.

Examples of applying Intent-based Network techniques are described in many different networking contexts, such as IoT network management [17], 5G cloud service provision [3], and video service assurance [52]. Therefore, exploring similar techniques in the edge-cloud continuum to set up and optimize video streaming applications presents a fruitful direction for future research.

1.6.3. Exploring Adaptation to Accommodate User Mobility

An important challenge to tackle in this domain is to address user mobility. As users move from location *A* to location *B*, many changes are required to take place throughout the infrastructure, platform, and service level to maintain the quality of streaming.

User mobility has been a concern for video streaming for a while now, with papers proposing solutions to cope with mobility dating back to 3G networks [36]. The problem with handling mobility has not been fully addressed but rather intensified over the years. Despite the fact that networking infrastructures are becoming faster with 5G deployments and the expectations of 6G networks, the volume of streaming content, their unprecedented quality, and the number of users continue to increase, aggravating the problems related to video streaming services in the context of user mobility.

Solutions considering caching content on the edge [14] and/or attempting to predict user mobility [68] have been explored in the literature [35]. However, there are still opportunities to explore these approaches at the scale on which edge-cloud continuum infrastructures operate while streaming high-quality videos with an incredibly high number of mobile users consuming video content.

An interesting line of research is the exploration of machine learning approaches to identify and predict user mobility patterns and execute video prefetching to cache services deployed close to their location. Moreover, the combination of prefetching solutions based on user mobility patterns in tandem with data placement considering content popularity amongst groups of users moving towards the same direction may increase the quality of streaming and optimize computing resource allocation.

Finally, the exploration of prefetching solutions and handling unexpected user behavior is also an interesting research direction, especially considering the volatile nature of the edge-cloud continuum infrastructure. In the case of user mobility, some users may detour from their expected predefined route and change mobility patterns from time to time. In these cases, reactive approaches to deal with unexpected situations are desired and expected to work in tandem with proactive approaches, where prefetching/preventive adaptive strategies are in place.

1.7. Conclusion

This chapter presented an overview of the technologies and concepts involved in supporting video streaming over edge-cloud continuum infrastructures. In particular, we have introduced the concept of edge-cloud continuum and discussed the main challenges and opportunities. The discussions included the infrastructures provided, the main video streaming protocols, the concept of the content steering architecture, and the technology used to deploy and configure platform-level management tools to control the underlying computing resources. It was also presented a hands-on practical experiment with detailed instructions to execute a fully functioning video streaming application using content steering.

We hope to have contributed to the community by supplying a starting point for researchers to explore the diversity of challenges presented in deploying and managing video streaming applications over the Edge-cloud Continuum. We also hope that the technical material we supply on our GitHub [14] repository will facilitate the creation of testbeds for further experiments with content steering architecture and video streaming applications on the edge-cloud platforms.

Acknowledgements

This work was partially supported by the São Paulo Research Foundation (FAPESP), grant 2021/00199-8, CPE SMARTNESS, and by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

References

[1] Content steering for dash, 2022. Accessed 28-oct-2023.

[14]GitHub: `https://github.com/robertovrf/content-steering-tutorial`

[2] A. T. Akabane, R. Immich, R. W. Pazzi, E. R. M. Madeira, and L. A. Villas. Exploiting vehicular social networks and dynamic clustering to enhance urban mobility management. *Sensors*, 19(16):3558, Aug 2019.

[3] F. Aklamanu, S. Randriamasy, E. Renault, I. Latif, and A. Hebbar. Intent-based real-time 5g cloud service provisioning. In *2018 IEEE Globecom Workshops (GC Wkshps)*, pages 1–6, 2018.

[4] F. Alessi, A. Tundo, M. Mobilio, O. Riganelli, and L. Mariani. Reprobes: An architecture for reconfigurable and adaptive probes, 2024.

[5] B. Alt, T. Ballard, R. Steinmetz, H. Koeppl, and A. Rizk. Cba: Contextual quality adaptation for adaptive bitrate video streaming. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pages 1000–1008, 2019.

[6] A. Bentaleb, A. C. Begen, S. Harous, and R. Zimmermann. Want to play dash? a game theoretic approach for adaptive streaming over http. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18, pages 13–26, New York, NY, USA, 2018. Association for Computing Machinery.

[7] A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, and R. Zimmermann. Bitrate adaptation and guidance with meta reinforcement learning. *IEEE Transactions on Mobile Computing*, (01):1–14, mar 5555.

[8] A. Bentaleb, B. Taani, A. C. Begen, C. Timmerer, and R. Zimmermann. A survey on bitrate adaptation schemes for streaming media over http. *IEEE Communications Surveys & Tutorials*, 21(1):562–585, 2019.

[9] C. Benzaid and T. Taleb. Ai-driven zero touch network and service management in 5g and beyond: Challenges and research directions. *IEEE Network*, 34(2):186–194, 2020.

[10] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana. The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, 3-4:134–155, 2018.

[11] L. F. Bittencourt, A. Goldman, E. R. Madeira, N. L. da Fonseca, and R. Sakellariou. Scheduling in distributed systems: A cloud computing perspective. *Computer Science Review*, 30:31–54, 2018.

[12] R. Buyya, M. Pathan, and A. Vakali. *Content delivery networks*, volume 9. Springer Science & Business Media, 2008.

[13] S. Chaudhary, N. K. Shukla, P. Sachdeva, S. Chakraborty, and M. Maity. Managing connections by quic-tcp racing: A first look of streaming media performance over popular http/3 browsers. *IEEE Transactions on Network and Service Management*, pages 1–1, 2024.

[14] Y. Chen, H. Yu, B. Hu, Z. Duan, and G. Xue. An edge caching strategy based on user speed and content popularity for mobile video streaming. *Electronics*, 10(18), 2021.

[15] I. Cohen, C. F. Chiasserini, P. Giaccone, and G. Scalosub. Dynamic service provisioning in the edge-cloud continuum with bounded resources. *IEEE/ACM Transactions on Networking*, 31(6):3096–3111, 2023.

[16] V. Colombo, A. Tundo, M. Ciavotta, and L. Mariani. Towards self-adaptive peer-to-peer monitoring for fog environments. In *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '22, pages 156–166, New York, NY, USA, 2022. Association for Computing Machinery.

[17] B. M. Cordeiro, R. Rodrigues Filho, I. G. Júnior, and F. M. Costa. Steer: An architecture to support self-adaptive iot networks for indoor monitoring applications. *Journal of Internet Services and Applications*, 14(1):107–123, 2023.

[18] E. Coronado, R. Behravesh, T. Subramanya, A. FernÃ ndez-FernÃ ndez, M. S. Siddiqui, X. Costa-PÃ©rez, and R. Riggio. Zero touch management: A survey of network automation solutions for 5g and 6g networks. *IEEE Communications Surveys and Tutorials*, 24(4):2535–2578, 2022.

[19] A. Diaconescu, B. Porter, R. Rodrigues, and E. Pournaras. Hierarchical self-awareness and authority for scalable self-integrating systems. In *2018 IEEE 3rd International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 168–175. IEEE, 2018.

[20] J. Du, C. Jiang, A. Benslimane, S. Guo, and Y. Ren. Sdn-based resource allocation in edge and cloud computing systems: An evolutionary stackelberg differential game approach. *IEEE/ACM Transactions on Networking*, 30(4):1613–1628, 2022.

[21] W. Eddy. Transmission Control Protocol (TCP). RFC 9293, Aug. 2022.

[22] J. Ehlers, A. van Hoorn, J. Waller, and W. Hasselbring. Self-adaptive software system monitoring for performance anomaly localization. In *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ICAC '11, pages 197–200, New York, NY, USA, 2011. Association for Computing Machinery.

[23] Y. Elkhatib, G. Coulson, and G. Tyson. Charting an intent driven network. In *2017 13th International Conference on Network and Service Management (CNSM)*, pages 1–5, 2017.

[24] R. Farahani, M. Shojafar, C. Timmerer, F. Tashtarian, M. Ghanbari, and H. Hellwagner. Ararat: A collaborative edge-assisted framework for http adaptive video streaming. *IEEE Transactions on Network and Service Management*, 20(1):625–643, 2023.

[25] K. Fu, W. Zhang, Q. Chen, D. Zeng, and M. Guo. Adaptive resource efficient microservice deployment in cloud-edge continuum. *IEEE Transactions on Parallel and Distributed Systems*, 33(8):1825–1840, 2022.

[26] K. Fu, W. Zhang, Q. Chen, D. Zeng, X. Peng, W. Zheng, and M. Guo. Qos-aware and resource efficient microservice deployment in cloud-edge continuum. In *2021*

IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 932–941, 2021.

[27] E. S. Gama, L. O. N. De Araujo, R. Immich, and L. F. Bittencourt. Video streaming analysis in multi-tier edge-cloud networks. In *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 19–25, Aug 2021.

[28] E. S. Gama, R. Immich, and L. F. Bittencourt. Towards a multi-tier fog/cloud architecture for video streaming. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 13–14, 2018.

[29] E. S. Gama, N. B. V, R. Immich, and L. F. Bittencourt. An orchestrator architecture for multi-tier edge/cloud video streaming services. In *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*, pages 190–196, 2023.

[30] A. Goscinski and M. Bearman. Resource management in large distributed systems. *ACM SIGOPS Operating Systems Review*, 24(4):7–25, 1990.

[31] Z. Houmani, D. Balouek-Thomert, E. Caron, and M. Parashar. Enabling microservices management for deep learning applications across the edge-cloud continuum. In *2021 IEEE 33rd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 137–146, 2021.

[32] T. Huang and L. Sun. Deepmpc: A mixture abr approach via deep learning and mpc. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1231–1235, 2020.

[33] R. Immich, L. Villas, L. Bittencourt, and E. Madeira. Multi-tier edge-to-cloud architecture for adaptive video delivery. In *2019 7th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 23–30, Aug 2019.

[34] J. Iyengar and M. Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, May 2021.

[35] M. A. Khan, E. Baccour, Z. Chkirbene, A. Erbad, R. Hamila, M. Hamdi, and M. Gabbouj. A survey on mobile edge computing for video streaming: Opportunities and challenges. *IEEE Access*, 10:120514–120550, 2022.

[36] A. Kyriakidou, N. Karelos, and A. Delis. Video-streaming for fast moving users in 3g mobile networks. In *Proceedings of the 4th ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE'05, pages 65–72, New York, NY, USA, 2005. Association for Computing Machinery.

[37] A. Leivadeas and M. Falkner. A survey on intent-based networking. *IEEE Communications Surveys and Tutorials*, 25(1):625–655, 2023.

[38] E. Liotou, T. Hoßfeld, C. Moldovan, F. Metzger, D. Tsolkas, and N. Passas. *The Value of Context-Awareness in Bandwidth-Challenging HTTP Adaptive Streaming Scenarios*, pages 128–150. Springer International Publishing, Cham, 2018.

[39] D. Liu, Z. Wang, and J. Zhang. Video stream distribution scheme based on edge computing network and user interest content model. *IEEE Access*, 8:30734–30744, 2020.

[40] M. Liyanage, Q.-V. Pham, K. Dev, S. Bhattacharya, P. K. R. Maddikunta, T. R. Gadekallu, and G. Yenduri. A survey on zero touch network and service management (zsm) for 5g and beyond networks. *Journal of Network and Computer Applications*, 203:103362, 2022.

[41] S. Moreschini, F. Pecorelli, X. Li, S. Naz, D. Hastbacka, and D. Taibi. Cloud continuum: The definition. *IEEE Access*, 10:131876–131886, 2022.

[42] V. Nathan, V. Sivaraman, R. Addanki, M. Khani, P. Goyal, and M. Alizadeh. End-to-end transport for video qoe fairness. In *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, pages 408–423, New York, NY, USA, 2019. Association for Computing Machinery.

[43] L. Pang, C. Yang, D. Chen, Y. Song, and M. Guizani. A survey on intent-driven networks. *IEEE Access*, 8:22862–22873, 2020.

[44] R. Pantos and W. May. HTTP Live Streaming. RFC 8216, Aug. 2017.

[45] M. L. M. Peixoto, T. A. L. Genez, and L. F. Bittencourt. Hierarchical scheduling mechanisms in multi-level fog computing. *IEEE Transactions on Services Computing*, 15(5):2824–2837, 2022.

[46] L. Peng, A. R. Dhaini, and P.-H. Ho. Toward integrated cloud-fog networks for efficient iot provisioning: Key challenges and solutions. *Future Generation Computer Systems*, 88:606–613, 2018.

[47] L. Peroni and S. Gorinsky. An end-to-end pipeline perspective on video streaming in best-effort networks: A survey and tutorial, 2024.

[48] T. Pfandzelter and D. Bermbach. Enoki: Stateful distributed faas from edge to cloud. In *Proceedings of the 2nd International Workshop on Middleware for the Edge*, MiddleWEdge '23, pages 19–24, New York, NY, USA, 2023. Association for Computing Machinery.

[49] F. Pisani, F. de Oliveira, E. S. Gama, R. Immich, L. F. Bittencourt, and E. Borin. Fog computing on constrained devices: Paving the way for the future iot. *Advances in Edge Computing: Massive Parallel Processing and Applications*, 35:22, 2020.

[50] C. Puliafito, C. Cicconetti, M. Conti, E. Mingozzi, and A. Passarella. Balancing local vs. remote state allocation for micro-services in the cloud-edge continuum. *Pervasive and Mobile Computing*, 93:101808, 2023.

[51] C. Quadros, E. Cerqueira, A. Neto, A. Riker, R. Immich, and M. Curado. A mobile qoe architecture for heterogeneous multimedia wireless networks. In *2012 IEEE Globecom Workshops*, pages 1057–1061, Dec 2012.

[52] C. E. Rothenberg, D. A. Lachos Perez, N. F. Saraiva de Sousa, R. V. Rosa, R. U. Mustafa, M. T. Islam, and P. H. Gomes. Intent-based control loop for dash video service assurance using ml-based edge qoe estimation. In *2020 6th IEEE Conference on Network Softwarization (NetSoft)*, pages 353–355, 2020.

[53] F. Santos, R. Immich, and E. R. Madeira. Multimedia services placement algorithm for cloud-fog hierarchical environments. *Computer Communications*, 191:78–91, 2022.

[54] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[55] M. Seufert, R. Schatz, N. Wehner, and P. Casas. Quicker or not? an empirical analysis of quic vs tcp for video streaming qoe provisioning. In *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 7–12, 2019.

[56] K. R. Sheshadri and J. Lakshmi. Qos aware faas for heterogeneous edge-cloud continuum. In *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pages 70–80, 2022.

[57] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[58] C. Sicari, D. Balouek, M. Parashar, and M. Villari. Event-driven faas workflows for enabling iot data processing at the cloud edge continuum. In *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*, UCC '23, New York, NY, USA, 2024. Association for Computing Machinery.

[59] D. Silhavy, W. Law, S. Pham, A. C. Begen, A. Giladi, and A. Balk. Dynamic cdn switching-dash-if content steering in dash. js. In *Proceedings of the 2nd Mile-High Video Conference*, pages 130–131, 2023.

[60] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE MultiMedia*, 18(4):62–67, 2011.

[61] P. Soumplis, P. Kokkinos, A. Kretsis, P. Nicopolitidis, G. Papadimitriou, and E. Varvarigos. Resource allocation challenges in the cloud and edge continuum. In *Advances in Computing, Informatics, Networking and Cybersecurity: A Book Honoring Professor Mohammad S. Obaidat's Significant Scientific Contributions*, pages 443–464. Springer, 2022.

[62] N. F. S. d. Sousa and C. E. Rothenberg. Clara: Closed loop-based zero-touch network management framework. In *2021 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 110–115, 2021.

[63] S. Taherizadeh, A. C. Jones, I. Taylor, Z. Zhao, and V. Stankovski. Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. *Journal of Systems and Software*, 136:19–38, 2018.

[64] B. Taraghi, H. Amirpour, and C. Timmerer. Multi-codec ultra high definition 8k mpeg-dash dataset. In *Proceedings of the 13th ACM Multimedia Systems Conference*, MMSys '22, pages 216–220, New York, NY, USA, 2022. Association for Computing Machinery.

[65] M. C. Thornburgh. Adobe's RTMFP Profile for Flash Communication. RFC 7425, Dec. 2014.

[66] M. Uitto and A. Heikkinen. Evaluation of live video streaming performance for low latency use cases in 5g. In *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 431–436, 2021.

[67] Y. Verginadis. A review of monitoring probes for cloud computing continuum. In L. Barolli, editor, *Advanced Information Networking and Applications*, pages 631–643, Cham, 2023. Springer International Publishing.

[68] X. Wang, T. Kwon, Y. Choi, H. Wang, and J. Liu. Cloud-assisted adaptive video streaming and social-aware video prefetching for mobile users. *IEEE Wireless Communications*, 20(3):72–79, 2013.

[69] H. Yousef, J. L. Feuvre, and A. Storelli. Abr prediction using supervised learning algorithms. In *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–6, 2020.

[70] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen. Toward efficient content delivery for automated driving services: An edge computing solution. *IEEE Network*, 32(1):80–86, 2018.

[71] E. Zavala, X. Franch, and J. Marco. Adaptive monitoring: A systematic mapping. *Information and Software Technology*, 105:161–189, 2019.