SARENA: SFC-Enabled Architecture for Adaptive Video Streaming Applications

Reza Farahani[1], Abdelhak Bentaleb[2], Christian Timmerer[1], Mohammad Shojafar[3], Radu Prodan[1], and Hermann Hellwagner[1]

[1] *Christian Doppler Laboratory ATHENA, Institute of Information Technology, Alpen-Adria-Universität Klagenfurt, Austria*

[2] *The Department of Computer Science and Software Engineering, Concordia University, Canada*

[3] *5GIC and 6GIC, Institute for Communication Systems (ICS), University of Surrey, United Kingdom*

***Abstract*—5G and 6G networks are expected to support various novel emerging adaptive video streaming services (*e.g.*, live, VoD, immersive media, and online gaming) with versatile *Quality of Experience* (QoE) requirements such as high bitrate, low latency, and sufficient reliability. It is widely agreed that these requirements can be satisfied by adopting emerging networking paradigms like *Software-Defined Networking* (SDN), *Network Function Virtualization* (NFV), and *edge computing*. Previous studies have leveraged these paradigms to present *network-assisted video streaming* frameworks, but mostly in isolation without devising chains of *Virtualized Network Functions* (VNFs) that consider the QoE requirements of *various* types of *Multimedia Services* (MS). To bridge the aforementioned gaps, we first introduce a set of multimedia VNFs at the edge of an SDN-enabled network, form diverse *Service Function Chains* (SFCs) based on the QoE requirements of different MS services. We then propose SARENA, an SFC-enabled ArchitectuRe for adaptive VidEo StreamiNg Applications. Next, we formulate the problem as a central scheduling optimization model executed at the SDN controller. We also present a *lightweight heuristic* solution consisting of *two* phases that run on the SDN controller and edge servers to alleviate the time complexity of the optimization model in large-scale scenarios. Finally, we design a large-scale cloud-based testbed including 250 *HTTP Adaptive Streaming* (HAS) players requesting two popular MS applications (*i.e.*, live and VoD), conduct various experiments, and compare its effectiveness with baseline systems. Experimental results illustrate that SARENA outperforms baseline schemes in terms of users' *QoE* by at least 39.6%, *latency* by 29.3%, and *network utilization* by 30% in both MS services.**

***Index Terms*—Network-Assisted Video Streaming; Dynamic Adaptive Streaming over HTTP (DASH); HTTP Adaptive Streaming (HAS); Service Function Chaining (SFC); Software-Defined Networking (SDN); Network Function Virtualization (NFV); Edge Computing.**

I. Introduction

Emerging adaptive video streaming services such as latency-sensitive (*e.g.*, *esport* and *online gaming*), bandwidth-sensitive (*e.g.*, *4K/8K video-on-demand* (VoD)) or both (*e.g.*, *immersive media* like *virtual reality* and *augmented reality*) have brought new *Quality of Experience* (QoE) requirements in terms of *latency*, *bandwidth*, and *reliability*. Satisfying these requirements is required for a pleasant user QoE which poses a new challenge for redesigning the current *best-effort* network architectures. Moreover, video traffic over mobile networks grew exponentially during recent years, where more than 75% of global mobile bandwidth will be used for video delivery by 2025 [1]. Such growth is due to the rapid development of mobile network technologies like 5G and 6G networks and their associated emerging paradigms (or enablers) such as SDN, NFV, and edge computing, demonstrating the critical role of next-generation mobile networks in the future Internet. That is why 5G and 6G hold promise to improve the QoE of various video streaming applications.

Motivation: While most video traffic of nowaday's services uses *HTTP Adaptive Streaming* (HAS)-based delivery systems [2] such as *Dynamic Adaptive Streaming over HTTP* (DASH) [3] and *HTTP Live Streaming* (HLS) [4], designing a holistic solution that supports various types of video streaming services with their strict QoE requirements over the next generation networks (*e.g.*, 5G and 6G) is still missing. Some prior works try to investigate this challenge, but mostly in isolation, without considering a holistic network architecture and their key enablers like SFC. Moreover, they only target one or two QoE requirements of a specific video streaming application.

These works are divided into two essential categories of textitclient-based, *server-assisted*, and *network-assisted* solutions. The client-driven solutions mainly optimize adaptive bitrate algorithms (ABR), the algorithm that selects the suitable bitrates depending on network conditions and/or playout buffer status [2]. These solutions are further divided into heuristic-based rules and learning-based approaches. The heuristic-based rules use one heuristic (or multiple heuristics) such as buffer-level [5], throughput [6], or both [7], in a sort of mathematical approximation to perform ABR decisions. Novel learning-based approaches [8], [9] leverage deep reinforcement learning (DRL) methods to improve users' QoE without any assumption about the environment or fixed rule-based heuristics. Different from client-driven solutions, the server-assisted solutions control clients' adaptive bitrate algorithms (ABR) decisions through client-server information sharing based on common media client/server data (CMCD/CMSD) standards [10] (*e.g.*, [11]–[14]), among others. The network-assisted solutions use in-network components, *e.g.*, an SDN controller or cloud/edge servers, to assure clients in their ABR decisions [15]–[21].

Different network services are sent through a set of network functions, nowadays in an NFV-based fashion, in a specific order to provide demanded functionalities and maintain service requirements. Therefore, designing appropriate VNFs for

particular services like MSs, mapping them to network nodes, and steering traffic among them is a necessary SFC procedure for successful service orchestration and delivery over next-generation networks. Although SFC has been investigated to solve various problems in the network domain, there is no SFC-enabled architecture for video streaming use cases. While there is a plethora of non-SFC based solutions, there exists no holistic solution that *(1)* leverages emerging network paradigms, *e.g.*, SDN, SFC, and edge computing, to *(2)* satisfy QoE requirements of *multiple types* of adaptive streaming services considering efficient *service functions* orchestration.

Contributions: This paper proposes an **S**FC-enabled **A**rchitectu**R**e for adaptive Vid**E**o Streami**N**g **A**pplications (SARENA). SARENA utilizes SFC for QoE improvement and awareness of various adaptive video streaming services. We design multimedia VNFs, build *SFC chains*, and utilize all possible *Contnet Delivery Network* (CDN) and edge server resources for serving different multimedia service requests in an SDN-enabled network. We formulate the problem as a central mixed-integer linear programming (MILP) optimization focusing on service scheduling. We then utilize the SDN controller capability to dynamically auto-scale edge servers based on service requirements. We augment each edge server with a lightweight scheduling function to alleviate the scalability and NP-complexity of the MILP model and make it deployable in practical environments. We analyze SARENA's performance through experiments performed in a large-scale cloud-based testbed including 250 clients and compare its results with selected baseline schemes. The experimental results confirms that SARENA outperforms baseline systems in terms of users' QoE, latency, and network utilization in a heterogeneous service environment.

Paper Outline: The remainder of this paper is organized as follows. Section II-A describes SARENA's architecture; the problem then is formulated as an MILP optimization model in Section II-B, and is solved by a lightweight heuristic method in Section II-C. The evaluation setup, methods, metrics, and results are described in Section III. Finally, Section IV concludes the paper and gives an outlook on future work.

II. SARENA System Design

In this section, we delve into the SARENA architecture, describe the system model and propose a heuristic solution.

A. SARENA Architecture

SARENA consists of three network layers as shown in Fig. 1.

Edge Layer (EL). This layer includes edge servers close to base stations (*e.g.*, gNodeB in 5G) and clients. Inspired by the *Consumer Technology Association* CTA-5004 standard [10], each edge server periodically communicates with the SDN controller and shares its available computational resources' status, *e.g.*, CPU and RAM (in so-called *comp stat* messages), the number of connected clients (*client stat* messages), the number/type of multimedia service requests (*MS stat* messages), and the cache occupancy (in *edge stat* messages) with the SDN controller. All edge devices are equipped with the

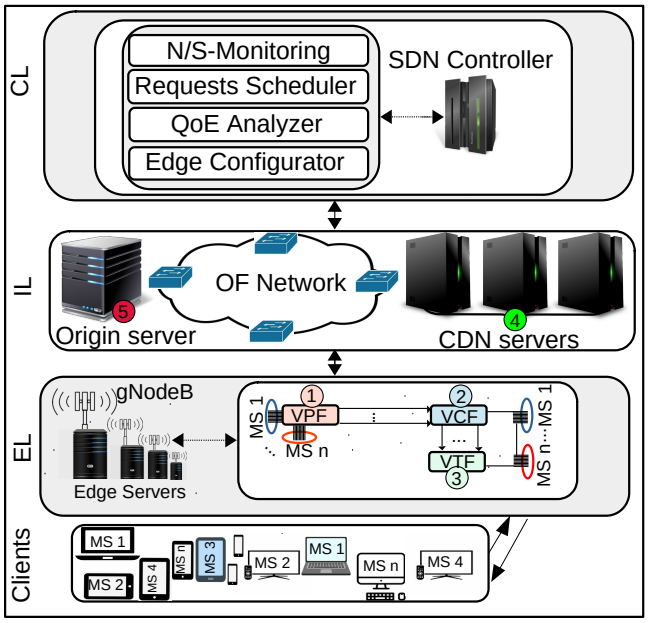

Figure 1: The proposed SARENA architecture

following VNFs: (1) *Virtual Proxy Function* (VPF), which enables an edge server to play the role of a gateway between its associated clients and the network, *i.e.*, receiving clients' demands for different MS services, employing determined decisions by the SDN controller, and responding to them; (2) *Virtual Cache Function* (VCF), utilized to store a limited number of popular segments for different MS services; (3) *Virtual Transcoding Function* (VTF), responsible for using computational resources and transcoding high quality segments to the required lower quality levels. These functions *placed* at the edge can collaborate and/or use other layer services to build SFCs and serve clients with desired service requirements.

Infrastructure Layer (IL). This layer consists of a group of CDN servers (either Over-the-Top (OTT) servers or purchased services from CDN providers) denoted by (4); each CDN contains various parts of video sequences. Like the EL layer, CDN servers periodically inform the SDN controller about their cache occupancy through *CDN stat* messages. Moreover, an origin server (shown by (5)) contains all video segments in multiple representations is placed in this layer. OpenFlow (OF) switches form an SDN-enabled backbone network.

Core Layer (CL). An SDN controller is placed in this layer, as shown in Fig. 1, augmented with a *Network/Service (N/S) Monitoring* module to monitor the EL and IL layers and collect the *stat* information about the network and MS services. Operating in a *time-slotted fashion*, the N/S monitoring module feeds a central decision-maker optimization module, called *Requests Scheduler*. This module optimally serves different MS requests with diverse requirements invoked by all edge servers. For this purpose, the requests scheduler module must respond to the following crucial questions: (1) Where is the optimal place (*i.e.*, edge, CDN servers, or origin server) for fetching the content quality level requested by each client, while efficiently employing layers' available resources and satisfying service requirements (*e.g.*, service deadlines)? (2)

How can we use the functions/services provided in the EL and IL layers to form MS function chains (SFCs)? (3) What is the optimal SFC for responding to the requested quality level with specific service requirements? To respond to these questions, the requests scheduler considers all possible functions/services to make the following SFCs, then runs an *optimization model* to determine the *optimal node* and *optimal SFC*, considering the service requirements. We will elaborate on the optimization model in more detail in Section II-B. Possible SFC chains are: *SFC#1*: Fetch the demanded quality level directly from the edge server (1 → 2). *SFC#2*: Transcode the demanded quality level from a higher quality at the edge (1 → 2 → 3). *SFC#3*: Fetch the requested quality level directly from the best CDN server (*i.e.*, in terms of the available bandwidth) (1 → 4). *SFC#4*: Fetch the requested quality from the origin server (1 → 5). *SFC#5*: Fetch a higher quality level from the best CDN server and transcode it into the demanded one at the edge (1 → 4 → 3). Notably, the SDN controller informs the VPF of edge servers about the optimal SFC and location; accordingly, edge servers serve their clients' requests from the determined server based on their service requirements.

Furthermore, the SDN controller employs a *QoE Analyzer* module to periodically analyze each edge server's served MS requests. Indeed, it uses information provided by the N/S monitoring and requests scheduler modules as inputs of a QoE model (see Section III), and if the calculated QoE value cannot meet the service threshold, it triggers the *Edge Configurator* module to adjust the edge configurations automatically. The *auto-scaling* feature utilized in the edge configurator, recently popular in both academia [22] and industry [23], assists the edge servers in providing assured service for streaming applications requiring varying amounts of computing resources in response to dynamic client behavior over time.

B. Problem Formulation

We introduce an MILP optimization model consisting of *four* constraint groups: *Chain Selection*, *Latency Calculation*, *Service/Policy*, and *Resource Utilization* constraints. Table I summarizes the notations used in this paper.

***(i)* Chain Selection constraint.** Let us define the binary decision-making variables $\mathcal{D}_{i,e,c}^{q,r}$ where $D_{i,e,c}^{q,r}=1$ shows edge $i=e$ or $i\in\mathcal{S}$ serves request r with quality level $q\in\mathcal{Q}^r$ via SFC chain $c\in\mathcal{C}$, otherwise $D_{i,e,c}^{q,r}=0$. Therefore, Eq. (1) chooses the best SFC chain for each request r issued by edge server $e\in\mathcal{E}$ by setting the $D_{i,e,c}^{q,r}$ to assure that each request is not parallelized, *i.e.*, split over multiple chains:

$$\sum_{i\in\{\mathcal{S}\cup e\}}\sum_{c\in\mathcal{C}}\sum_{q\in\mathcal{Q}^r}\mathcal{D}_{i,e,c}^{q,r}\times\alpha_i^{q,r}=1,\qquad\forall e\in\mathcal{E},r\in\mathcal{R}_e\quad(1)$$

***(ii)* Latency Calculation constraints.** Eq. (2) measures the transmission time $\mathcal{T}_{i,e}^{q,r}$, if cloud server i in one of the SFC#3, SFC#4, or SFC#5 is selected to transmit quality level $q\in\mathcal{Q}^r$ to edge server e:

$$\sum_{c\in\mathcal{C}}\mathcal{D}_{i,e,c}^{q,r}\times\delta^{q,r}\le\mathcal{T}_{i,e}^{q,r}\times\omega_{i,e},\quad\forall r\in\mathcal{R}_e,q\in\mathcal{Q}^r,i\neq e\quad(2)$$

Table I: Summary of main notations.

Notation	Description
	Input Parameters
$\mathcal{E}$	Set of n edge servers
$\mathcal{S}$	Set of k cloud servers, including CDNs and an origin (*i.e.*, $s=0$)
$\mathcal{C}$	Set of SFC chains, where $c=\{1,2,3,4,5\}$
$\mathcal{R}$	Set of x various MS requests received by the SDN controller from $\mathcal{E}$
$\mathcal{R}_e$	Set of various MS requests issued by edge server $e\in\mathcal{E}$
$\mathcal{Q}^r$	Set of possible quality levels for serving quality q^* issued by $r\in\mathcal{R}$, where $\mathcal{Q}^r=\{q^*,q^*+1,...,q_{max}^*\}$ and q_{max}^* is the maximum quality level for the requested segment
$\mathcal{A}_i^{q,r}$	Available quality levels in edge VCF (*i.e.*, $i\in\mathcal{E}$) or cloud servers (*i.e.*, $i\in\mathcal{S}$) to serve $r\in\mathcal{R}$; $\alpha_i^{q,r}=1$ means i hosts quality q to serve $r\in\mathcal{R}$, otherwise $\alpha_i^{q,r}=0$
$\delta^{q,r}$	Segment size in quality q requested by $r\in\mathcal{R}$
$\sigma^{q,r}$	Required resources (*i.e.*, CPU time in seconds) for transcoding quality $q\in\mathcal{Q}^r$ requested by $r\in\mathcal{R}$
$\eta^{q,r}$	Bitrate associated to quality level $q\in\mathcal{Q}^r$ requested by $r\in\mathcal{R}$
$\mu^{q,r}$	Required time for transcoding quality $q\in\mathcal{Q}^r$ into the quality q^* requested by $r\in\mathcal{R}$
π_e	Available computational resource (available CPU) of VTF in $e\in\mathcal{E}$
$\omega_{i,e}$	Available bandwidth on path between $i\in\mathcal{S}$ and $e\in\mathcal{E}$
θ^r	Given deadline for delivering request $r\in\mathcal{R}$ based on its MS type
	Variables
$\mathcal{D}_{i,e,c}^{q,r}$	Binary variable where $D_{i,e,c}^{q,r}=1$ indicates edge $i=e$ or $i\in\mathcal{S}$ serves request r with quality level $q\in\mathcal{Q}^r$ via SFC chain $c\in\mathcal{C}$, otherwise $D_{i,e,c}^{q,r}=0$
$\mathcal{P}_e^{q,r}$	Required transcoding time at VTF in $e\in\mathcal{E}$ for serving $r\in\mathcal{R}$ with quality level $q\in\mathcal{Q}^r$
$\mathcal{T}_{i,e}^{q,r}$	Required time for transmitting quality $q\in\mathcal{Q}^r$ in response to request $r\in\mathcal{R}$ issued by $e\in\mathcal{E}$ from $i\in\mathcal{S}$ and $c\in\{3,4,5\}$
χ	Total serving latency consisting of $\mathcal{P}_e^{q,r}$ and $\mathcal{T}_{i,e}^{q,r}$ for all MS services

Furthermore, Eq. (3) determines the required transcoding time $\mathcal{P}_e^{q,r}$ at edge e in case of serving the quality demanded by r from a higher quality q in SFC#2 or SFC#5:

$$\sum_{c\in\mathcal{C}\setminus\{1,3,4\}}\sum_{q\in\mathcal{Q}^r}\mathcal{D}_{i,e,c}^{q,r}\times\mu^{q,r}\le\mathcal{P}_e^{q,r},\quad(3)$$
$$\forall e\in\mathcal{E},i\in\{\mathcal{S}\cup e\},r\in\mathcal{R}_e$$

***(iii)* Service/Policy constraints.** The first constraint (Eq. (4)) of this group guarantees that the total request's serving latency for preparing the requested quality q of request r must respect the request service deadline (denoted by θ^r):

$$\sum_{i\in\{\mathcal{S}\cup e\}}\sum_{q\in\mathcal{Q}^r}\mathcal{T}_{i,e}^{q,r}+\mathcal{P}_e^{q,r}\le\theta^r,\qquad\forall e\in\mathcal{E},r\in\mathcal{R}_e\quad(4)$$

The total services' serving latency, namely χ, *i.e.*, fetching time plus transcoding time, for all requests can be expressed as the following constraints (Eq. (5)):

$$\sum_{r\in\mathcal{R}}\sum_{i\in\{\mathcal{S}\cup e\}}\sum_{e\in\mathcal{E}}\sum_{q\in\mathcal{Q}^r}\mathcal{T}_{i,e}^{q,r}+\mathcal{P}_e^{q,r}\le\chi\quad(5)$$

Moreover, Eq. (6) sets the IL policy by forcing the model to fetch the exact quality q^* from the origin server when the origin server (*i.e.*, $s=0$) is selected to serve r.

$$\sum_{q\in\mathcal{Q}^r}\mathcal{D}_{i=0,e,c=4}^{q,r}\times q=q^*,\qquad\forall e\in\mathcal{E},r\in\mathcal{R}_e\quad(6)$$

***(iv)* Resource Utilization constraints.** Eq. (7) ensures that the required bandwidth for transmitting quality q on the link

between cloud server i and edge e must respect the available bandwidth (denoted by $\omega_{i,e}$):

$$\sum_{r\in\mathcal{R}_e}\sum_{c\in\mathcal{C}}\sum_{q\in\mathcal{Q}^r}\mathcal{D}^{q,r}_{i,e,c}\times\eta^{q,r}\leq\omega_{i,e},\quad\forall e\in\mathcal{E},i\in\mathcal{S},i\neq e \quad (7)$$

Furthermore, Eq. (8) restricts the maximum required processing capacity for transcoding to the available computational resource on each edge server e (denoted by π_e):

$$\sum_{r\in\mathcal{R}_e}\sum_{i\in\{\mathcal{S}\cup e\}}\sum_{c\in\mathcal{C}\setminus\{1,3,4\}}\sum_{q\in\mathcal{Q}^r}D^{q,r}_{i,e,c}\times\sigma^{q,r}\leq\pi_e\quad\forall e\in\mathcal{E} \quad (8)$$

Central Scheduling Optimization Model. An ABR algorithm embedded in a HAS player assesses the network's bandwidth by measuring the time between sending the request to download a segment and receiving the segment's last packet. Thus, minimizing the serving latency in the optimization model directly impacts the HAS clients' performance. To this end, the model for minimizing total requests' serving latency, denoted by χ, can be expressed as follows:

$$\begin{aligned} &\textit{Minimize} \quad \chi \qquad\qquad\qquad\qquad (9)\\ &\quad s.t. \quad \text{constraints} \quad \text{Eq.}(1)-\text{Eq.}(8)\\ &\quad vars. \quad \mathcal{T}^{q,r}_{i,e},\mathcal{P}^{q,r}_{e},\chi\geq 0, D^{q,r}_{i,e,c}\in\{0,1\}\end{aligned}$$

By running the MILP model (Eq. (9)), an optimal chain will be selected for each request to minimize the total serving time. However, since the MILP model is NP-hard [24], it suffers from high time complexity and is impractical for large-scale scenarios. Thus, we introduce a lightweight heuristic solution in the next section to cope with the aforementioned problems.

C. Lightweight Heuristic Algorithms

This section proposes two simple lightweight algorithms distributed on the SDN controller and edge servers. Our solution aims at satisfying the constraints (1)–(8) by reducing the responsibilities of the SDN controller in terms of MS requests' scheduling and introducing a new VNF called *Virtual Scheduler Function* (VSF) deployed at the edge. The VSF function utilizes constraints (1)–(8) in the form of a lightweight heuristic algorithm to produce a nearly-optimal solution for its local edge server instead of running a central complex optimization model for all edge servers. Although we split the central MILP model (9) into edge VSFs, the SDN controller, as a coordinator node, still collects *stats* information from the layers (*i.e.*, by N/S monitoring), advertises this information to edge servers, and analyzes served requests' QoE (*i.e.*, by the QoE analyzer) and reconfigures edge servers if their associated MS services can not meet the defined service thresholds (*i.e.*, by the edge configurator).

Like the centralized model, the VSF-based solution works in a time-slotted manner. A time slot consists of *two* intervals: *(i) Stats/Requests Collector* (SRC) interval and *(ii) Requests Scheduler* (RES) interval. In the SRC interval, an edge server simultaneously receives *stats* information and MS requests provided by the SDN controller and the VPF function, respectively. Considering the provided data, the VSF function in the RES interval runs Alg. 1 to serve MS requests by choosing suitable SFCs, *i.e.*, *SFC#1*: (2), *SFC#2*: (2)→(3), *SFC#3*: (4), *SFC#4*: (5), or *SFC#5*: (4)→(3), and minimizes the total serving latency w.r.t. the service requirements and objective function of Eq. (9). We present the proposed algorithms separately in Alg. 1 and Alg. 2, which are deployed on the edge servers and the SDN controller, respectively.

Algorithm 1: Edge server heuristic algorithm

```
   Input: requests, stat_info, BW_info, on_the_fly
1  features ← ExtractFeatures(requests)
2  MS_queues ← MakeQueues(requests, features)
3  */Each MS_m Thread: VSF()
4  sort(MS_queue_m, features_m)
5  for each req in MS_queues_m do
6      if req ∈ on_the_fly then
7          HoldReq(req)
8      else
9          on_the_fly.add(req)
10         SFC_set ←SFCDetector()
11         SFC_cost ← CostFunction(SFC_set)
12         opt_SFC ←OptimalSFC(SFC_set,SFC_cost)
13         ServeRequest(req,opt_SFC)
14         UpdateVariables()
```

Edge-based Scheduling Heuristic Algorithm. As shown in Alg. 1, each edge server receives data provided by the SDN controller, *i.e.*, $stat\_info$ and information on bandwidth to CDN/origin servers (*i.e.*, BW_info) in the SRC interval. The edge server calls the *ExtractFeatures* function to extract some essential features of input requests, *e.g.*, MS types, requested videos/channels, requested qualities, request receiving time, and service deadlines (line 1). After that, based on the extracted features stored in the $features$ list, the *MakeQueues* function is utilized to form different queues of requests (line 2). For instance, two popular MS applications, *i.e.*, VoD and live requests, are placed in separate queues based on MS type, requested live channel/VoD video IDs, and bitrate levels. Considering the system's current state, *i.e.*, available information on resources (*i.e.*, $stat\_info$) and queues of MS requests, the edge server in the RES interval must run multiple threads of a VSF function (one thread per MS type) to answer the questions mentioned in Section II-A.

At the start of each VSF thread, associated MS queues are sorted based on the extracted information, like service deadline using the *sort* function (line 4). Next, each MS request (req) is compared to on_the_fly requests (*i.e.*, requests currently being served). If req is in the on_the_fly list, then it calls *HoldReq* to hold the request and prevent network resource wastage and congestion (lines 6–7). Otherwise, the on_the_fly list is updated by the req to be processed (line 9). In the next step, the *SFCDetector* function determines all feasible SFCs (*i.e.*, *SFC#1–SFC#5*) and stores them in the SFC_set list (line 10). Considering the objective function (9) and defined constraints (1)–(8), the serving latencies of the SFCs contained in SFC_set are calculated by calling the *CostFunction* function and then the results are saved in SFC_cost (line 11). Next, the *OptimalSFC* function calculates the minimum value (*i.e.*, serving latency)

Algorithm 2: SDN controller heuristic algorithm

```
1 while True do
2     stat_info, BW_info ← N/S-Monitoring()
3     for each e in E do
4         MS_QoE_e ← QoEAnalyzer(e)
5         if MS_QoE_e < MS_QoE_th then
6             EdgeConfigurator(e)
7     Update(E)
8     Wait (τ)
```

in the SFC_cost structure, retrieves its associated SFC from SFC_set, and saves it as the optimal SFC (line 12). Finally, the *ServeRequest* and *UpdateVariables* functions are utilized to serve the clients' request with the optimal SFC and to upgrade *stat* information and the on_the_fly list, respectively (lines 13–14). Note that since more than one queue can proceed and might violate all/several resource constraints (*e.g.*, the bandwidth and/or computational limits), they are evaluated in a priority order where the queue with more requests and earlier service deadlines comes first. This process will be repeated in each RES interval until the MS session ends and all queues are served. Assume β_1, β_2, and β_3 indicate the number of MS services, number of channels/videos, and number of bitrates per channel/video. In the worst case, the time complexity of the multi-threaded Alg. 1 employed by each edge server would be $O(\beta_1 \times \beta_2 \times \beta_3)$ in each time slot.

SDN-based Management Heuristic Algorithm. In Alg. 2, the SDN controller uses the *N/S-Monitoring* function to collect the *stats* and bandwidth information from the network and servers, plus average QoE parameters for the served requests in the previous time slot (line 2). Based on the collected data, the *QoE_Analyzer* function is called to calculate the MS QoE scores for each edge server and store values in each edge MS_QoE_e list (line 4) (see Section III for the used MS QoE model). Next, if the calculated values violate the service QoE thresholds (denoted by MS_QoE_th), which is adjusted by the network or video operator based on their business plans and services, the *EdgeConfigurator* function is called to reconfigure the edge server (lines 5–6) and then updates all edge servers (line 7). We note that any policy on QoE analyzing and auto-scaling can be applied. This explained procedure will repeat periodically after τ seconds, where τ is the duration of the SRC interval within the while loop (line 8). The overall time complexity of Alg. 2 can be given as $O(n)$, where n is the number of edge servers.

III. Performance Evaluation

Evaluation Setup: To evaluate the performance of SARENA in a realistic large-scale environment, we instantiate our testbed on the CloudLab [25] and use InternetMCI [26] as a real backbone network topology. Our testbed includes 280 components, *i.e.*, *(i)* 250 *AStream* [27] DASH players running the *BOLA* [5] ABR algorithm in headless modes (five groups of 50 peers); *(ii)* five Apache HTTP servers (*i.e.*, four CDN servers with a total cache size of 40% of the MS video datasets and an origin server, holding all MS video sequences); *(iii)* 19 OpenFlow (OF) backbone switches and 45 backbone layer-2 links; *(iv)* five edge servers, each of which is responsible for one group of clients and includes a partial cache size of only 5% of the MS video sequences; and a FloodLight SDN controller. Note that each element is run on Ubuntu 18.04 LTS inside Xen virtual machines within separated Linux namespaces. To emulate the auto-scaling feature, a basic configuration (*i.e.*, 4 CPU cores and 6 GB RAM) of virtual machines supporting a maximum of 10 CPU cores and 16 GB RAM is assigned to all edge servers at the beginning of all experiments. Next, the edge servers' configurations scale up edge configurations by adding 2 CPU cores and 2 GB RAM whenever the SDN edge configurator module is triggered. Although SARENA is independent of the caching policy, for the sake of simplicity, Least Recently Used (LRU) cache replacement policy is considered in all CDNs and VCFs. Note that the most popular MS VoD videos are pre-cached on the VCFs to avoid a slow startup of the system. Python 3.7 is used to implement all modules of the SDN controller and the VNFs. Moreover, the Python PuLP library with the CPLEX solver and Dockerimage *jrottenberg/ffmpeg* [28] are employed to implement the MILP model and VTSs, respectively.

We evaluate the performance of SARENA through two prevalent MS services, *i.e.*, VoD and live streaming applications. (MS_QoE_th, θ_r) values associated with live and VoD services are set to (4, 2 s) and (3.5, 4 s), respectively, to make different service requirements. We consider five live channels, where each of which plays a unique video [29] with 300 s duration, comprising two-second segments of the following bitrate ladder {(0.089,320p), (0.262,480p), (0.791,720p), (2.4,1080p), (4.2,1080p)} [Mbps, resolution]. Moreover, 20 video sequences [29] with 300 seconds duration and four-second segments in seven representations {(0.128,320p), (0.320,480p), (0.780,720p), (1.4,720p), (2.4,1080p), (3.3,1080), (3.9,1080)} are employed for the VoD services. For simplicity, we assume each client requests only one MS type during its streaming session. The live channel and VoD video access probability are generated following a Zipf distribution with the skew parameter $\alpha = 0.75$. The probability of an incoming request for the i^{th} most popular live channel or VoD video are given as $prob(i) = \frac{1/i^{\alpha}}{\sum_{j=1}^{K} 1/j^{\alpha}}$, where $K = 5$ and $K = 20$ for the live and VoD services, respectively. In the literature, the bandwidth value from the CDN servers to an edge server is assumed to be higher than from the origin server to an edge server. [30]. Therefore, the Linux *Wondershaper* tool is employed to set 50 and 100 Mbps as a bottleneck bandwidth in different paths from the CDN and origin servers to edge servers, respectively. In addition, we use a real *4G network trace* [31] (average bandwidth 3780 kbps and standard deviation 3190 kbps) collected on bus rides for links between clients to edge servers in all experiments. The SDN QoE analyzer module uses the *ITU-T Rec. P.1203* model in mode 0 [32], as a standard comprehensive QoE model. The computational and bandwidth costs are set to 0.029\$ per CPU per hour and 0.12\$ per GB, respectively [33].

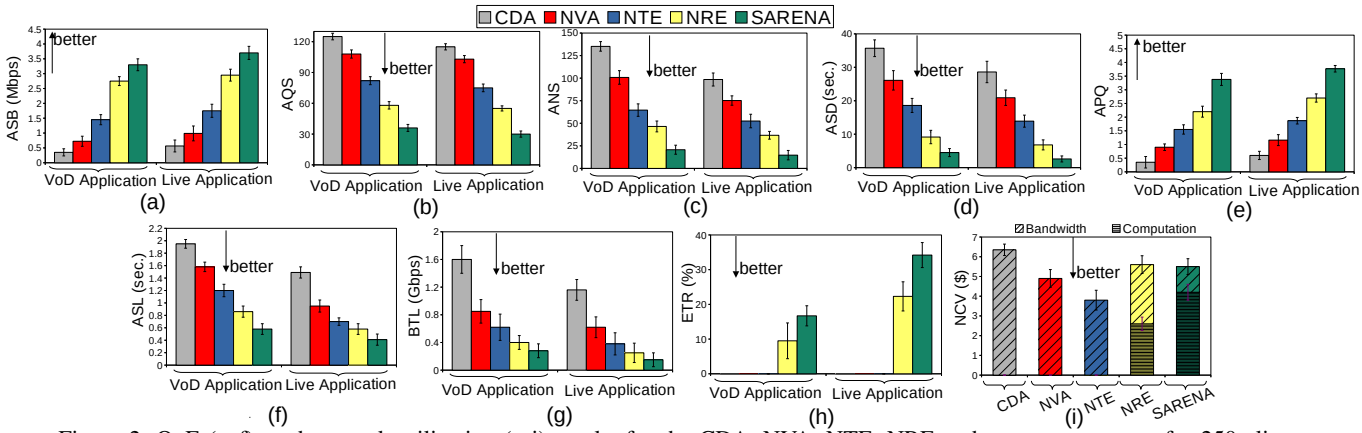

Figure 2: QoE (a–f), and network utilization (g-i) results for the CDA, NVA, NTE, NRE and SARENA systems for 250 clients

Evaluated Methods: Since there are no SFC-enabled frameworks supporting various HAS applications in the literature, we compare the results obtained by SARENA with the following baseline methods: *(i) CDN-assisted* (CDA): this is a regular CDN-based streaming method. *(ii) Non VNF-assisted* (NVA): edge devices without VCF and VTF functions are used to find the best CDN server with maximum available bandwidth. *(iii) Non VTF-enabled* (NTE): edge servers are equipped with the VCFs, but do not support VTFs. *(iv) Non Reconfiguration-enabled (NRE)*: this approach employs a simple version of SARENA without the SDN edge configurator module. For fair comparisons, the results of all systems are derived with respect to the same settings and the same topology in our testbed. Moreover, each experiment is run 20 times, and the average and standard deviation values are reported in the experimental results.

Evaluation Metrics: *(i) Average Segment Bitrate* (ASB) of all the downloaded segments; *(ii) Average Number of Quality Switches* (AQS), the average number of segments whose bitrate level changed compared to the previous one; *(iii) Average Stall Duration* (ASD), the average of total video freeze time of all clients; *(iv) Average Number of Stalls* (ANS), the average number of rebuffering events; *(v) Average Perceived Overall QoE* (APQ) calculated by the ITU-T Rec. P.1203 model in mode 0; *(vi) Average Serving Latency* (ASL), defined as the overall time for serving all clients, including fetching latency plus transcoding latency; *(vii) Backhaul Traffic Load* (BTL), the volume of segments downloaded from the origin server; *(viii) Edge Transcoding Ratio* (ETR), the fraction of segments reconstructed by the VTFs; *(ix) Cache Hit Ratio* (CHR), defined as the fraction of segments fetched from the CDN or edge servers; *(x) Network Cost Value* (NCV), consisting of computational and bandwidth costs. Note that the arrows in each metric plot (Fig. 2 (a)–(i)) indicate which direction (up or down) represents a better result.

Evaluation Results: In the first scenario, we conduct experiments to assess SARENA's performance in terms of the QoE metrics and compare the results with described baseline systems. As plotted in Fig. 2(a–d), SARENA downloads VoD and live streaming segments with higher ASBs, improves AQSs and ANSs, and shortens ASDs in both VoD and live streaming applications. Although enhancing the aforementioned common QoE parameters can generally improve the users' satisfaction, the standard comprehensive QoE model is utlized by the APQ metric to evaluate overall users' QoE. Note that stalling events (measured by ASD and ANS) seriously impact APQ compared to other parameters. Utilizing the VCFs and VTFs in SFCs enables the SARENA system to download MS segments with improved QoE metrics, especially ASD, and leads to achieving better APQ by at least 53% and 39.6% for VoD and live streaming, respectively, compared to the baseline approaches (Fig. 2(e)). Moreover, as illustrated in Fig. 2(f), the average serving latency values (ASL) obtained by SARENA in VoD and live applications are reduced by at least 32.5% and 29.3% compared to other methods. This is because SARENA *(i)* uses latency- and QoE-sensitive policies to meet θ_r and MS_QoE_th values, *(ii)* utilizes all layers of possible resources employed in SFCs, and *(iii)* operates an auto-scaling policy provided by the SDN controller, for serving MS requests.

The second scenario studies the effectiveness of SARENA in terms of network utilization (*i.e.*, BTL and ETR) and network cost metrics (*i.e.*, NCV). Since SARENA *(i)* fetches requested or higher MS quality levels from the VCF or CDN servers and *(ii)* transcodes requested MS quality levels by VTF, it downloads fewer segments from the origin server, consequently outperforms the CDA, NVA, and NTE systems in terms of BTL (Fig. 2(g)). Moreover, the auto-scaling capability provided by the SDN controller assists SARENA in yielding a better BTL (40% and 30% for VoD and live applications, respectively) compared to the NRE system. Moreover, the ETR metric indicates that the auto-scaling feature allows SARENA to operate more computational resources of the edge servers to transcode MS applications, especially for serving live streaming requests, as latency-sensitive demands (Fig. 2(h)). Note that the ETR metrics for not transcoding-enabled systems, *i.e.*, CDA, NVA, and NTE, are zero. Although the SARENA uses more edge computational resources, our final experiment (Fig 2(i)) demonstrates that its NCV metric is still lower than for CDA and (almost) identical to the NRE system. This is

because SARENA improves the costly BTL metric.

IV. Conclusion and Future work

We presented SARENA, an SFC-enabled architecture for adaptive video streaming applications. The main objective of SARENA is to use the cooperation of emerging 5G/6G paradigms such as SDN, SFC, and edge computing to serve efficiently various types of adaptive video streaming services such as latency- and/or bandwidth-sensitive ones, satisfying their QoE requirements in terms of latency, bandwidth, and reliability. Our experimental results over a large-scale testbed show that SARENA outperforms baseline approaches in terms of QoE by at least 39.6%, latency by 29.3% , and network utilization by 30%. In the future, we plan to address real-time decision-making for the SFC selection of multimedia services by adopting reinforcement (RL) learning techniques.

Acknowledgment

The financial support of the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association is gratefully acknowledged. Christian Doppler Laboratory ATHENA: https://athena.itec.aau.at/.

References

[1] Ericsson, "Ericsson Mobility Report." [Online]. Available: https://www.ericsson.com/4a4be7/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-q2-2022.pdf

[2] 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*, 2018.

[3] I. Sodagar, "The MPEG-DASH standard for multimedia streaming over the Internet," *IEEE Multimedia*, 2011.

[4] R. Pantos and W. May, "HTTP Live Streaming," RFC 8216, Aug. 2017. [Online]. Available: https://rfc-editor.org/rfc/rfc8216.txt

[5] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions on Networking*, 2020.

[6] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. C. Begen, and D. Oran, "Probe and adapt: Rate adaptation for HTTP video streaming at scale," *IEEE Journal on Selected Areas in Communications*, 2014.

[7] X. Yin, A. Jindal, V. Sekar, and B. Sinopoli, "A control-theoretic approach for dynamic adaptive video streaming over HTTP," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015.

[8] A. Bentaleb, M. N. Akcay, M. Lim, A. C. Begen, and R. Zimmermann, "Catching the Moment With LoL$^+$ in Twitch-Like Low-Latency Live Streaming Platforms," *IEEE Transactions on Multimedia*, 2021.

[9] I. M. Ozcelik and C. Ersoy, "ALVS: Adaptive Live Video Streaming using deep reinforcement learning," *Journal of Network and Computer Applications*, 2022.

[10] CTA-5004, "Web Application Video Ecosystem–Common Media Client Data." 2020. [Online]. Available: https://cdn.cta.tech/cta/media/media/resources/standards/pdfs/cta-5004-final.pdf

[11] A. Bentaleb, M. Lim, M. N. Akcay, A. C. Begen, and R. Zimmermann, "Common media client data (cmcd) initial findings," in *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2021.

[12] R. Farahani, H. Amirpour, F. Tashtarian, A. Bentaleb, C. Timmerer, H. Hellwagner, and R. Zimmermann, "RICHTER: hybrid P2P-CDN architecture for low latency live video streaming," in *Proceedings of the 1st Mile-High Video Conference*, 2022.

[13] R. Farahani, A. Bentaleb, E. Çetinkaya, C. Timmerer, R. Zimmermann, and H. Hellwagner, "Hybrid P2P-CDN Architecture for Live Video Streaming: An Online Learning Approach," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022.

[14] R. Farahani, A. Bentaleb, M. Shojafar, C. Timmerer, and H. Hellwagner, "CP-Steering: CDN- and Protocol-Aware Content Steering Solution for HTTP Adaptive Video Streaming," in *Proceedings of the 2nd Mile-High Video Conference*, 2023.

[15] A. Bentaleb, A. C. Begen, and R. Zimmermann, "SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking," in *Proceedings of the 24th ACM International conference on Multimedia*, 2016.

[16] A. Zhang, Q. Li, Y. Chen, X. Ma, L. Zou, Y. Jiang, Z. Xu, and G.-M. Muntean, "Video super-resolution and caching—An edge-assisted adaptive video streaming solution," *IEEE Transactions on Broadcasting*, 2021.

[17] R. Farahani, F. Tashtarian, C. Timmerer, M. Ghanbari, and H. Hellwagner, "LEADER: A Collaborative Edge-and SDN-Assisted Framework for HTTP Adaptive Video Streaming," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022.

[18] R. Farahani, "CDN and SDN Support and Player Interaction for HTTP Adaptive Video Streaming," in *Proceedings of the 12th ACM Multimedia Systems Conference*, 2021.

[19] R. Farahani, F. Tashtarian, H. Amirpour, C. Timmerer, M. Ghanbari, and H. Hellwagner, "CSDN: CDN-Aware QoE Optimization in SDN-Assisted HTTP Adaptive Video Streaming," in *Proceedings of the 46th IEEE Conference on Local Computer Networks (LCN)*, 2021.

[20] R. Farahani, F. Tashtarian, A. Erfanian, C. Timmerer, M. Ghanbari, and H. Hellwagner, "ES-HAS: An Edge- and SDN-Assisted Framework for HTTP Adaptive Video Streaming," in *Proceedings of the 31st ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2021.

[21] 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*, 2022.

[22] M. S. Aslanpour, A. N. Toosi, R. Gaire, and C. M. Aamir, "Auto-scaling of web applications in clouds: A tail latency evaluation," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2020.

[23] https://aws.amazon.com/autoscaling/, accessed: 20123-02-13.

[24] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[25] R. Ricci, E. Eide, and C. Team, "Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications," *login:: The Magazine of USENIX & SAGE*, 2014.

[26] http://www.topology-zoo.org/dataset.html, accessed: 20123-02-13.

[27] https://github.com/pari685/AStream, accessed: 20123-02-13.

[28] https://hub.docker.com/r/jrottenberg/ffmpeg, accessed: 20123-02-13.

[29] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over HTTP dataset," in *Proceedings of the 3rd ACM Multimedia Systems Conference*, 2012.

[30] A. O. Al-Abbasi, V. Aggarwal, and M.-R. Ra, "Multi-tier caching analysis in CDN-based over-the-top video streaming systems," *IEEE/ACM Transactions on Networking*, 2019.

[31] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: a 4G LTE dataset with channel and context metrics," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018.

[32] "ITU-T. Rec. P.1203. Parametric bitstream-based quality assessment of progressive download and adaptive audiovisual streaming services over reliable transport - video quality estimation module." [Online]. Available: http://handle.itu.int/11.1002/ps/P1203-01

[33] https://calculator.aws/, accessed: 20123-02-13.