Fog Computing for Low Latency, Interactive Video Streaming

A Thesis

Presented to the

Graduate Faculty of the

University of Louisiana at Lafayette

In Partial Fulfillment of the

Requirements for the Degree

Master of Science

Vaughan Veillon

April 2019

Fog Computing for Low Latency, Interactive Video Streaming

Vaughan Veillon

APPROVED:

Mohsen Amini Salehi, Chair Assistant Professor of Computer Science The Center for Advanced Computer Studies	Nian-Feng Tzeng Professor of Computer Science The Center for Advanced Computer Studies
Miao Jin Associate Professor of Computer Science The Center for Advanced Computer Studies	Sheng Chen Assistant Professor of Computer Science The Center for Advanced Computer Studies
Mary Farmer-Kaiser Dean of the Graduate School	

To my parents for all their love and support.

Acknowledgments

My sincerest gratitude goes out to my supervisor, Professor Mohsen Amini Salehi, for persistence and encouragement to push me further as an academic. I would like to thank my thesis committee, Nian-Feng Tzeng, Miao Jin, and Sheng Chen. Finally, my thanks goes out to the Center for Advanced Computer Studies and the Graduate School at the University of Louisiana at Lafayette for their support.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Video streaming occupies more than 75% of the whole Internet bandwidth and it is predicted that the growth will persist [3]. The resources required to provide video streams are also increasing. High quality video (such as 4K Ultra High-Definition/UHD) and advanced video streaming (such as 360 degree videos, motion tracking, face recognition analysis, dynamic censorship) are becoming commonplace.

High quality and advanced video streaming increases data rate consumption and demands low streaming latency [4]. With the demand for streaming video content steadily increasing, the ability to effectively deliver video content to viewers that are spread on a global scale is a major concern for video streaming providers [5]. Many video stream providers (e.g., Netflix [6], YouTube [7], Amazon Prime Video [8]) are reliant on clouds for their offered services; hence, clouds have gained a pivotal role in the streaming process over the past few years. For many video stream providers, cloud services are the major source of ongoing costs [9]. In addition, clouds are inherently built in a centralized manner via gigantic datacenters. However, such a centralized nature is detrimental to video streaming latency.

Accordingly, in this thesis, the goal is to investigate how distributed, fog, and cloud computing can be efficiently utilized to offer a high quality, low latency, and albeit, cost-efficient video streaming.

1.1 Motivations

In this thesis, we define *interactive video streaming* as the ability to perform any form of processing for viewers, enabled by video stream providers, on the videos being

streamed. Examples of interactive video streaming include:

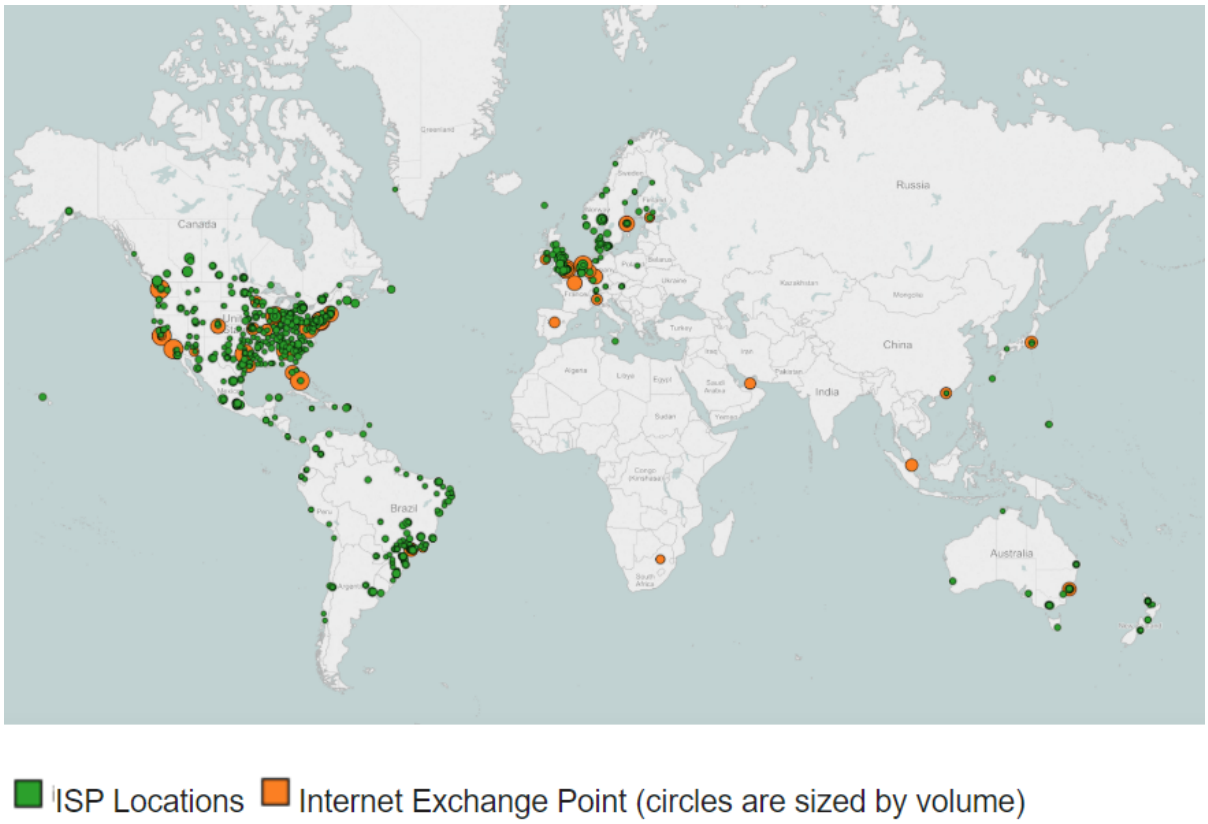
- Dynamic Video Stream Summarization - Consider an e-learner who does not have time to stream the whole educational video and would like to stream only a summary of the video with a given duration.

- Dynamic Content Rating of Video Streams - Family viewers of an online movie would like to remove inappropriate content from the stream. Adult viewers of the same stream may not have such a constraint.

- Dynamic Video Transcoding - The display device of a viewer cannot play a video stream because the device does not support the streaming format. The viewer would like to stream a converted (*i.e.,* transcoded) version of the stream supported by their device. In this case, viewers require dynamic transcoding of video streams based on the characteristics of their display devices to be able to watch the best quality video on their devices.

If a streaming service provider would like to provide these services in a non-dynamic manner, then multiple versions of each video would have to be pre-processed and persisted. With the increasing number of video processing options, the number of possible versions of a video increases as well. In addition, the heterogeneity of viewers' devices (*e.g.,* smart phone, smart TV, laptop) increases the overall number of possible versions for each video. This is the case because a video stream must be configured to the specific device that requests the stream. It quickly

Figure 1.1. Map of the global distribution of the Open Connect Appliances that constitute Netflix's content delivery network [1].



becomes infeasible to persist and cache all versions of videos, especially in the context of a distributed system attempting to service globally-spread viewers.

To address increasing data rate concerns, streaming providers require large-scale computing and storage resources. Therefore, many video providers (*e.g.,* Netflix) have migrated to cloud services to host and deliver their video contents. Using cloud services alleviates the burden of maintaining and upgrading physical resources from the video streaming provider. For instance, since 2015, Netflix stopped using their own datacenters and moved their entire streaming service to Amazon cloud (AWS) and Open Connect Appliances (OCA) [6]. Also, YouTube utilizes Google cloud services to

achieve their streaming demands [10]. However, the latency of accessing cloud services can be significant, specifically, for viewers that are distant to the cloud servers [5].

In order to overcome this inherent latency issue, stream providers commonly utilize a distributed system known as a *Content Delivery Network* (CDN) [5]. A CDN caches part of the video repository into its edge locations that are physically close to viewers, resulting in a lower latency compared to accessing from a more centrally located cloud server. Figure 1.1 [1] shows the locations of the OCA devices that constitute Netflix's CDN infrastructure. A motivation of this work is to take existing CDN practices and further enhance them, specifically with the capability of interactive streaming.

1.2 Research Problem and Objectives

In this research the question that must be addressed is: *How can we have a generic video streaming engine that can support any interaction on the video streams for viewers?* Since there are a wide variety of possible interactions with video streams, it is not possible to pre-process and store video streams in all possible versions; instead, they must be processed upon viewer's request, in a real-time manner. It is not possible to process the video streams on viewers' thin-clients (*e.g.,* smartphones), due to energy and computational limitations [11]. The emergence of cloud services has provided the computational power required for such processing. However, the remaining question is: how to provision cloud resources (*e.g.,* containers and storage) for viewers' interactions and schedule streaming tasks on the allocated resources so that the *Quality of Experience* (QoE) for viewers is guaranteed and the minimum cost is incurred to the

streaming provider?

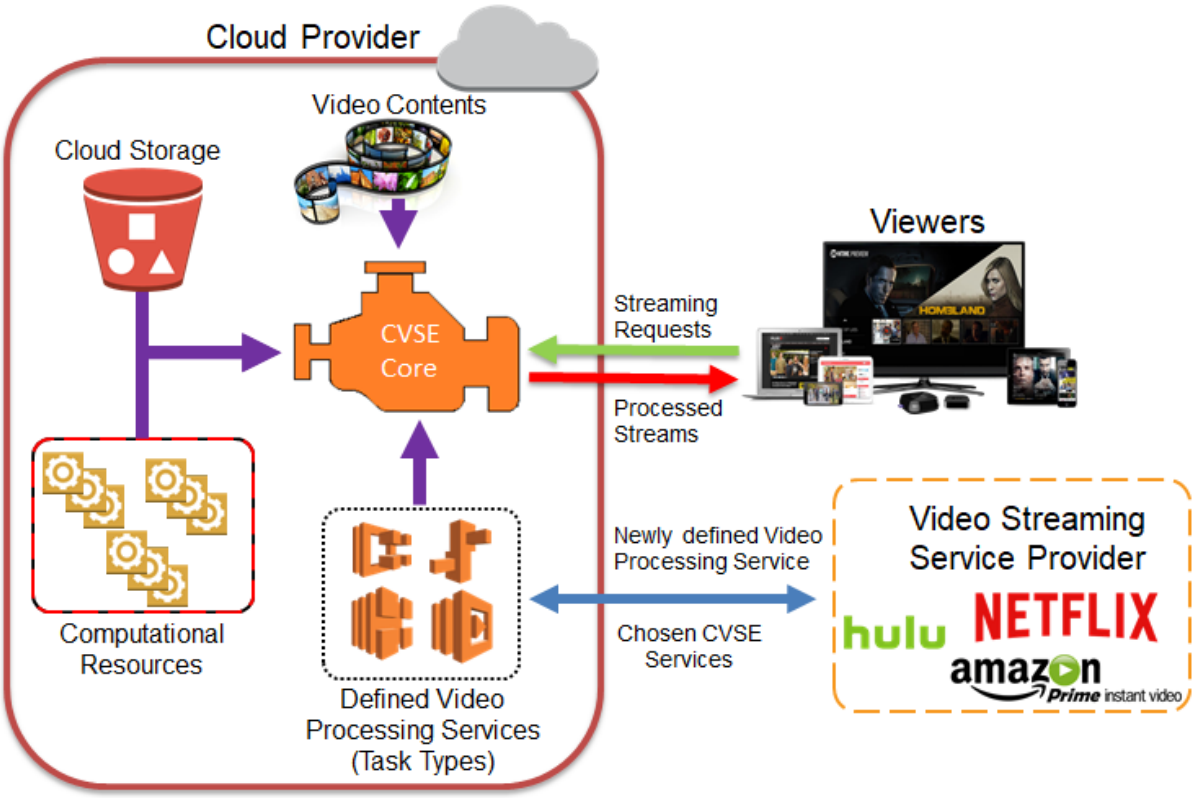
The problem is that the large and fast-growing repository size of streaming providers has made it infeasible to cache a large portion of the overall content on their CDNs. In addition, caching on CDNs is less effective because of the fact that streaming providers have to maintain multiple versions of the same video to be able to support heterogeneous display devices and network conditions [12]. As such, instead of pre-processing video streams into multiple versions, mechanisms for on-demand processing (*e.g.,* on-demand transcoding [12]) of video streams are becoming prevalent [13, 14]. However, the challenge is that on-demand video processing cannot be performed on CDNs since they are predominantly used for caching purposes [15].

These limitations lead to frequent streaming directly from more centrally located cloud servers, which increases streaming latency, hence decreasing viewers' QoE, particularly in distant areas [16]. In this research, we aim to overcome these limitations in existing systems and develop a system that can provide low latency interactive video streams independent of a viewer's geographical location.

1.3 Methodology Overview

Our goal is to enable video stream providers to offer a wide range of interactive services to viewers (*e.g.,* dynamic video summarization or dynamic transcoding) through on-demand processing of video streams on the cloud. To achieve this goal we have developed the platform *Cloud-based Video Streaming Engine* (CVSE). This platform enables interactive streaming services through on-demand video stream processing using potentially heterogeneous cloud services, in a cost-efficient manner,

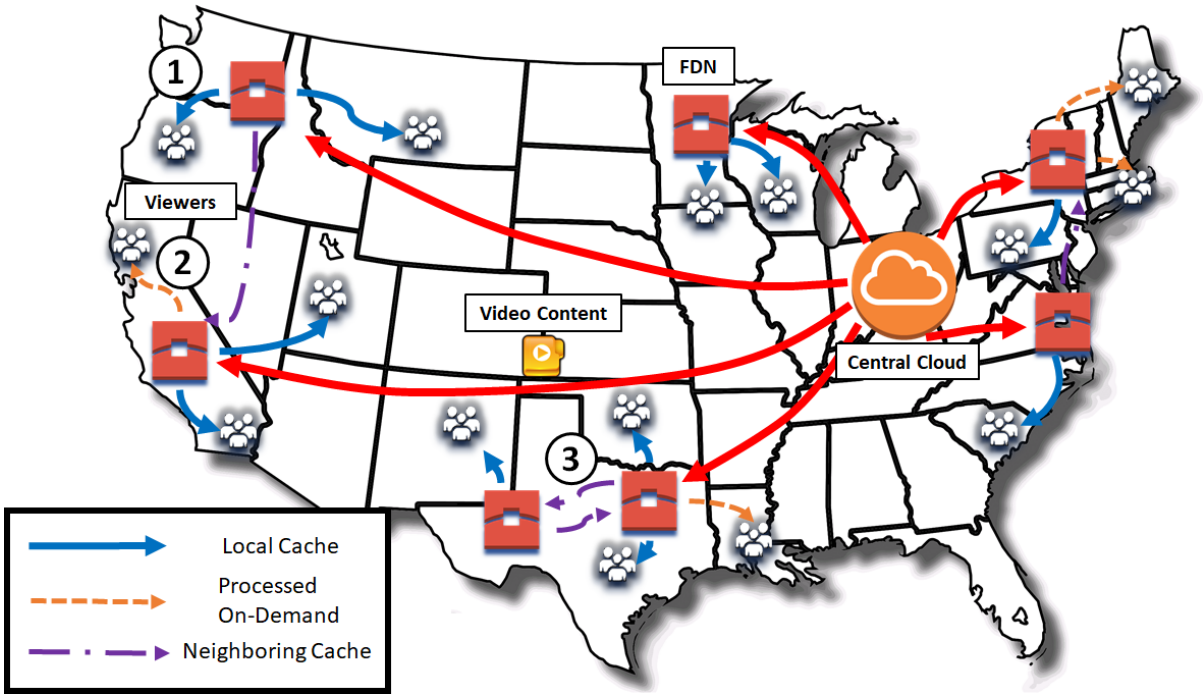
Figure 1.2. Viewers' and video stream service providers' interaction with Cloud-based Video Streaming Engine (CVSE)



while observing viewers' QoE guarantees. CVSE will be extensible, meaning that the stream service provider will be able to introduce new interactive services on video streams and the core architecture can accommodate the services while respecting the QoE and cost constraints of the stream service provider. The ways in which viewers and streaming providers can interact with CVSE are shown in Figure 1.2.

To ensure that the interactive video streams are provided with sufficiently low latency, we have also developed a distributed video delivery platform, *Federated Fog Delivery Networks* (F-FDN). F-FDN leverages the computing ability of fog systems to carry out on-demand processing of videos at the edge level. F-FDN is composed of several Fog Delivery Networks (FDN) that collaboratively stream videos to viewers

Figure 1.3. High level view of the F-FDN architecture. Note the viewers that are receiving video content from multiple FDN. 1) shows video content coming from FDN local cache 2) shows video content being processed on-demand 3) shows video content coming from a neighboring FDN's cache



with the aim of minimizing video streaming latency. Our goal in this study is to utilize our platform CVSE to provide interactive video streams within our video stream delivery platform F-FDN taking advantage of fog computing practices and dynamic decision making based on probabilistic methodologies.

Using F-FDN, video streaming providers only need to cache a base version of a video in an edge (fog) and process them to match the requested video processing service and characteristics of the viewers' devices in an on-demand manner. In addition, F-FDN can achieve location-aware caching (*i.e.,* pre-processing) of video streams. That is, video streams that are popular (*i.e.,* hot) in a certain region are pre-processed and cached only in that region. Due to resource limitations of FDN, we propose to

pre-process only the hot portions of videos [17] and the remaining portions are processed in an on-demand manner. To alleviate the on-demand processing load in an FDN, we develop a method to leverage the distributed nature of F-FDN and reuse pre-processed video contents on neighboring FDNs. The full ultilization of the F-FDN platform can be observed in Figure 1.3. This allows the streaming of different portions of a video from multiple sources (*i.e.,* FDNs), subsequently, increasing viewers' QoE.

1.4 Thesis Contributions

This thesis makes the following contributions:

- Development of an interactive cloud-based streaming platform (CVSE). The platform follows micro-service architecture and adopts serverless computing paradigm in a sense that users (*i.e.,* stream providers) who deploy CVSE do not need to worry about details of server configurations. CVSE is flexible from several aspects. In particular, it can be extended by new services defined by service providers. It can work under various computing platforms, and offers variety of billing options to viewers.

- Evaluation of CVSE platform when performing various stream processing tasks. We evaluate the performance of CVSE in accommodating new interactions (services) defined by streaming providers. We also experiment with the streaming performance under various computing engines, namely emulation, thread-based, and containers.

- Proposing F-FDN platform to improve QoE for viewers' located in distant areas.

The platform leverages regional popularity of video streams and creates a federation of fog computing systems to improve interactive video streaming QoE for viewers.

- Developing a method within each FDN to achieve video streaming from multiple FDNs simultaneously. The platform has the ability to stream cached video segments from multiple sources and at the same time processes some other segments. The platform makes decision probabilistically at the video segment level and determines the best way to stream a video segment to a requesting viewer.

- Analyzing the impact of F-FDN on the viewers' QoE, under varying workload characteristics. We evaluate the performance of F-FDN against CDN technology and several other baseline approaches for interactive video streaming. We demonstrate the impact of considering end-to-end latency which is composed of both communication and computation latencies. We evaluate the performance of F-FDN against streaming approaches that are oblivious to the latency imposed both by communication and computation.

1.5 Thesis Organization

This thesis is organized into the following chapters:

- Chapter 2 is a collection of background and related works in the literature. The contributions of these works and their relation to the work of this thesis will be presented.

- Chapter 3 provides an overview of the architecture and development of the CVSE platform. The means in which interactive video streams are provided and the different ways in which CVSE can be deployed are further detailed. We are preparing a journal paper to be submitted on development of CVSE platform.

- Chapter 4 details the architecture of the F-FDN platform. In addition, the probabilistic decision making method used for streaming video segments are explained. A number of experiments were performed testing F-FDN against alternative video delivery methods through emulation of video stream requests. This chapter of thesis has derived from the following publication:

  - Vaughan Veillon, Chavit Denninnart, Mohsen Amini Salehi, "F-FDN: Federation of Fog Computing Systems for Low Latency Video Streaming", In Proceedings of the 3rd IEEE International Conference on Fog and Edge Computing (ICFEC '19), Larnaca, Cyprus, May 2019.

- Chapter 5 concludes the work and outlines the future directions of the CVSE and F-FDN platforms.

Chapter 2: Background and Related Works

2.1 Overview

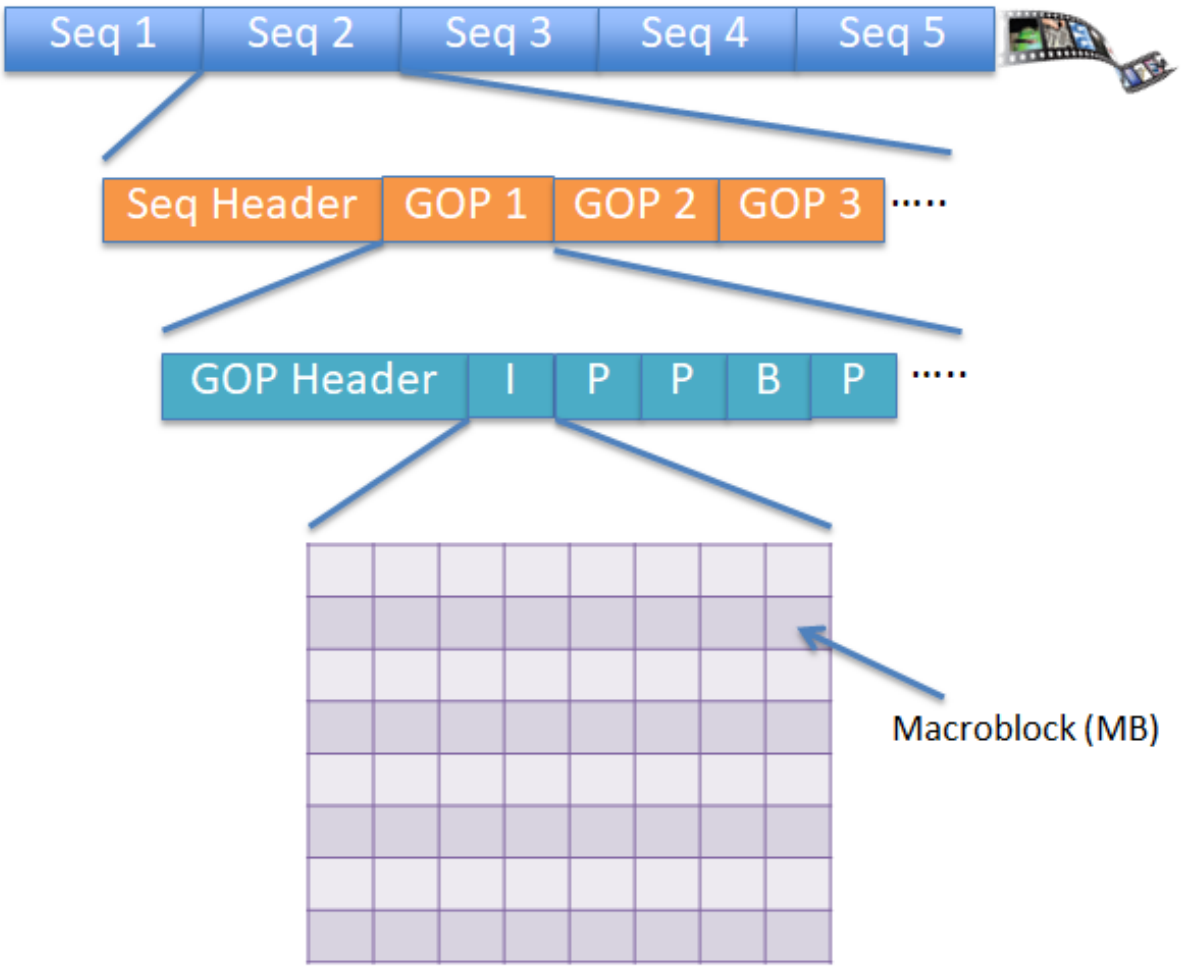
This chapter covers a number of background works in regards to the nature of how videos are streamed and the viewing behavior of viewers when videos are streamed. Additionally, a number of works related to this research are provided in this chapter. This thesis builds upon the field of cloud resource provisioning and utilizing certain fog computing practices that aim to enhance user experience.

2.2 Background

2.2.1 Video processing libraries. In CVSE we utilize a video processing software library known as FFmpeg. FFmpeg is a mutlimedia framework that allows CVSE to perform various video processing tasks (*i.e.,* decoding, encoding, transcoding, transmuxing, filtering, etc.) [18]. However, it is important to note that CVSE is not limited strictly to the use of FFmpeg as the sole video processing software. Based on the extendability of the architecture of CVSE, any new software can be integrated into the platform to be utilized in newly defined services.

2.2.2 Structure of video streaming. As shown in Figure 2.1 [2], a video stream consists of a number of sequences. Each sequence is divided into several GOPs. A GOP in composed of multiple frames beginning with the I (intra) frame, the rest of the frames consist of either P (predicted) frames or B (bi-directional predicted) frames. Every frame within a GOP has multiple slices that are composed of a number of macroblocks (MB) which is the basic operation unit for video encoding and decoding. There are two types of GOPs, open-GOP and closed-GOP. In the case of closed-GOP,

Figure 2.1. Structure of a video stream [2].



there exists no interdependence between GOPs, which allows each closed-GOP to be processed independently [2]. It is important to note that in this work, we use closed-GOPs in the videos that we stream. Due to this nature of GOPs, video streaming is achieved via processing and streaming independent video segments in the form of *Group Of Pictures* (GOP) [19].

QoE of the viewer is defined as the ability to stream each GOP within its allowed latency time to create an uninterrupted streaming experience. The allowed latency time for a GOP is its presentation time; hence, that is considered as the GOP's deadline [19, 20]. It is important to note that "GOP" and "video segment" are used interchangeably throughout this work.

A large body of research studies have been undertaken to maintain the desired video streaming QoE through efficient use of cloud services [12, 13]. Particularly, the earlier studies have shown that the access pattern of video streams follows a long-tail distribution [21, 17]. Meaning, only a small percent of videos are streamed frequently (known as *hot* video streams) and the majority of videos are rarely streamed [22]. For instance, in the case of YouTube, it has been shown that only 5% of videos are hot [23]. It has also been shown that, even within a hot video, some portions are accessed more often than others. For instance, the beginning portion of a video or a popular highlight in a video is typically streamed more often than the rest of the video [17].

Considering this long-tail access pattern, streaming service providers commonly pre-process (store) hot videos or GOPs in multiple versions to fit heterogeneous viewers' devices [17]. Alternatively, they only keep a minimal number of versions for the

rarely accessed videos [12, 24]. For any portions of the video that are not pre-processed, they are processed in an on-demand manner upon viewer's request [13].

A video stream of an interactive video, such as 360 degree or story branching videos, changes based on how a viewer is consuming it. These types of videos benefit greatly from lowered streaming latency. In general, if the latency is high, the viewer will need a higher amount of buffer to cover processing and streaming delay. Based on the nature of interactive videos, some parts of the buffer may end up not being viewed. Therefore, low latency streaming reduces the amount of buffer needed and thus reduces the amount of wasted processing.

2.3 Video Stream Scheduling

A number of works have been done regarding scheduling video streaming tasks on cloud. Video processing is computationally heavy to process. As such, many video streaming providers tend to outsource their storage, computation, and bandwidth requirements to clouds [25].

Jokhio *et al.,* [26] present a computation and storage trade-off strategy for cost-efficient transcoding in the cloud. For a given stream, they determine if it should be pre-transcoded, or should be processed upon request. Zhao *et al.,* [27] consider the popularity, computation, and storage costs of each version of a video stream to determine if it should be pre-transcoded or not. Both of these works demonstrate the possibility of lazy processing of videos streams. However, they do not study the general case of interactive video streaming and do not explore the impact of efficient scheduling and VM provisioning.

In systems with uncertain tasks arrival, scheduling can be performed either in the *immediate* or *batch* mode [28]. In the former, the tasks are mapped to machines as soon as they arrive whereas, in the latter, few tasks are batched in a queue before they are scheduled. Khemka *et al.,* [28] show that the batch mode scheduling outperforms the immediate mode in heterogeneous systems. In the batch-mode, tasks can be shuffled and they do not have to be assigned in the order they arrive. Nonetheless, currently, there is no batch scheduling method tailored for video streaming to consider their unique QoS demands.

Ashraf *et al.,* [29] propose an admission control and scheduling systems to foresee the upcoming streams rejection rate through predicting the waiting time at each server. The scheduling method drops some video segments to prevent video processing delays. In contrast, the admission control policy we propose assigns priority to video streaming tasks based on their position in the stream. Also, it is aware of the viewer's subscription type and is able to reject tasks with lower priority (*e.g.,* free viewer) to alleviate over-subscription of VMs.

2.4 Cloud Resource Provisioning for Video Stream

Resource (VM) Provisioning for Video Stream Processing. Previous works on cloud-based VM provisioning for video processing (*e.g.,* [30, 31]) mostly consider the case of offline video processing. Thus, they mainly focus on reducing the makespan (*i.e.,* total execution times) and the incurred cost. Netflix uses a time-based approach for VM provisioning on cloud [32]. It periodically checks the utilization of allocated VMs and scales them up by 10%, if their utilization is greater than 60% for 5 minutes.

The VMs are terminated by 10%, if their utilization remains less than 30% for 20 minutes. In [33, 25], a QoS-aware VM provisioning was proposed for lazy video stream processing. In [34, 35], authors provide an on-demand transcoding for cost-efficient live-streaming videos on geographically distributed clouds. Nonetheless, neither of these methods consider heterogeneous VMs offered by cloud nor they provide a generic platform that can accommodate any user-defined interaction on video streams.

With traditional CDNs, the CDN servers are only used to cache data. This implies that any update to the CDNs' contents is dependent on centrally located cloud servers. With the integration of fog/edge computing into a distributed system like a CDN, computation can be performed on the network edge, near data users. Having the computation performed closer to the data source reduces the streaming latency, hence, higher QoE [36]. Alternatively, our system utilizes fog computing to perform on-demand video processing to reduce video streaming latency.

Li *et al.,* [12] developed an architecture for CVSS, Cloud-based Video Streaming Service. CVSS utilizes cloud resources to deliver video streams through a balanced combination of on-demand processing and partial caching in order to minimize use of resources and maintain high QoE to viewers. Our work integrates CVSS into a distributed system in the form of F-FDN, where fog computing is performed by the CVSS located in FDN. In this work, the CVSS that was proposed has been further expanded, resulting in our video stream processing platform CVSE.

2.5 Distributed Fog Computing Systems

Lin *et al.,* [37] propose a system, called CloudFog, that utilizes fog computing to enable thin-client Massive Multiplayer Online Gaming while maintaining a high user QoE. Their system works by having powerful and centrally located servers perform the computational tasks that are associated with the game state. The less computationally intensive task of rendering and streaming game video is handled by intermediate machines (called supernodes) that are physically closer to the users. This enables a user to play the game without the need of a powerful device, since the heavy computation is handled within the system enabled by fog computing. Similar to [37] we stream video from physically close servers (*i.e.,* FDN). However, our work is different than [37] in the sense that our system has more intelligence in reusing contents on peering FDNs. This enables F-FDN to operate with a greater independence from the central cloud.

Ryden *et al.,* [38] provides an architecture for a fog system, called Nebula, designed for applications where user data is geographically spread. An example of this type of application is managing video feeds from multiple cameras spread amongst various locations. Nebula specializes in performing location-aware and and location-specific processing of data-intensive and compute-intensive tasks. They were able to fully utilize the fog machines by monitoring the data storage and computational potential of the machines by forming machine groups based on their proximity to neighboring machines. Their methodology allows for multiple machines to compute or store data that is only relevant to its location. In a similar fashion, F-FDN keeps updated knowledge of the data stored in its FDN. Unlike [38], we do not estimate the

storage or computational potential of our FDN, but the knowledge we maintain allows us to consider the cached video content of multiple FDN for every video being streamed.

Provensi *et al.,* [39] worked on a platform called Maelstream, a decentralized, self-organizing system that delivers media streams in a peer-to-peer manner. They focus on live streaming applications with users that consist of producers and consumers, such as webinars. Each node of Maelstream receives its media stream from neighboring nodes based on dynamic latency estimations and fair bandwidth usage. Similarly, F-FDN chooses the FDN from where videos are streamed based on accurate latency estimations in addition to the estimations from on demand video stream processing. Also, F-FDN follows a hybrid peer-to-peer and hierarchical structure as opposed to a purely decentralized nature that Maelstream has.

2.6 Summary

Many research works have been carried out to improve system performance utilizing fog computing, however, none of them have concentrated on video streaming in the ways we propose in F-FDN. To elaborate further on the way our on-demand video streaming engine (CVSE), which is utilized within FDNs, operates, in the next chapter, we explain architectural details and study the performance of CVSE.

Chapter 3: Developing CVSE

3.1 Overview

In this chapter, an interactive video streaming engine is developed (called CVSE). We present the challenges that are inherent to interactive video streaming. The goals of CVSE include:

- The ability to extend the platform with any video processing service even if added from third parties.
- Update the platform with these new service in a real-time manner.
- Allocate an appropriate amount of resources based on the current workload of the system.

We provide a detailed explanation of the CVSE architecture. In the final point of the chapter, we run tests to measure the performance of CVSE when executing various video processing tasks.

3.2 Challenges in Providing Interactive Video Streaming Services

While the research problems of cloud resource provisioning and scheduling have been thoroughly investigated (*e.g.,* in [40, 41, 42]), the challenges for video streaming are different in certain ways from existing solutions for other applications. In particular, video streaming tasks have unique QoE demands, distinct processing demands, and characterizable access patterns. In addition, video streaming has become a dominant, resource-demanding service and is used in various applications, from e-learning [43] to natural disaster management [44], and public broadcasting [34]. The prevalence of

video streaming (currently forms around 75% of the whole Internet traffic [45]) has generated a need for tailored solutions in order to provide efficient processing of videos.

The main challenge in this chapter is: *Is it feasible to provide interactive video streaming using cloud resources?* One approach to address this challenge is to pre-process all possible versions of each video. However, this approach is not always feasible for interactive video streaming. The main reason being, in *video on demand* (VOD), given the long tail distribution in accessing video streams [46] and the fact that viewers' requests can contain combinations of multiple configurations together, generating exhaustively all possible versions for each video is cost-prohibitive [27]. For instance, just to cover all display standards, for each video stream, 90 to 270 versions must be pre-processed and stored [47, 48]. In fact, pre-processing of video streams can be performed only for frequently-used interactive services, in a way called "lazy processing". The majority of streams (*i.e.,* those in the tail of access distribution) can be processed in an on-demand (*i.e.,* lazy) manner.

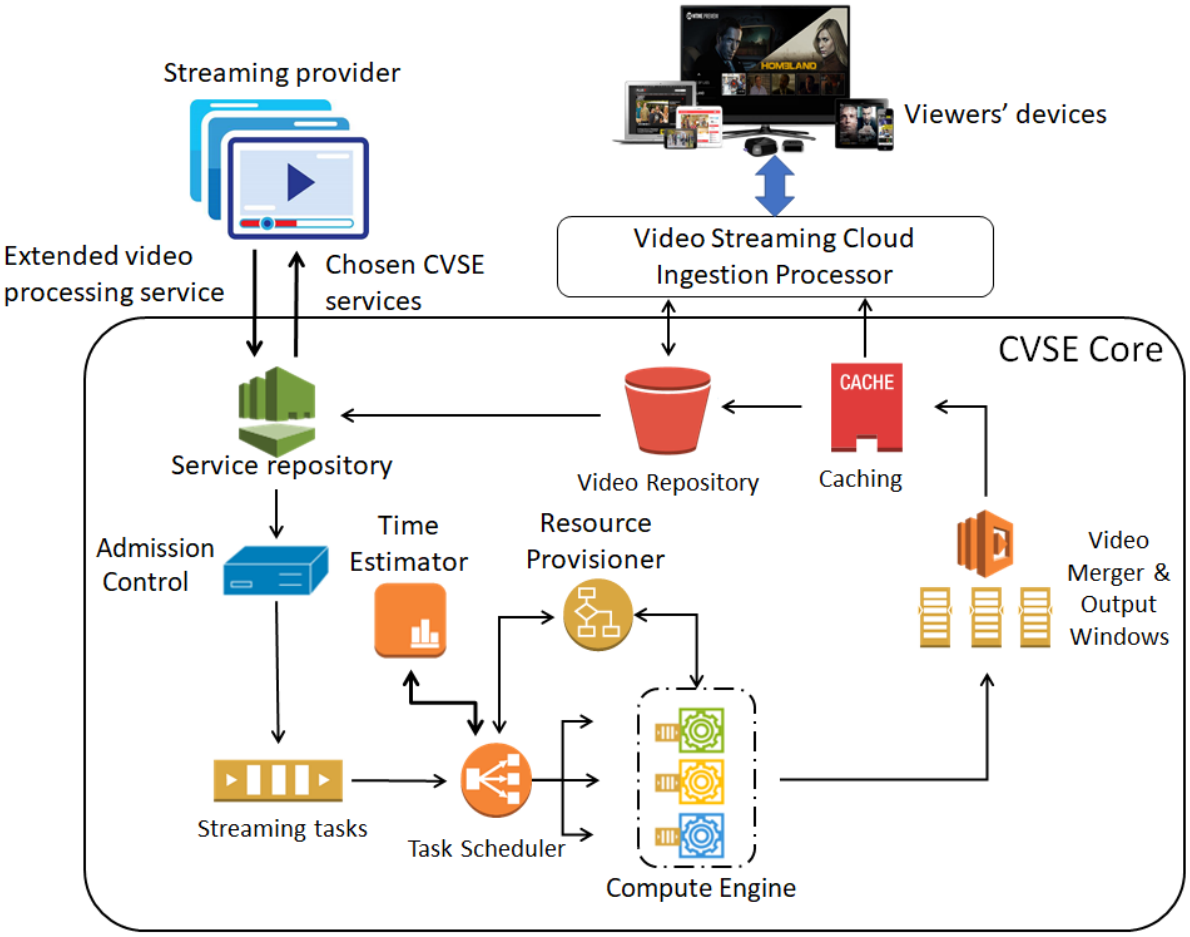
The challenge in lazy processing of video stream requests is two-fold: *first,* video stream processing is computationally expensive and requires large amounts of computational infrastructures, requiring a significant cost to the stream service provider [49]; *second,* the video processing has to be performed in a real-time manner (*i.e.,* with minimum latency) to guarantee the QoE demands of the viewers.

To address the *first* challenge, video stream providers commonly rely on cloud services for their computational demands [32]. Since cloud providers commonly provide heterogeneous computational services (virtual machines *(VMs)* and containers) that

can be matched with different interactive tasks and have diverse prices. To minimize the incurred cost, the stream service provider needs to acquire heterogeneous cloud resources that have the best match (*i.e.,* affinity) with the requested interactions.

For the *second* challenge, a stream service provider needs to assure that it can guarantee a certain QoE level, regardless of the viewer's geographical location, in the presence of budget constraints, and with uncertainties in request arrival. Robustness is defined as the degree to which a system can function correctly in the presence of uncertain parameters in the system [50]. An interactive video streaming service is *robust*, if it can guarantee a certain QoE level, regardless of the request arrival intensity and the geographical location of the viewers. The main QoE demands from interactive video playing is to have video streams delivered uninterrupted in a timely manner. That is, the video stream tasks must be completed within a short deadline, which is their presentation time. In addition, viewers generally judge the quality of a stream service provider based on the delay they perceive when streaming begins, termed the *startup delay*. Recent studies [46] show that around 60% of viewers watch only a few seconds of the stream before terminating it. Hence, the unique QoE demand of viewers is defined as *minimizing the start-up delay and the deadline miss rate*. To address the problem of delivering video streams while maintaining sufficient QoE to geographically spread viewers, we developed the F-FDN platform which utilizes CVSE. The F-FDN platform is further explained in Chapter 4. While the challenge of maximizing cloud resource usage is addressed in the CVSE platform through a combination of on-demand processing and partial caching of pre-processed video streams explained in this chapter.

Figure 3.1. System components of the Cloud-based Video Streaming Engine (CVSE)



3.3 Architecture of CVSE

The individual components and the ways in which stream service providers and viewers interact with CVSE are presented in Figure 3.1. Firstly, each viewer issues a streaming request for a particular video that needs to be processed to the viewer's specifications. Upon receiving a streaming request from a viewer, multiple video streaming requests are generated. Each request is a Group of Pictures (*GOPs*) [51] processing requests. Each GOP can be processed independently and it has its own individual deadline which is its presentation time. For each video stream, the Admission Control component assigns a priority value to all its GOPs and dispatch it to the scheduling queue. The Resource Provisioner component allocates the appropriate resources from cloud (*e.g.,* containers and VMs), forming a cluster of worker nodes to execute the processing of GOPs. Each worker node maintains a local queue to pre-load GOPs' data and execute assigned tasks in order. The Resource Provisioner component monitors the performance of the worker nodes and adaptively configures the size of the cluster based on workload intensity. The Time Estimator provides predictive information on the affinity of GOP streaming tasks based on historical execution times. The Scheduler component uses the estimated execution time information for efficient assign tasks to worker nodes. Video Merger collects the processed streams using an output window for each stream. In the event that a GOP is delayed (*e.g.,* due to failure), Video Merger asks the Admission Control to resubmit the GOP request with urgent priority. The Caching component decides if a part of, or the whole processed stream should be stored and reused.

3.3.1 Architectural components of cvse. CVSE is able to achieve interactive video streaming through a number of individual components working in tandem as shown in Figure 3.1. The function of each component is explained below.

- Ingestion Processor - handles all stream requests being made by viewers. The processing of the video stream is determined by the characteristics within a viewers' request.

- Video Repository - contains a copy of each video available on the CVSE platform at an unspecified setting. Once a video stream request is made, then a copy of the video will be processed to the specifications of the video stream request.

- Caching - contains cached, pre-processed portions of videos in different versions based on their popularity. For parts of a video that are not cached, they are processed on-demand.

- Service Repository - the collection of processing types that the CVSE can perform. As shown the Figure 3.1, new stream processing services can be defined and extended within CVSE by third parties (*e.g.,* stream service providers).

- Admission Control - assigns a priority level to each video segment based on their level of urgency.

- Time Estimator - collects and maintains historical execution data to estimate execution time of new tasks.

- Task Scheduler - distributes streaming tasks to available worker nodes based on

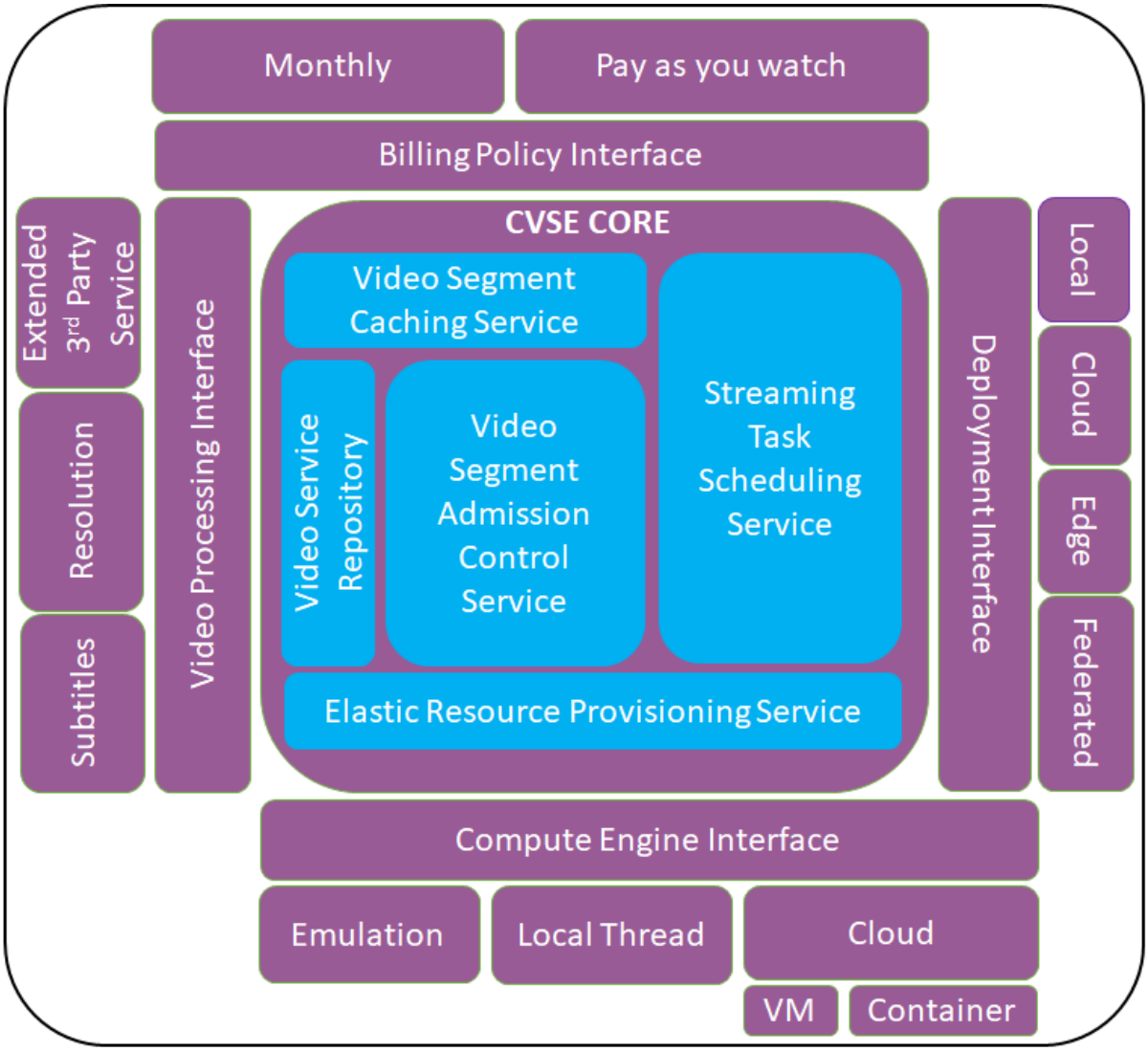
scheduling policies, the current workload of resources, task's meta data and its assigned priority.

- Resource Provisioner - allows the platform to dynamically allocate or deallocate heterogeneous resources in the Compute Engine based on processing demands.
- Compute Engine - performs the processing of video segments to the specified characteristics.
- Video Merger and Output Windows - location where the video segments of the stream are copied to once processing is finished. These segments are then sorted and streamed to the viewer.

CVSE also contains various interfaces in which the platform can be extended. These interfaces can be observed in Figure 3.2. The details of these interfaces are explained in the following subsections.

3.3.2 Video processing interface. The Video Processing Interface allows CVSE to perform any type of video processing that a stream service provider would like to offer its viewers. When a stream service provider chooses to utilize CVSE as its video streaming engine, any form of video processing offered to viewers is made available within the Video Service Repository of CVSE. When the stream service provider wants to offer a new type of video processing that is not readily available, they can send the request through the Video Processing Interface of CVSE. The request needs a defined unique processing service name, required software, and the command that must be executed to perform the processing.

Figure 3.2. High level view of interfaces that Cloud-based Video Streaming Engine (CVSE) deals with. These interfaces provide the extensibility to CVSE.



For example, if a stream service provider would like to provide subtitles in a foreign language that is currently not offered in CVSE, then they needs to define the video processing task of providing said subtitles to a video. Once the processing task is defined, it can be added to the CVSE as a new service. When the CVSE acknowledges the request, the newly defined service can be offered to viewers.

3.3.3 Compute engine interface. The Compute Engine Interface enables the worker nodes that perform the video processing of CVSE to exist in various forms. Currently we have four options in which CVSE can allocate its compute engine: emulation, local threads, VMs, and containers. Emulation mode allows us to run benchmarks on CVSE, observing how CVSE operates in each configuration (such as during oversubscription) without utilizing large amounts of resources in the process. The local thread compute engine option creates threads as the worker nodes for video processing on the host machine that is also running CVSE. This option is primarily used for in-house testing or when the workload is not enough to warrant even a single dedicated worker node. CVSE can outsource its compute engine to cloud resources through either VMs or containers operating as its worker nodes. This is achieved by CVSE establishing a socket connection to the worker node daemon running in the VM or container. Once the connection is established, CVSE can send any number of streaming tasks to the worker nodes of the compute engine. CVSE will also be able to accommodate additional compute engine options as they are continuously developed and become readily available (*e.g.,* serverless computing service).

3.3.4 Deployment interface. The Deployment Interface allows for CVSE to be deployed in different capacities. The ways in which CVSE can be deployed allows for the platform to be flexible in accommodating the needs of stream service providers whether it be large-scale or small-scale. We currently have four options in which CVSE can be deployed: local, cloud, edge, or federated deployment. The local deployment option allows CVSE to operate on a desktop, laptop, or any local machine. Similar to the local deployment, the cloud deployment option allows CVSE to operate within public cloud resources (*e.g.,* Amazon Web Services, Microsoft Azure, Google Cloud Platform) or private cloud servers.

The edge deployment of CVSE pertains to a stream service provider maintaining a geographically distributed system in which there are a number of edge servers that are located physically close to viewers. In this type of distributed system, a form of CVSE is deployed in each of these edge servers. Finally, the federated deployment of CVSE is similar to the edge deployment, however there is also a certain level of communication and coordination that is performed between the edge servers in the distributed system. The nature of the federated and edge deployment are further expanded upon in Chapter 4.

3.3.5 Billing interface. The Billing Interface allows for stream service providers to tailor the costs incurred to their viewers when using CVSE. CVSE offers two main forms of billing: a monthly charge in the form of a subscription, or a *pay-as-you-watch* model. The monthly billing policy allows for unlimited amounts of streaming at whatever tier of subscription level the viewer has paid. The pay as you

watch billing policy allows for viewers to be charged not only by the amount of time that videos are streamed, but also by the type of video processing that is performed for the video stream. Complicated types of video processing require more resources, therefore, they are be more expensive than simpler tasks.

We note that our CVSE platform enables the pay-as-you-watch billing manner. This type of billing is fairer and fits viewers who sporadically use the streaming service (*e.g.,* to watch movies). In this case, CVSE has the ability to charge users based on the amount of cloud resources they have utilized to offer their requested services. It is possible that third-party companies emerge whose job is only offering new services on video streams and they charge for making use of their services, in addition to the cost of cloud resources.

3.4 Implementation

CVSE is implemented primarily in Java. It utilizes ffmpeg (multimedia conversion library) [18], and other video processing utilities through the calling of batch scripts. All communications from front end (*i.e.,* viewers and streaming service providers) are through web-services. All communication between the components that span across multiple machines is achieved through socket connections. Communication with computing resource provider (*i.e.,* cloud service provider) are made through official API of each platform.

CVSE is designed and implemented in a modular and extendable way. Each main components are implemented as a class interface, there are several different implementations which extend the same interface. Therefore a suitable variant of each

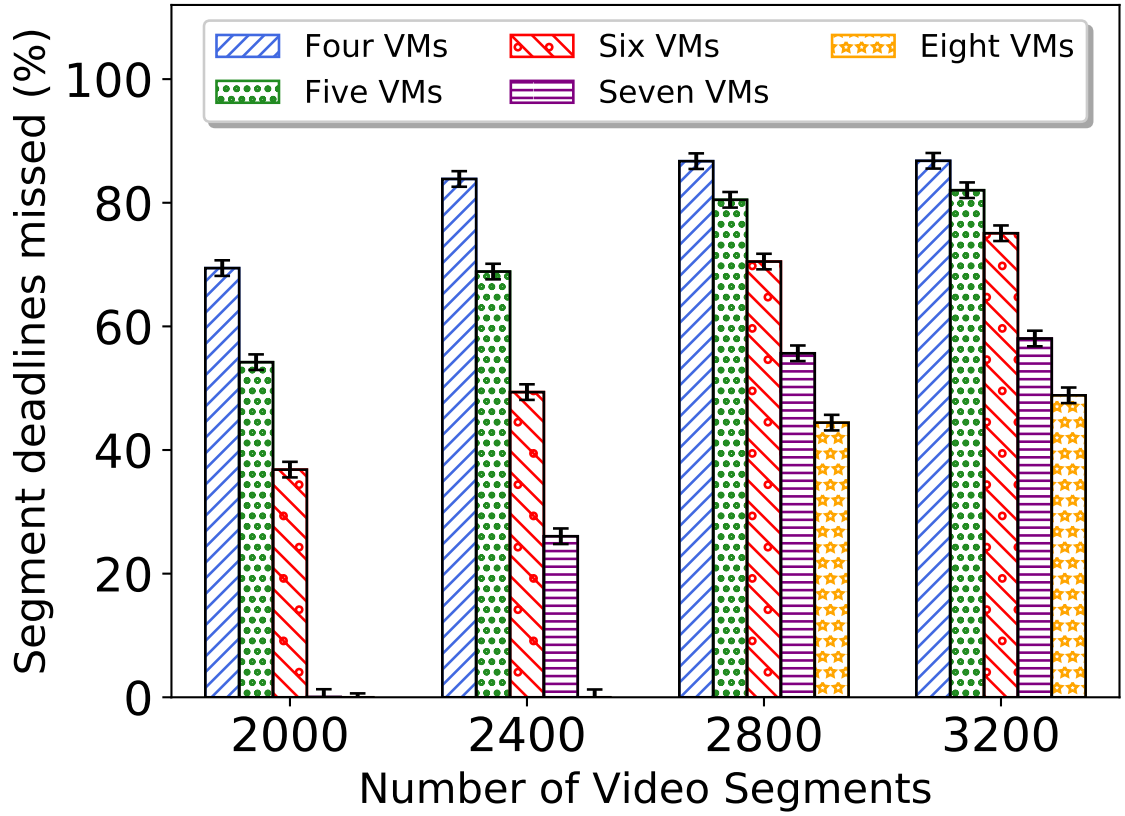
component can be selected at the deployment stage.

For example, the time estimator must have a function to estimate the execution time based on GOP request and machine type. The most basic version returns a constant number based on operation type. A more sophisticated version allows for more detailed estimation based on various offline-learned historical data tables and other various factors. An experimental version learns and adapts its estimation on the fly as the the CVSE continues to run. This allows for CVSE to be not only a practical video streaming platform, but also a platform for experiments to research in depth for each component. For instance, when focusing on task scheduling policy, CVSE can be deployed with the caching system disabled and Resource Provisioner is set to not gain extra resources in face of oversubscription. With the ability to isolate each component of CVSE, we are able to focus more specifically on system weaknesses and strengths when testing new features.

3.5 Experiments

In order to show the performance of CVSE, we ran tests to demonstrate the dynamics within the platform. In Subsection 3.5.1, we show CVSE operating in emulation mode. We show the impact on performance within CVSE when the number of worker nodes in the compute engine varies during oversubscription. To show the results of video stream processing by CVSE, in Subsection 3.5.2 we provide an example of CVSE operating in local thread mode as a part of a web demo. Here we demonstrate CVSE processing a video to different settings requested by a viewer.

Figure 3.3. Deadline miss rate of CVSE in emulation mode with varying numbers of worker nodes in the compute engine. Each VM's processing capability is based on the AWS g2.2xlarge VM type.



3.5.1 Analyzing worker node cluster size. CVSE needs to be sensitive to the QoE requirements for each video stream requests that is being processed. In order for compute engine of CVSE to sufficiently meet these QoE demands, enough resources must be allocated in the form of worker nodes, which is handled by the resource provisioner. An additional consideration of CVSE is to be cost-efficient with its allocation of computational resources. If an excess of worker nodes are allocated based on the current workload of the platform, then there are resources are wasted.

These factors dictate the balance that the resource provisioner must keep to neither over-allocate nor under-allocate computational resources (*i.e.,* size of the cluster of worker nodes) when CVSE is processing video segments. In Figure 3.3 we ran CVSE in emulation mode against a number of workload testbenches. The lowest workload starts at 2000 video segments and the workload increases by 400 total segments up to 3200 total segments. In this experiment, the cluster size of the worker nodes were set to a fixed number in order to show the relationship of resource allocation and oversubscription. It is important to note that the processing power of worker node within the compute engine of CVSE is modeled after the AWS g2.2xlarge VM type.

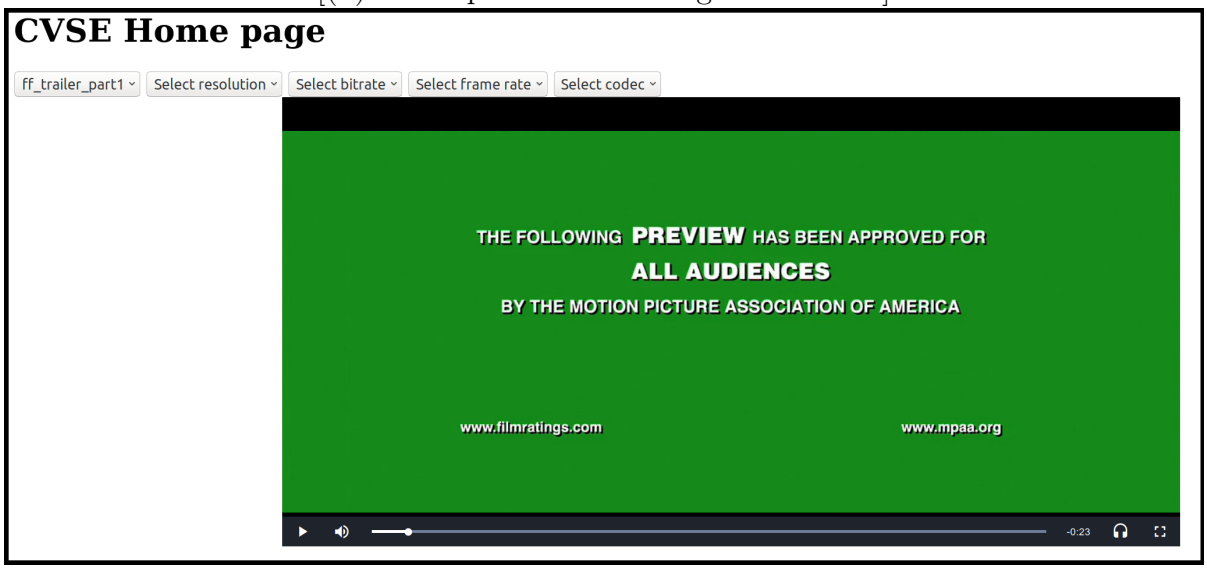
The results show a dramatic difference in deadline miss rate for particular cluster sizes once workloads reach a certain volume. For instance when the cluster of worker nodes consists of 7 VMs, at 2000 total video segments the deadline miss rate is close to 0%. However when the workload is increased to 2400, the deadline miss rate for the 7-VM cluster size increases to approximately 26%. These results show that dynamic allocation of resources within the compute engine of CVSE is crucial to maintaining

performant QoE standards.

Figure 3.4. Home page of CVSE Web Demo.

[(a) Video processed to a low resolution.]

[(b) Video processed to a high resolution.]



3.5.2 Web demo. In Figure 3.4 we provide screenshots of a web demo of CVSE. This demo processes videos to various requested settings, then allows the viewer to watch the processed video. The demo consists of two parts: 1) A Java project that

runs CVSE as a web service with an open socket that can receive incoming stream requests, 2) A web page hosted on a local web server that sends streaming requests to the web service, and then plays the processed video provided by CVSE. In this demo, CVSE is being run using local threads as the worker nodes of its compute engine. In Subfigure 3.4 (a), we show a video that CVSE has processed to a very low resolution. In Subfigure 3.4 (b), the same video is shown, but the video has been processed by CVSE at a high resolution.

3.6 Summary

In this chapter, we addressed the feasibility of an interactive video streaming using cloud resources through the development of the platform CVSE. We have shown the extensibilty of CVSE in multiple capacities, namely in its deployment, video processing services, and billing policies. We also highlighted the importance of proper allocation of cloud resources within the CVSE platform. In the following chapter, we address the challenge of providing interactive video stream while maintaining low latency to geographically spread viewers.

Chapter 4: Federated Fog Delivery Networks (F-FDN)

4.1 Overview

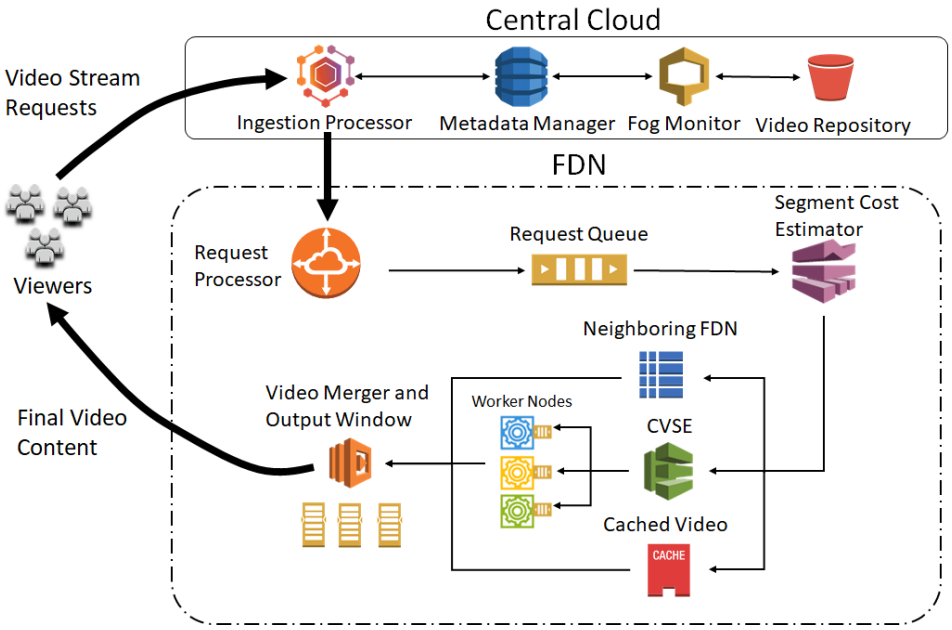
The aim of F-FDN is to deliver the highest possible QoE to viewers, independent of their geographical location. We position F-FDN to achieve this by utilizing all of the resources that we have available in the system. One of the most differentiating qualities of F-FDN compared to other video streaming systems is the ability to evaluate on a segment by segment basis how to stream a video. It is worth noting that, in traditional CDNs, the entire video is always streamed from the same CDN server as long as a connection is maintained with the viewer [52].

F-FDN is composed of several connected, peer Fog Delivery Networks (FDNs) that are also connected to a central cloud server. An FDN caches pre-processed GOPs for videos that are hot in that region. This results in each FDN having varied pre-processed video content that is optimized to the viewers local to that FDN. When a video stream is requested, the viewer is connected to its most local FDN. As the video is being streamed, decisions are made on a segment (GOP) by segment basis as to how the GOP is delivered to the viewer. The aim of these decisions is to maximize the likelihood of meeting each GOP's deadline. The process in making these decisions is described in detail in Section 4.2

An example of a video being streamed to a viewer from multiple sources can be observed in Figure 1.3. As we can see in the figure, using F-FDN, a video segment can be streamed to a viewer in three different ways:

1. FDN Local Cache - the FDN local to the viewer already has the requested video

Figure 4.1. System Components of the F-FDN architecture. It is composed of a centrally located cloud and a federation of multiple Fog Delivery Networks (FDNs).



segment in it's cache and streams the segment to the viewer.

2. Processed On-demand - the local FDN processes the missing (*i.e.*, non-cached) segments according to the characteristics of the request and then stream them to the viewer.

3. Neighboring FDN's Cache - the missing segments exist in the cache of a neighboring FDN, the segment is then transferred to the local FDN and then streamed to the viewer.

At a high level, F-FDN is composed of two main components, namely a Central Cloud and a distributed network of FDNs. Figure 4.1 shows the internals of the Central Cloud and each FDN. Details of each component is elaborated in the next subsections.

4.1.1 Central cloud. The Central Cloud, with virtually unlimited storage and processing capabilities, is where all streaming requests are initially ingested and where all FDNs in the system are managed. As we can see in Figure 4.1, Central Cloud consists of four main components:

- Ingestion Processor - handles all the incoming stream requests made by viewers. It determines which FDN is the most local to the viewer in order to start the video streaming process. The FDN that is selected is the one that determines the way each video segment is streamed to the viewer.
- Metadata Manager - keeps track of all the cached video segments contained throughout the FDNs. In addition, it keeps track of other metadata, such as file size and network latency between neighboring FDNs. In Section 4.2, we explain how this metadata helps in determining the way to stream a video to the viewer.
- Fog Monitor - keeps track of the availability of FDNs via sending heartbeat pings to them. Also, it evaluates network latency between FDNs which is then communicated to the Metadata Manager to maintain up-to-date information.
- Video Repository - contains a repository of all videos, pre-processed in multiple versions. This is where any cached video content on FDNs originates from. In the event that one or more video segments are missing in a local FDN, one decision can be fetching the segments from the video repository of the Central Cloud.

4.1.2 Fog delivery network (fdn). Each FDN consists of six components, as shown in Figure 4.1 and explained below.

- Request Processor - receives video stream requests from the Central Cloud and puts them into a request queue.

- Segment Cost Estimator - makes the decision as to how each video segment in a video stream request should be streamed to viewers with minimum latency (*i.e.,* maximum likelihood of meeting the segment's deadline). For a given video segment, it is determined whether the segment should be streamed from the local FDN's cache, processed on-demand by the local FDN, or retrieved from a neighboring FDN's cache and then streamed by the local FDN. The process by which these decisions are made is explained in Section 4.2

- Neighboring FDN Metadata - contains knowledge of all cached video segments on other FDNs and the network latency for accessing them. It is worth noting that the latency of accessing Central Cloud is also maintained by this component.

- On-demand Processing Engine - is in charge of on-demand processing of video streams. We have developed the engine in our prior studies [12, 13]. The engine uses multiple worker Virtual Machines (VMs) and enables the FDN to process incoming video segments based on the characteristics of the viewer's device.

- Cached Video Segments - GOPs of videos that are determined to be hot [17] in a region are pre-processed and cached by the FDN. For a given video streaming request, if some of the segments are locally cached, they impose the minimum network latency and cause higher QoE for the viewer.

- Video Merger and Output Window - where the video segments of the video stream are put in the correct order and then streamed to the viewer.

4.2 Maximizing Robustness of F-FDN

Streaming service providers aim at providing an uninterruptable streaming experience to their viewers. They strive to avoid and minimize missing deadlines of streaming tasks. The distributed nature of F-FDN provides multiple options (sources) to stream a single video segment. To minimize missing the presentation deadline of a video segment, it should be streamed from the source that imposes the minimum streaming latency; hence, offering the maximum probability to meet the segment's deadline.

The streaming latency is affected by two main factors, namely video segment processing time and transfer time across the network. In particular, both of these factors have a stochastic nature [53]. An ideal method to stream videos in F-FDN should be robust against these stochasticities. That is, the method should maintain its performance, in terms of meeting the deadlines of streaming tasks, even in the presence of these uncertainties. We implement a method within FDNs to account for the uncertainties of F-FDN and maximize the probability of meeting the deadline for a given streaming task. This method is utilized within the Segment Cost Estimator component of FDN and makes the FDN robust.

4.2.1 Network latency of streaming a video segment in fdn. For a given video segment i, the latency probability distribution of transferring it between two points can be obtained based on the segment size (detnoted s_i) and the amount of data that can be transferred within a time unit between the two points (*i.e.,* network throughput). Prior studies show that the latency probability distribution to transfer

Table 4.1. Important symbols used in Section 4.2

Symbol	Description
s_i	size of video segment i
r_i	probability of processing segment i on time (robustness)
$N_i^C(\mu_i, \sigma_i)$	overall delivery distribution (end-to-end latency) for segment i
$N_i^\tau(\mu_{jv}, \sigma_{jv})$	distribution of network throughput between two points for a given segment i
$N_i^E(\mu_{ij}, \sigma_{ij})$	processing time distribution for segment i

video segment i follows a normal distribution (denoted N_i^τ) [54, 14]. The two points can be between two FDNs or an FDN and a viewer.

4.2.2 Robust video segment delivery in f-fdn. We formally define *robustness* of segment i, denoted r_i, as the probability that segment i is delivered to the viewer's device before or at its deadline δ_i.

As mentioned earlier, each video segment can be retrieved using one of the following choices: (A) from the FDN's local cache; (B) processing it on-demand in the local FDN; (C) from a neighboring FDN's cache.

In choice (A), the robustness of delivering segment i is obtained from the segment latency probability distribution between local FDN j and viewer's device v. As such, the probability distribution for delivering a segment to the viewer, denoted $N_i^C(\mu_i, \sigma_i)$, for choice (A) is determined using Equation 4.1.

$$N_i^C(\mu_i, \sigma_i) = N_i^\tau(\mu_{jv}, \sigma_{jv}) \tag{4.1}$$

In choice (B), the latency is impacted not only by the segment latency probability distribution between FDN j and the viewer's device v, but also by the time

to process the segment in FDN j. The processing times of a video segment can be estimated based on a probability distribution. This distribution is obtained from historical execution times of a particular processing type (*e.g.,* bit-rate transcoding) for a segment. It has been shown that the processing time of a video segment exhibits a normal distribution [14]. Let $N_i^E(\mu_{ij}, \sigma_{ij})$ be the probability distribution of completing the processing of segment i on FDN j; also let $N_i^T(\mu_{jv}, \sigma_{jv})$ be a normal distribution representing latency to deliver segment i from the local FDN j to the viewer's device v. Then, the probability distribution of delivering segment i to the viewer is calculated by convolving the two distributions as shown in Equation 4.2.

$$N_i^C(\mu_i, \sigma_i) = N_i^E(\mu_{ij}, \sigma_{ij}) * N_i^\tau(\mu_{jv}, \sigma_{jv}) \tag{4.2}$$

Similarly, the latency for choice (C) is impacted by two factors, the latency distribution for retrieving a segment from a neighboring FDN k to the local FDN j, denoted $N_i^T(\mu_{kj}, \sigma_{kj})$, and the segment latency distribution between FDN j and viewer's device v, denoted $N_i^\tau(\mu_{jv}, \sigma_{jv})$. Therefore, to obtain the probability distribution of delivering segment i, we convolve these two probability distributions as shown in Equation 4.3.

$$N_i^C(\mu_i, \sigma_i) = N_i^\tau(\mu_{kj}, \sigma_{kj}) * N_i^\tau(\mu_{jv}, \sigma_{jv}) \tag{4.3}$$

Once we have the final distribution, $N_i^C(\mu_i, \sigma_i)$ the robustness of segment i can be measured using its deadline δ_i based on Equation 4.4. In fact, in this case the liklihood that segment i can be delivered before δ_i is the cumulative probability for a

random variable X to be less than or equal to δ_i which is the robustness of segment i.

$$r_i = P(X \leq \delta_i) \tag{4.4}$$

The algorithm for the Segment Cost Estimator utilizes the robustness for each segment of a video stream to determine how to fetch that segment, hence, assuring a high quality and uninterruptable video streaming experience for viewers. The algorithm first checks if segment i exists in the local cache of FDN j to be streamed to the viewer (Step 1). If it does not exist locally, then a list of all neighboring FDN containing segment i is retrieved from the Metadata Manager and their respective robustness values are calculated (Steps 2—6). The robustness of on-demand processing segment i is also calculated and compared against the neighboring FDN that has segment i with the highest robustness (Steps 7—8). Finally, in Step 9, the option with the highest robustness is chosen to provide segment i. The individual steps of the algorithm are provided below.

Upon receiving a video stream request m, at FDN_j and for every segment i in video stream request m:

(1) if i exists in FDN_j's local cache, stream i to viewer

(2) else if i is available in neighboring FDNs or in central cloud:

(3) retrieve list of metadata of all remote locations that match segment i

(4) For each metadata item l in the metadata list:

(5) calculate the cumulative probability of the transfer time from l using the presentation time of i

(6) track which FDN l offers the highest probability of success

(7) convolve processing and transfer time distributions for segment i in FDN_j and calculate its cumulative probability

(8) compare the probability of processing i on-demand with the probability of streaming i from FDN offers the highest probability of success

(9) stream segment i from the option with the highest probability of satisfying $i's$ deadline

4.3 Methods for Video Streaming Delivery

Table 4.2. Characteristics of various methods implemented to examine the performance of the F-FDN platform.

Methods \ Characteristics	Caching at Edge	Federated	On-demand Processing	Robustness Consideration
Central Cloud	no	no	no	no
CDN	yes	no	no	no
Federated CDN (F-CDN)	yes	yes	no	yes
Isolated FDN (I-FDN)	yes	no	yes	yes
Deterministic F-FDN	yes	yes	yes	no
Robust F-FDN	yes	yes	yes	yes

In this section, we explain alternative methods for stream delivery. Table 4.2 provides an overview for the various methods we implemented and highlights differences in their characteristics. These methods encompass current practices for video streaming (namely, CDN and Central Cloud) and baseline methods (namely, F-CDN, Isolated FDN, and Deterministic F-FDN) that focus on various aspects of the F-FDN platform in isolation. Finally, the *Robust F-FDN* is the streaming delivery method operating based on the theory developed in Section 4.2. It is noteworthy that these methods are implemented within the *Segment Cost Estimator* component of the FDN (see Figure 4.1). The rest of this section further elaborates on the characteristics of the implemented methods that are used in the experiment section.

Central Cloud.

This method considers only the central cloud where all the video contents are available in the main video repository. Every video segment is streamed directly from the cloud and no geographically spread FDN or CDN are considered to reduce the

streaming latency.

CDN. Due to popularity of the CDN approach in the streaming industry, we consider it in our evaluations. Our simulated CDN consists of a central cloud that holds the same characteristics as the previously described system and CDN servers which have 75% of the requested videos cached, which is a realistic level for CDN caching [55]. As CDNs are located close to viewers, any segments streamed from them have a lower latency compared to the central cloud. It is noteworthy that CDN servers do not perform any computation and caches the entirety of a video, rather than only a portion. Also, any segment that is not found in a CDN, is streamed from the central cloud.

Federated CDN. The Federated CDN (F-CDN) includes a central cloud and CDN servers in its system. The key difference of F-CDN with CDN is partial video caching. In F-CDN, it is possible to cache only few segments of a video, rather than the entirety of the video. Owing to the federated nature, in F-CDN, video segments can be streamed from the local CDN, a neighboring CDN, or from the central cloud. The rationale of implementing this method is to study the impact of federation of cached contents, without the ability to process videos on-demand. This method makes use of the robustness definition, introduced in Section 4.2, to stream a given video segment from the CDN that offers the highest probability to meet the deadline of that segment.

Isolated FDN (I-FDN). The Isolated FDN method includes a central cloud and a single FDN. In this system, the FDN node performs on-demand processing of video segments, in addition to caching. However, it does not consider retrieving segments

from neighboring FDNs. That is, the segments are streamed only from the FDN's cache, processed on-demand, or from the central cloud. The streaming decisions for each video segment is made between the local FDN and central cloud based on the robustness definition (see Section 4)2 The rationale of implementing this method is to study the impact of lack of federation on the streaming QoE.

Deterministic F-FDN.

The Deterministic F-FDN method consists of a central cloud and a federation of FDNs. While each FDN can perform caching and on-demand processing, the federation enables the option to stream cached segments from neighboring FDNs as well. In the Deterministic F-FDN, for each segment, streaming decisions are only made based on expected transmission and processing times, *i.e.,* it ignores the stochastic nature that exists in the F-FDN environment. This method demonstrates the impact of ignoring uncertainties that exist in the system on the overall streaming QoE.

Robust F-FDN.

Unlike Deterministic F-FDN, the Robust F-FDN operation takes into account the stochastic nature that exists in both communication and computation of the F-FDN platform. The more informed decision making is expected to have more streaming tasks meeting their deadlines, resulting in a more robust streaming service, regardless of the viewers' geographical location. It is, in fact, the implementation of the theory developed in Section 4.2 and the method described in the algorithm of Figure 4.2.2

4.4 Performance Evaluation

Experimental Setup

We conducted an emulation study to understand the behavior of the F-FDN platform. We developed a prototype of F-FDN by expanding the CVSS (Cloud Video Streaming Service) platform [13, 12, 56]. The prototype has the ability to operate for all video streaming methods described in Section 4.3. Within CVSS, we simulated three worker VMs that are modeled after the Amazon GPU (`g2.2xlarge`) VM to perform the video processing. The reason we considered GPU-based VMs is that in [14] it is shown that these VM types fit the best for video processing tasks. Our experiments are conducted by using three FDNs in the system, in addition to a central cloud server. Our experiments consider streaming requests arriving to one of these FDNs and we measure the performance metrics obtained in that FDN. The other two FDNs serve as neighbors, caching a portion of video segments (as detailed in Subsection 4.4.1).

We generated different workload traces of video stream requests to examine behavior of the system under various workload conditions and streaming methods. The workloads used in the experiments are created using a set of benchmark videos that contain different lengths and content types. The benchmark videos are publicly available for reproducibility purposes at `https://goo.gl/TE5iJ5`. For each video segment in the workload traces, there is an associated processing (*i.e.,* execution) time, which is obtained from the mean of 30 times execution of that segment on Amazon GPU VM. For the sake of accuracy and to remove uncertainty in the results, we generated 30 different workload traces. Each workload trace simulates 3 minutes of

video stream request arrivals. The arrival time of each streaming request is determined by uniformly sampling within the time period. All segments of the same video have the same arrival time but different deadlines (*i.e.,* presentation times). Accordingly, each experiment is conducted with the 30 workload traces and the mean and 95% confidence interval of the results are reported.

We track the number of deadlines that are missed, which indicates the robustness of the system. A deadline is considered missed due to a segment being streamed after its associated presentation time. The presentation time of a segment is based on the order of the segment's appearance in a video. As we consider a Video On-demand streaming service, even if a segment misses its deadline, it still must complete its execution and is streamed to the viewer.

To consider bandwidth usage in the evaluations, we have a limited bandwidth value from the local FDN to the viewer, and from the local FDN to other neighboring FDN. This bandwidth value becomes more congested as segments are initially streamed and less congested as segments finish streaming. Each node also has an associated latency value. The network latency values used for the edge servers and the central cloud server were taken from [5, 57]. For our bandwidth values, we used an average of 1 Gbps.

Experimental Results

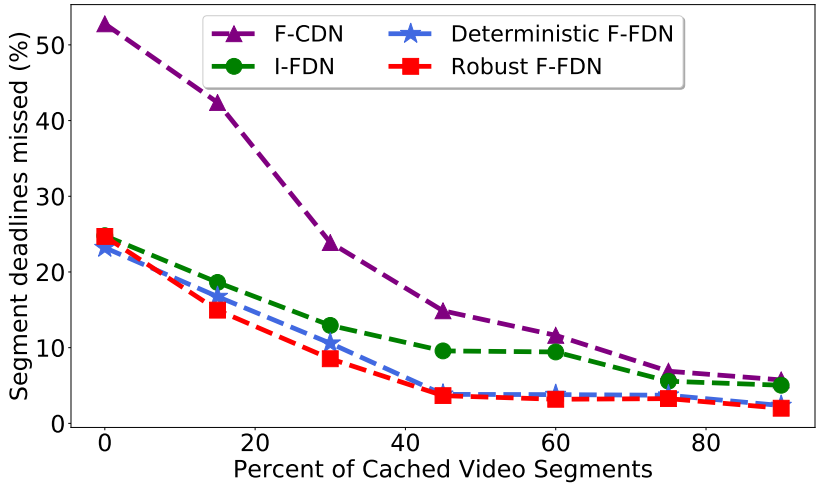
4.4.1 Analyzing suitable cache size for fdns. In our first experiment, we intend to find the minimum percentage of video contents that needs be cached within an FDN, so that a high level of QoE is maintained for viewers. Recall that we measure

QoE in terms of percentage of video segments missing their presentation deadline. We evaluate variations of FDNs (namely, F-CDN, I-FDN, Deterministic F-FDN, and Robust F-FDN) to understand how different methods take advantage of the caching feature. For that purpose, as shown in Figure 4.2, we increase the percentage of video segments that are cached in each FDN (horizontal axis) and measure the percentage of video segments that miss their deadlines (vertical axis). In this experiment, we used workload traces consisting of 3,500 segments being streamed to viewers.

We observe that as the percentage of cached segments is increased, the deadline miss rate drops remarkably across all methods—from approximately 53% to around 2%. Specifically, when the total cached video content is at zero percent, we are able to see a major difference between F-CDN and the other three systems. Zero percent caching for F-CDN, in fact, shows the case of streaming only from the central cloud. However, we can observe that other methods with the ability to process segments at the fog (FDN) level, in addition to streaming segments from cloud, can dramatically reduce deadline miss rate (approximately 52% improvement).

As the level of cached video content is increased, we observe the benefit of streaming video segments from neighboring FDNs. For instance, comparing the I-FDN and Deterministic F-FDN shows that at 30% caching, deadline miss rate is reduced by 2.3% (denoting 18% improvement), whereas at 90% caching, the deadline miss rate of Deterministic F-FDN is 2.7% lower than I-FDN (denoting 53% improvement). We can conclude that the streaming of video segments from neighboring FDNs unburdens the on-demand processing of the local FDN to the point where missing a deadline becomes

Figure 4.2. Deadline miss rate of different streaming methods as the caching level is increased. The simulations are run using a work load of 3500 segments.



significantly less likely.

Based on our observation and analysis in this experiment, we choose to use a caching level of 30% for the FDN systems in the next experiments. We believe that this caching level provides a sustainable trade-off between caching size and streaming QoE.

4.4.2 Analyzing the impact of oversubscription. In this experiment, our goal is to study the robustness of the F-FDN platform against increasing workload intensity (aka oversubscription) and compare it against alternative methods. For that purpose, we vary the number of arriving video segments from 3,000 to 4,500 (with increments of 500) within the same time interval and measure the percentage of video segments that miss their deadlines. In this experiment, FDN-based methods cache 30% and the CDN method caches 75% (for practical reasons [55]) of video segments, while Central Cloud stores all the video contents.

Figure 4.3 demonstrates the performance of different methods as the workload

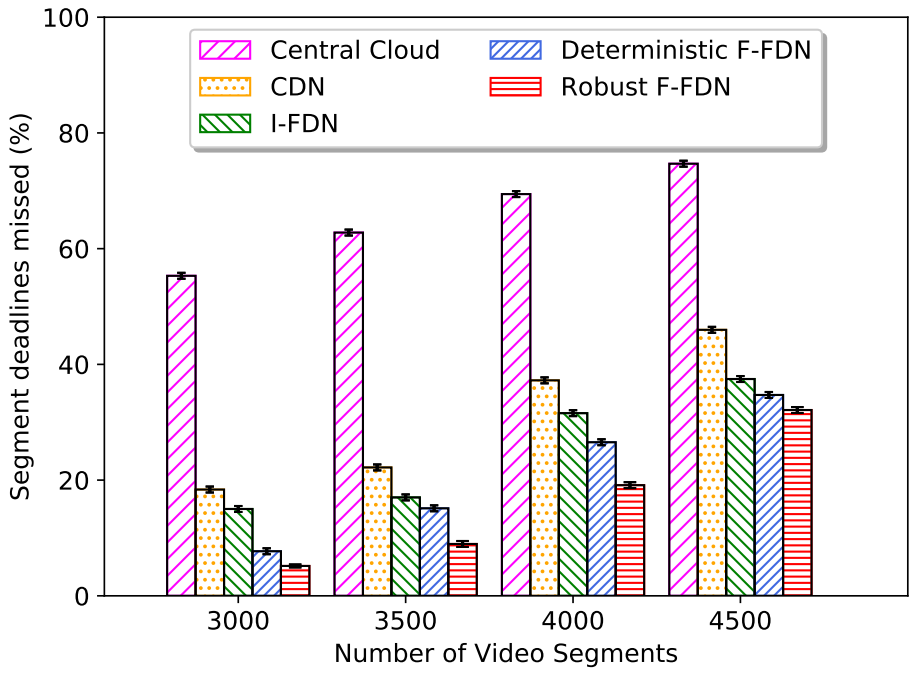
size increases (horizontal axis). We observe that as the number of arriving requests increases, the percentage of segments missing their deadlines increases too. In particular, in comparing the CDN and Central Cloud methods, we see the benefit of a viewer being able to access a CDN server that is much closer to them geographically. Across all workloads the CDN method misses an average of 54% less deadlines than the Central Cloud method. With the presence of on-demand processing in the I-FDN compared to CDN, there is an average of a 17% deadline miss rate improvement. Performance is shown to further increase upon adding the federation of FDNs for streaming, as is present in the Deterministic and Robust F-FDN methods. Compared to the I-FDN, the Deterministic F-FDN has an average of 34% less deadlines missed.

Comparing the performance of the Deterministic F-FDN and the Robust F-FDN methods, we observe a further improvement of deadline miss rate. Across all workloads, the Robust F-FDN performs an average of 28% better than the Deterministic F-FDN. This is due to capturing the stochastic factors (related to communication and computation) present in Robust F-FDN and absent in the Deterministic F-FDN.

4.4.3 Analyzing the impact of network latency. The goal of this experiment is to evaluate the robustness of our methods against viewers' geographical locations. This is particularly important for viewers located in distant areas, where the quality of the edge network commonly fluctuates and is highly uncertain. For that purpose, we study the performance of different methods where the uncertainty in the network latency between the viewer and FDNs and between FDNs is steadily increased.

The result of this experiment is shown in Figure 4.4. The horizontal axis shows

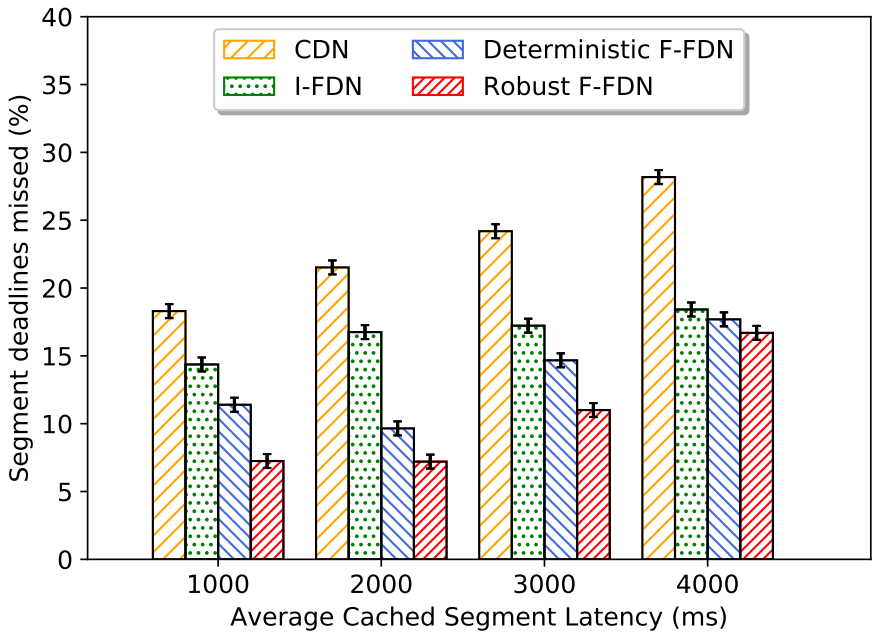
Figure 4.3. Deadline miss rate at increasing workload intensity. FDN-based methods cache 30% of video segments while CDN has 75% of videos cached.



the average latency to receive a cached video segment and the vertical axis shows the percentage of segments that missed their deadlines. We evaluated CDN, I-FDN, Deterministic F-FDN, and Robust F-FDN methods. Because our focus is on the edge network, we do not consider the Central Cloud method. This experiment is conducted with 3,500 segments and 30% of video segments are cached in each FDN, except CDN that caches 75% of all videos.

We observe that all methods result in a higher deadline miss rate as the network latency is increased. However, we observe that CDN deadline miss rate is increased at a greater rate than that of the other methods. When comparing the CDN and I-FDN system at an average network latency of 1,000 ms the deadline miss rate is at a difference of 20.2%. Nonetheless, when the average network latency is increased to

Figure 4.4. Deadline miss rate with increasing latency of the edge network. The experiments are conducted using 3500 video segments. FDN-based methods contain 30% and CDN has 75% cached video contents.



4,000 ms, the difference in deadline misses maintains at 20.4%.

For the CDN method, all segments that are not cached must be streamed from the central cloud. When the latency for streaming from the CDN is not ideal, more segments are streamed from central cloud. Since I-FDN can perform on-demand processing for segments that are not cached, there is less of a reliance on the central cloud. This explains the consistently better performance of I-FDN as the average network latency is increased when compared to the CDN method.

In comparing I-FDN to Deterministic F-FDN, we notice that the difference of deadline miss rate decreases between the two methods as the network latency increases. Particularly, at an 1,000 ms network latency, the Deterministic F-FDN performs 64%

better than I-FDN. However, when the network latency is increased to 4,000 ms, there is only a 7.6% difference in deadline miss rate between the two methods. This decrease in difference of deadline miss rates can be explained by the Deterministic F-FDN choosing to stream segments from neighboring FDNs less, instead, processing those segments in an on-demand manner. The same phenomenon can be observed when comparing I-FDN and Robust F-FDN. At 1,000 ms network latency, the deadline miss rate difference is 97% and at 4,000 ms network latency, the difference is only 15.4%. This observation shows the adaptability that is inherent to our system.

In general, we observe that Robust F-FDN outperforms other methods. However, the difference between Deterministic F-FDN and Robust F-FDN becomes less prominent, when the network latency at the edge is at the highest we tested (4,000 ms). The reason for outperformance is capturing stochasticity in the network latency. However, the reason for similar performance at 4,000 latency is that both methods choose to process on-demand as opposed to relying on a highly uncertain network and fetch segments from neighboring FDNs.

We can conclude that F-FDN platform, in general, can remarkably improve the performance of streaming compared to traditional CDN-based methods. Interestingly, the improvement becomes more significant, as the network becomes more uncertain. Among FDN-based methods, Robust F-FDN can capture the stochasticity that exists in the network to a certain level and further improve the performance.

4.5 Summary

We presented F-FDN, a novel distributed platform for low latency video streaming that enhances the traditional CDN system with fog computing capabilities. The primary goal of F-FDN is to deliver a streaming service with a high QoE, specifically to viewers located in geographically distant areas. Along with the proposal of the platform, we created a method to make intelligent streaming decisions at each FDN, so that the likelihood of having an uninterrupted video streaming experience is maximized. Our experiment results show F-FDN having an average of a 52% improvement in deadline miss rate, when compared to the traditional CDN system that neither processes videos at the edge nor considers the cached video content of neighbors. Adaptive decision making within each FDN provides robustness against network latency fluctuations.

Chapter 5: Conclusion and Future Research Directions

5.1 Discussion

With the prevalence of video streaming and lack of flexible streaming services for viewers, this thesis explored the viability of offering *interactive* video streaming services using cloud systems. This exploration was achieved in two major directions that form the core chapters of this thesis.

The *first* direction of this thesis, discussed in Chapter 3, verifies the feasibility of interactive video streaming. We developed a platform, called CVSE, that provides interactive video streaming by enabling stream providers to develop new services for their viewers. CVSE then accommodates the new service by maintaining certain QoE standards for viewers. In order to maintain QoE performance, a combination of pre-processed partial video caching and on-demand processing is utilized. Also, cloud resources (VMs or containers) are provisioned in an elastic manner to provide sufficient computing power for the requested services. The architecture for CVSE is designed to be modular and extensible. Because of this modular design, CVSE can be tailored or expanded in terms of resource type (*e.g.,* cloud or edge), available services, type of computing resources (*e.g.,* emulation, local threads, containers, and VMs), and billing types (*e.g.,* pay-as-you-watch) to fit the demands of the context in which the platform is utilized.

The *second* direction of this thesis, discussed in Chapter 4, focuses on the possibility of satisfying QoE demands of viewers in interactive video streaming. We considered the fact that the viewers can be scattered globally and the QoE must be

offered regardless of the viewers' location. We developed a platform, F-FDN, that utilizes CVSE within a distributed fog system. This platform was developed in order to provide interactive video streaming with a high QoE, specifically to viewers located in geographically distant areas. We developed methods within F-FDN to make streaming decisions that provide the highest probability of streaming a video with low latency. It is important to note that the current evaluations of F-FDN have been primarily through emulation. As we develop more practical implementations of F-FDN, measuring overhead and deployment costs of the platform will become another area of focus in this research. Finally, through our experiments we were able to observe an improvement in performance in comparison to other existing distributed video streaming systems.

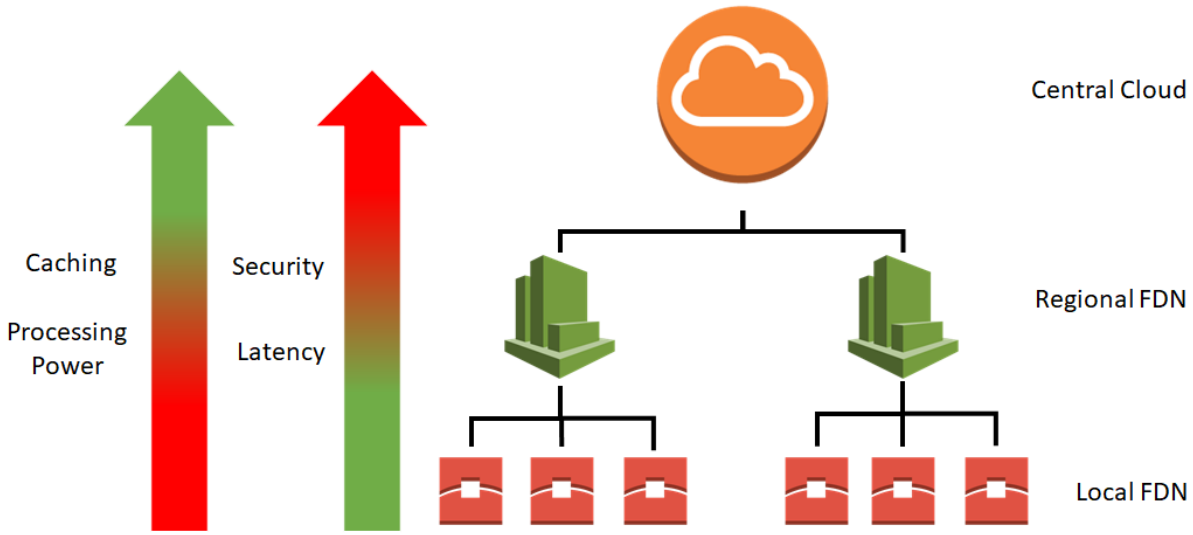
In addition to those areas we explored in this thesis, there are a number of future research avenues that should be explored to further evolve the interactive video streaming. In the next section, we mention a few of these avenues.

5.2 Future Research Directions in Interactive Video Streaming

5.2.1 Heterogeneous container types. As currently implemented, all the container-based resources that constitute the compute engine of CVSE are considered to be a universal type (*i.e.,* can process tasks of any request type). However, there are some performance benefits in specializing the containers for specific task types (*i.e.,* make each of them specialized to only certain types of tasks). For instance, we can have containers with hardware specifications that matches better to changing video codecs and another container type that matches better with object detection in videos.

To incorporate heterogeneous container types into CVSE, there are certain

Figure 5.1. Structure of multi-tiered F-FDN platform



management mechanisms and policies to be explored. For example, container types for tasks that are very common should be made available at all times, while containers for uncommon tasks are allocated when those requests arrive. What types of tasks should be processed on specialized containers versus general-use containers? How many of the containers of each type should the resource provisioner instantiate from, in both reactive to workload and predictive to expected workloads?

This change even has some implications on software engineering methodologies used for micro-service based applications. In fact, in such software systems some services are permanently running (warm services) while others have intermittent nature (cold services). Function-as-a-service can be used particularly to accommodate cold services.

Such research questions that are needed to be addressed in future.

5.2.2 Multi-tier f-fdn architecture. To further enhance F-FDN, a multi-tiered (*i.e.,* hierarchical) structure of FDNs will be explored. The current version of F-FDN has two tiers—Central Cloud and FDN. A multi-tier architecture of F-FDN, however, consists of a Central Cloud and a number of increasingly granular FDN. In Figure 5.1 a 3-tier system is depicted that consists of a Central Cloud, a number of Regional FDN, and a number of Local FDN, with Local FDN being the most granular pieces of the platform.

As the FDN become more granular (*e.g.,* more geographically spread) the associated latency for video streaming decreases since those FDN are physically closer to viewers. With a higher granularity, the security of those FDN is higher, conversely, the computational processing power and video segment caching capabilities decrease.

For example, the Central Cloud, which has the greatest amount of resources within our platform, has 100% of videos cached, the Regional F-FDN has 70% of videos cached, and the Local FDN have 30% of videos cached. If a video that is being requested does not exist in the Local FDN, the cache of the Regional F-FDN will be considered before the cache of the Central Cloud is considered. The multi-tier architecture of F-FDN allows for more tailored solutions specifically in the context of a very large, geographically spread population of viewers that a streaming service provider is servicing.

Another area of exploration within the multi-tier F-FDN architecture, is the utilization of *Distributed Hash Tables* (DHT) in the Regional F-FDN and Local FDN tiers. This could potentially replace the function of the Metadata Manager located in

the Central Cloud. Through the use of DHT, the FDN would keep an updated knowledge of neighboring FDN metadata without the need of constant referrals and updates to the Central Cloud. This communication would occur directly between neighboring FDN. The use of DHT could provide a much more efficient overhead for this feature of the F-FDN platform.

Another issue that can be more specifically evaluated with a multi-tier architecture is the concerns of Net Neutrality and the prevalence of "fast tracks" in Internet bandwidth that exist solely for larger companies. We will be able to investigate if the consideration of retrieving cached video segments from neighboring FDN is either a benefit or detriment when there is biased sharing of the Internet bandwidth.

5.2.3 On-demand processing of 360 degree videos. Another future research of CVSE involves developing the methods on how to best deliver 360 degree video streams. 360 degree videos are comprised of multiple video sections that are stitched together. This future work focuses on processing these sections of video in an on-demand manner at different resolutions, rather than streaming the full resolution of the 360 degree view.

GOPs for the default view of 360 degree videos and its peripheral area would be pre-processed at high resolutions, while unobserved sections of the 360 degree videos are streamed at lower resolution to save bandwidth. However, viewers are still able to change their viewing angle without interrupting the video. As the viewer pans around, the system can process and stream parts of the requested view at higher resolutions on-demand.

This type of video benefits greatly from lowered streaming latency. In general, if the latency is high, viewers will need a higher amount of buffer to cover processing and streaming delay. In the case of 360 degree videos, some parts of the buffer may end up not being viewed. Low latency streaming reduces the amount of buffer needed, thus reducing the amount of wasted processing and transferring bandwidth. Therefore, extending CVSE and F-FDN platform to leverage on-demand processing for 360 degree videos can reduce cost and improve QoE considerably.

5.2.4 Dynamic billing. In this work, the way in which viewers are charged when using CVSE services is further explored. While most video streaming service companies are using a monthly subscription as their payment model for viewers, we believe there is a desire for a new payment policy with the pay-as-you-watch model. The CVSE platform would collect resource usage information that enables the pay-as-you-watch payment model. This usage information is then calculated in order to determine the cost charged to the viewer.

There are numerous factors to be considered when calculating cost in the pay-as-you-watch model. Here are a few examples: (1) *What is the availability of the video in the viewer's requested format?*, (2) *What is the current oversubscription status of the platform?*, (3) *Is the video currently cached?*, and (4) *What is the level of processing required for the video stream request?* are just a few of the many possible factors to consider while calculating cost.

The pay-as-you-watch model mostly benefits viewers who want to stream certain contents from a stream service provider, but do not watch said videos often. We believe

the pay-as-you-watch model can become a legitimate alternative to the commonly used monthly subscription model in certain cases for viewers.

Bibliography

[1] "How Netflix Works With ISPs Around the Globe to Deliver a Great Viewing Experience," https://media.netflix.com/en/company-blog/how-netflix-works-with-isps-around-the-globe-to-deliver-a-great-viewing-experience", accessed on Mar. 31, 2019.

[2] X. Li, "High performance on-demand video transcoding using cloud services," PhD dissertation, University of Louisiana at Lafayette, 2016.

[3] "Cisco Visual Networking Index: Forecast and Methodology, 2016-2021," www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html#_Toc484813971", accessed on Aug. 07, 2018.

[4] J. Wu, C. Yuen, B. Cheng, M. Wang, and J. Chen, "Streaming high-quality mobile video with multipath tcp in heterogeneous wireless networks," *IEEE Transactions on Mobile Computing*, vol. 15, no. 9, pp. 2345–2361, 2016.

[5] "CDN Performance," www.cloudflare.com/learning/cdn/performance/", accessed on Aug. 07, 2018.

[6] "Open Connect Overview," https://openconnect.netflix.com/Open-Connect-Overview.pdf", accessed on Aug. 06, 2018.

[7] A.-T. Nguyen, O. Fourmaux, and C. Deleuze, "Exploring youtubes cdn heterogeneity," in *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*. Springer, 2018, pp. 158–166.

[8] M. L. Wayne, "Netflix, amazon, and branded television content in subscription video on-demand portals," *Media, Culture & Society*, vol. 40, no. 5, pp. 725–741, 2018.

[9] "Can Netflix Rein in Skyrocketing Costs?" https://www.fool.com/investing/2016/06/22/can-netflix-rein-in-skyrocketing-costs.aspx".

[10] W. Hoiles, O. N. Gharehshiran, V. Krishnamurthy, and H. Zhang, "Adaptive caching in the youtube content distribution network: A revealed preference game-theoretic learning approach," *IEEE Transactions on Cognitive Communications and Networking (TCCN)*, vol. 1, no. 1, pp. 71–85, Mar. 2015.

[11] Y. Lin and H. Shen, "Cloudfog: Leveraging fog to extend cloud gaming for thin-client mmog with high quality of service," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 28, no. 2, pp. 431–445, Feb. 2017.

[12] X. Li, M. Amini Salehi, M. Bayoumi, and R. Buyya, "CVSS: A Cost-Efficient and QoS-Aware Video Streaming Using Cloud Services," in *Proceedings of the 16th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, ser. CCGrid '16, May 2016.

[13] X. Li, M. A. Salehi, M. Bayoumi, N.-F. Tzeng, and R. Buyya, "Cost-Efficient and Robust On-Demand Video Stream Transcoding Using Heterogeneous Cloud Services," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 29, no. 3, pp. 556–571, Mar. 2018.

[14] X. Li, M. Amini Salehi, Y. Joshi, M. Darwich, L. Brad, and M. Bayoumi, "Performance Analysis and Modelling of Video Stream Transcoding Using Heterogeneous Cloud Services," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Sep. 2018, doi: url10.1109/TPDS.2018.2870651.

[15] "Content Delivery Network (CDN) Caching," aws.amazon.com/caching/cdn/", accessed on Aug. 06, 2018.

[16] S. Khan, R. Schollmeier, and E. Steinbach, "A performance comparison of multiple description video streaming in peer-to-peer and content delivery networks," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, ser. ICME '04, vol. 1, Jun. 2004, pp. 503–506.

[17] M. Darwich, M. A. Salehi, E. Beyazit, and M. Bayoumi, "Cost-efficient cloud-based video streaming through measuring hotness," *The Computer Journal*, Jun. 2018.

[18] "FFmpeg," https://ffmpeg.org/".

[19] F. Jokhio, T. Deneke, S. Lafond, and J. Lilius, "Analysis of video segmentation for spatial resolution reduction video transcoding," in *Proceedings of the 19th International Symposium on Intelligent Signal Processing and Communications Systems*, Dec. 2011, pp. 1–6.

[20] M. Hosseini, M. A. Salehi, and R. Gottumukkala, "Enabling Interactive Video Stream Prioritization for Public Safety Monitoring through Effective Batch Scheduling," in *Proceedings of the 19th IEEE International Conference on High Performance Computing and Communications*, ser. HPCC '17, Dec. 2017.

[21] M. Darwich, E. Beyazit, M. A. Salehi, and M. Bayoumi, "Cost efficient repository management for cloud-based on-demand video streaming," in *Proceedings of the 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, Apr. 2017, pp. 39–44.

[22] L. C. Miranda, R. L. Santos, and A. H. Laender, "Characterizing video access patterns in mainstream media portals," in *Proceedings of the 22nd Conference on World Wide Web*, May 2013, pp. 1085–1092.

[23] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, Oct. 2007, pp. 15–28.

[24] X. Li, M. A. Salehi, and M. Bayoumi, "VLSC: Video Live Streaming Using Cloud Services," in *Proceedings of the 6th IEEE International Conference on Big Data and Cloud Computing Conference*, ser. BDCloud '16, Oct. 2016, pp. 595–600.

[25] X. Li, M. A. Salehi, M. Bayoumi, and R. Buyya, "CVSS: A Cost-Efficient and QoS-Aware Video Streaming Using Cloud Services," in *Proceedings of the 16th IEEE/ACM International Conference on Cluster Cloud and Grid Computing*, ser. CCGrid '16, May 2016.

[26] F. Jokhio, A. Ashraf, S. Lafond, and J. Lilius, "A computation and storage trade-off strategy for cost-efficient video transcoding in the cloud," in *Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications*, ser. SEAA '13, Sep. 2013, pp. 365–372.

[27] H. Zhao, Q. Zheng, W. Zhang, B. Du, and Y. Chen, "A version-aware computation and storage trade-off strategy for multi-version VOD systems in the cloud," in *Proceedings of the 20th IEEE Symposium on Computers and Communication*, ser. ISCC '15, July 2015, pp. 943–948.

[28] B. Khemka, R. Friese, L. D. Briceno, A. A. Maciejewski, G. A. Koenig, G. Okonski, M. M. Hilton, R. Rambharos, S. Poole, and C. Groer, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," *IEEE Transactions on Computers*, vol. 64, no. 8, pp. 2394–2407, Aug. 2015.

[29] A. Ashraf, F. Jokhio, T. Deneke, S. Lafond, I. Porres, and J. Lilius, "Stream-based admission control and scheduling for video transcoding in cloud computing," in *Proceedings of the 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, ser. CCGrid '13, May 2013, pp. 482–489.

[30] S. Lin, X. Zhang, Q. Yu, H. Qi, and S. Ma, "Parallelizing video transcoding with load balancing on cloud computing," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, ser. ISCAS '13, May 2013, pp. 2864–2867.

[31] M. Kim, Y. Cui, S. Han, and H. Lee, "Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, no. 2, pp. 213–224, Mar. 2013.

[32] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, Mar. 2014.

[33] X. Li, M. A. Salehi, and M. Bayoumi, "High Perform On-Demand Video Transcoding Using Cloud Services," in *Proceedings of the 16th IEEE/ACM International Conference on Cluster Cloud and Grid Computing (Ph.D. Symposium)*, ser. CCGrid '16, May 2016.

[34] Q. He, J. Liu, C. Wang, and B. Li, "Coping with heterogeneous video contributors and viewers in crowdsourced live streaming: A cloud-based approach," *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 916–928, May 2016.

[35] Q. He, C. Zhang, X. Ma, and J. Liu, "Fog-based transcoding for crowdsourced video livecast," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 28–33, Apr. 2017.

[36] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.

[37] Y. Lin and H. Shen, "Leveraging fog to extend cloud gaming for thin-client mmog with high quality of experience," in *Proceedings of the 35th IEEE International Conference on Distributed Computing Systems*, ser. ICDCS '15, Jun. 2015, pp. 734–735.

[38] M. Ryden, K. Oh, A. Chandra, and J. Weissman, "Nebula: Distributed edge cloud for data-intensive computing," in *Proceedings of the International Conference on Collaboration Technologies and Systems*, ser. CTS '14, May 2014, pp. 491–492.

[39] L. L. Provensi, A. Singh, F. Eliassen, and R. Vitenberg, "Maelstream: Self-organizing media streaming for many-to-many interaction," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, Jan. 2018.

[40] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network System Management*, vol. 23, no. 3, pp. 567–619, Jul 2015.

[41] M. A. Salehi, P. R. Krishna, K. S. Deepak, and R. Buyya, "Preemption-aware energy management in virtualized data centers," in *Proceedings of the 5th IEEE International Conference on Cloud Computing*, ser. CLOUD '12, Jul., pp. 844–851.

[42] S. Singh and I. Chana, "A survey on resource scheduling in cloud computing: Issues and challenges," *Journal of Grid Computing*, vol. 14, no. 2, pp. 217–264, June 2016.

[43] E. Delen, J. Liew, and V. Willson, "Effects of interactivity and instructional scaffolding on learning: Self-regulation in online video-based environments," *Computers & Education*, vol. 78, pp. 312–320, Sep. 2014.

[44] N. Chen, Y. Chen, Y. You, H. Ling, P. Liang, and R. Zimmermann, "Dynamic urban surveillance video stream processing using fog computing," in *In Proceedings*

of the 2nd IEEE International Conference on Multimedia Big Data, Apr. 2016, pp. 105–112.

[45] C. V. N. Index, "Forecast and methodology, 2014-2019," 2015.

[46] X. Cheng, J. Liu, and C. Dale, "Understanding the characteristics of internet short video sharing: A YouTube-based measurement study," *IEEE Transactions on Multimedia*, vol. 15, no. 5, pp. 1184–1194, Aug. 2013.

[47] Y. Jin, Y. Wen, and C. Westphal, "Optimal transcoding and caching for adaptive streaming in media cloud: an analytical approach," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 1914–1925, Dec 2015.

[48] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," in *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, ser. MMSys '11, 2011, pp. 145–156.

[49] J. He, Y. Wen, J. Huang, and D. Wu, "On the Cost–QoE Tradeoff for Cloud-Based Video Streaming Under Amazon EC2's Pricing Models," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 24, no. 4, pp. 669–680, Apr. 2014.

[50] M. A. Salehi, J. Smith, A. A. Maciejewski, H. J. Siegel, E. K. P. Chong, J. Apodaca, L. D. Briceno, T. Renner, V. Shestak, J. Ladd, A. Sutton, D. Janovy, S. Govindasamy, A. Alqudah, R. Dewri, and P. Prakash, "Stochastic-based robust dynamic resource allocation for independent tasks in a heterogeneous computing system," *in Journal of Parallel and Distributed Computing (JPDC)*, vol. 97, no. C, Nov. 2016.

[51] F. Song, C. Zhu, Y. Liu, Y. Zhou, and Y. Liu, "A new gop level bit allocation method for hevc rate control," in *Proceedings of 12th IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, ser. BSMB '17, June 2017, pp. 1–4.

[52] V. Stocker, G. Smaragdakis, W. Lehr, and S. Bauer, "The growing complexity of content delivery networks: Challenges and implications for the internet ecosystem," *Telecommunications Policy*, vol. 41, no. 10, pp. 1003–1016, Nov. 2017.

[53] A. Shojaeifard, K.-K. Wong, W. Yu, G. Zheng, and J. Tang, "Full-duplex cloud radio access network: Stochastic design and analysis," *IEEE Transactions on Wireless Communications*, vol. 17, no. 11, pp. 7190–7207, Nov. 2018.

[54] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results," in *The International Conference for High Performance Computing, Networking, Storage, and Analysis*, ser. SC '15, Nov. 2015, p. 73.

[55] S. Dernbach, N. Taft, J. Kurose, U. Weinsberg, C. Diot, and A. Ashkan, "Cache content-selection policies for streaming video services," in *Proceedings of the 35th Annual IEEE International Conference on Computer Communications*, ser. INFOCOM '16, Apr. 2016, pp. 1–9.

[56] C. Denninnart, M. Amini Salehi, A. N. Toosi, and X. Li, "Leveraging computational reuse for cost-and qos-efficient task scheduling in clouds," in *Proceedings of the International Conference on Service-Oriented Computing*, ser. ICSOC '18, Nov. 2018, pp. 828–836.

[57] "CDN Network Test," cloudharmony.com/speedtest-for-cdn", accessed on Aug. 07, 2018.

Veillon, Vaughan. Bachelor of Science, University of Louisiana at Lafayette, Spring 2017; Master of Science, University of Louisiana at Lafayette, Spring 2019
Major: Computer Science
Title of Thesis: Fog Computing for Low Latency, Interactive Video Streaming
Thesis Director: Mohsen Amini Salehi
Pages in Thesis: 81; Words in Abstract: 196

Abstract

Video streaming is growing in popularity and has become the most bandwidth-consuming Internet service. As such, robust streaming in terms of low latency and uninterrupted streaming experience, particularly for viewers in distant areas, has become a challenge. The common practice to reduce latency is to pre-process multiple versions of each video and use Content Delivery Networks (CDN) to cache videos that are popular in a geographical area. However, with the fast-growing video repository sizes, caching video contents in multiple versions on each CDN is becoming inefficient. Accordingly, in this paper, we propose the architecture for Fog Delivery Networks (FDN) and provide methods to federate them (called F-FDN) to reduce video streaming latency. In addition to caching, FDNs have the ability to process videos in an on-demand manner. F-FDN leverages cached contents on the neighboring FDNs to further reduce latency. In particular, F-FDN is equipped with methods that aim at reducing latency through probabilistically evaluating the cost benefit of fetching video segments either from neighboring FDNs or by processing them. Experimental results against alternative streaming methods show that both on-demand processing and leveraging cached video segments on neighboring FDNs can remarkably reduce streaming latency (on average 52%).

Biographical Sketch

Vaughan Veillon received his Bachelor of Science in the field of computer science in the spring of 2017 from the University of Louisiana at Lafayette. The son of Bernard Veillon and Monique Lake, he began pursuing a master's degree in computer science at the University of Louisiana at Lafayette in the fall of 2017. He completed the requirements for the degree in the spring of 2019.