
Department of Information Engineering and Computer Science

Bachelor's Degree in
Information and Business Organisation Engineering

Final dissertation

# A service migration framework for live video streaming using SDN and Virtualization

Supervisor

Prof. Fabrizio Granelli

Student

Davide Parpinello

Academic Year 2020/2021

# Acknowledgments

*Before proceeding with the dissertation, it is my duty to dedicate this space to the people who have helped me achieve the same with their tireless support.*

*In the first place, heartfelt thanks go to my supervisor Professor Fabrizio Granelli for his infinite availability to follow my work and timeliness for my every request. Thanks for his advice, his knowledge conveyed, and every valuable material for my research.*

*I am infinitely grateful to my parents that have always supported me in the study course, sharing every decision I made and bearing every tough time.*

*A huge thank you goes to my colleagues Alessio, Alberto, Mattia, Davide, Giorgia, Matthias, and Chiara, with which I shared my whole university course. Thanks for their essential help and for sharing the joy of every achievement.*

*And last but not least, thanks to my friends, especially Andrea, Jacopo, and Matteo, for always being there also in this final stage and for every carelessness moment.*

*Thanks so much to everyone. Without you, I couldn't have done it.*

*Finally, I dedicate this little finish line to myself, which can be the beginning of a long and brilliant professional career.*

# Index

# Abstract

Video streaming is nowadays one of the most used technology to create information and engage users. Thanks to the increased processing power, the increasing bandwidth, and new tools like social media or streaming platforms, everyone now engages using videos.

This dissertation focuses on this technology, its advantages, disadvantages, and the problems that the increasing traffic is giving to the network, especially in the connections near the final user. A simplified service migration framework will be developed and tested using a testbed that emulates a Software Defined Network compared to traditional networks. This service migration system will be an exciting starting point in realizing a video streaming system of the future, providing a great experience to the user and reducing network traffic and server load.

In the first short introductive chapter, we will analyze the diffusion of video streaming with some statistics and graphs about the last few years. It will also be presented the spread of services like YouTube and Netflix, given their wide usage. Concluding the chapter, the issue about increasing network traffic will be raised, with the benefit of using network virtualization as a solution.

The second chapter will be a more theoric explanation about the technologies that this research use. The most used video streaming protocols will be discussed and compared, with their advantages and disadvantages. Then we will introduce the concept of Network Function Virtualization (NFV) with the benefits it gives to video streaming and modern networks. We will discuss some of the most used NFV frameworks. Accompanied with that, Software Defined Networks will be explained, given their valuable role in NFV. After this explanation, the network orchestration model will be introduced with its advantages for this research. Ending the chapter, a short introduction to the leading technologies we will use in the experimental part of this dissertation: Docker, containers, and the network emulator ComNetsEmu.

The third chapter will be the central part of this dissertation, explaining the experimental work of this research and the steps taken to develop the final testbed. Every step will be accompanied by a description of the tools and software used, graphical explanations of the system, code snippets, and the problems faced during the experimentation. We will conclude this chapter with a description of the final result and its proper functioning.

This dissertation will be concluded with two small chapters, showing in the first time the experimental results, a comparison with other solutions and significant issues, then possible future developments with the scalability of this system and other optimizations.

# 1 Introduction to video streaming: advantages and issues

The Internet has seen radical changes during its evolution, moving from simple text accompanied by images and styles, then interactive pages. The content is updated in real-time, even without requiring user action. This evolution has paved the way for inserting multimedia inside a web page: in the first time, audio and recorded videos, then live videos.

Through the evolution of technology, the increased bandwidth available globally in the network, and the ever-growing processing power of user devices (computers, laptops, tablets, and smartphones) and servers, the birth of video-oriented services has been inevitable. The most iconic one, YouTube, founded in 2005 and bought by Google in 2006, offers free hosting for an infinite range of videos, from music videos to vlogs. Like YouTube, many other similar services will become important in the following years, as this dissertation will show later in this chapter.

## 1.1 The growth of the video streaming industry and the COVID-19 pandemic

The video streaming industry has increased a lot in the last few years. Videos are helpful for businesses that want to engage users to convince them to buy their products. 60% of companies use video as a marketing tool: sales are the most critical sector, 32% of the content is dedicated to that.

Marketers have now understood the power of video: 61% of them see video as a "very important or extremely important" part of their marketing strategy because they say video has a better return on investment than static imagery.[4]

Apart from businesses, videos are helpful for online newspapers that are moving from simple articles to enrich with media content. Because of this, video has stunning numbers nowadays: people watch an average of 16 hours of online video per week, which is a 52% increase in the last two years. Also, social video generates 1200% more shares than text and image content combined, and viewers retain 95% of a message when they watch it in a video compared to 10% when reading it in text.[35]

Video streaming is ideal for reaching users with little effort. When used for live events, it shows at its best: anybody can connect and watch his favorite concert or attend a conference from everywhere in the world. Live streaming has more incredible statistics than simple videos: live video grew by 93%, with an average viewing time of 26.4 minutes per session [30]. The live music event with the highest live viewership was 2019 Coachella, at about 82.9 million live views [33].

In addition, video streaming has been essential to bring entertainment into everybody's homes and connect people together with video calls and webinars.

During the COVID-19 pandemic, the usefulness of video streaming and live streaming has been noted more than ever: people were required to attend job meetings or school and university lessons from their computers using video calls services. There was a 500% increase in search impressions for web and video conferencing software in the first four months of the pandemic [31], and Comcast said that the traffic for video calls has increased by more than 210% during that period.

As for entertainment, spending more time at home, people used many services like Netflix and YouTube, watching movies or live events. This pushed up the internet traffic for streaming by at least 12% [13].

## 1.2 On-demand and live OTT video services

As stated previously, YouTube was the first platform that offered free video hosting. Since its birth has done an incredible growth: every day worldwide, its users watch more than one billion hours of

videos [38], and more than 500 hours of content are uploaded on YouTube every minute [32]. These numbers indicate the role that YouTube nowadays has in our society: everyone searches for a video on the platform for a cooking recipe, a school subject, or simply watching music videos or entertainment videos.

Later than YouTube, many other platforms took place on the Internet and became just as famous: especially for gamers, Twitch has been an important player to allow everyone to live-stream his games directly from their room. As of February 2020, it has about 15 million daily active users, with 1.4 million average concurrent users, and thanks to its wide diffusion, it has become the only work for many creators.

But the real breakthrough was accomplished by Netflix, an over-the-top content platform that is present almost in everyone's home. It offers a rich library of more than 3600 movies and 1800 television series, with a good part of in-house produced titles. As of April 2021, Netflix has 208 million subscribers, and it's available almost worldwide, with total revenue in 2020 of $ 25 billion. The most streamed series in the US over 2020 was The Office, with 57.1 billion minutes [6].

These platforms, YouTube, Netflix, and their competitors, like Amazon Prime Video or Hulu, have certainly changed radically almost everyone's habits and are also causing trouble for the global network and the ISP operators.

## 1.3 The problem with the global network capacity

As we've seen later, videos are a powerful marketing tool: people like them, and they create revenues. But this wide-spreading is also creating problems for the network's capacity: it's estimated that by 2022, 82% of the global internet traffic will come from video streaming and downloads [8]. Internet-enabled HD televisions are becoming essentials in houses, but they could generate as much internet traffic as an entire household with only three hours of content.

Technology is also evolving, introducing Ultra High Definition video streaming, which requires at least 15 Mbps of traffic [19]. If we count the significant number of houses that will start to use that, we obtain a massive internet traffic consumption, estimated to 240 exabytes of data per month by 2022.

As said previously, during the COVID-19 pandemic, the usage of video streaming services increased a lot: as a result, to reduce the strain on ISPs, Netflix had to reduce the video quality of its service in Europe, lowering the data consumption by 25% [3]. It's now clear how video streaming services are becoming a massive problem for the internet and ISPs.

## 1.4 From CDNs to Virtualized CDNs

One solution that almost every ISP is deploying for several years now is Content Delivery Networks (CDN). CDNs are distributed network of servers that provides high availability and performance to users: they are deployed in multiple locations and tend to be placed nearest to the user as possible, as edge servers, to reduce bandwidth costs, links strain, and load times [27]. Even Netflix has its CDN, called Open Connect, intending to provide the highest quality experience possible: this is achieved by placing their servers inside the ISPs' networks with a substantial amount of Netflix traffic.

The use of CDNs had indeed improved the delivery of high-quality content reducing at the same time the network strain. Still, with the increase in the audience and the traffic required, there are some significant challenges that operators are looking to overcome: capacity planning, agility, cost, and quality [1].

The first one, capacity planning, refers to the difficulty for operators to anticipate the traffic for an event: over-provisioning leads to wasted resources, and under-provisioning leads to poor quality or service outages. The goal here is to remove the guesswork. About agility, the amount of time needed to build CDN capacity could be a lot, so in case of events that trigger unanticipated and instantly high viewership, there isn't enough time to activate additional CDN capacity. Regarding the cost challenge, as said previously, it can be wasteful to overspend in capacity and harmful to under-spend. Also, nowadays, it is a habit to have purpose-built solutions designed specifically for one type of service (e.g., VoD): this is costly to ISPs and is not the answer given the forecasted growth of streaming

Figure 1.1: Topology of a Content Delivery Network

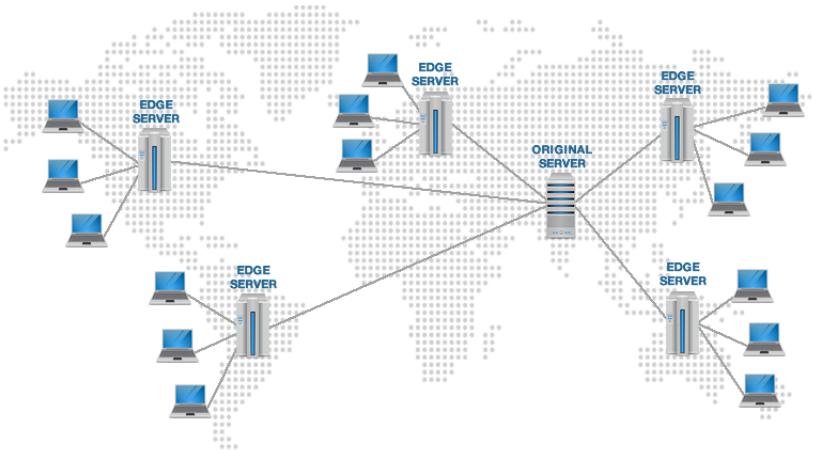
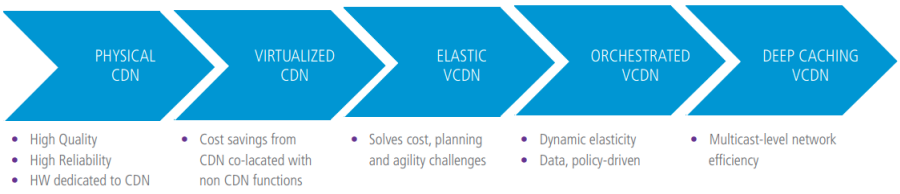



Figure 1.2: The evolution of virtualized CDN to maximum scale unicast video delivery




services. The challenge is to develop a single, flexible, highly scalable multi-purpose platform. The last challenge is about quality: traditional CDNs are deployed in centralized data centers, but it's more efficient to deliver from CDN located in PoPs deeper in the network for high-audience events. The challenge will be for operators to find a way to deploy CDN resources "closer to the eyeballs" cost-effectively and in more centrally located facilities.

The ultimate solution will be an orchestrated elastic vCDN, where CDNs become a Virtual Network Function to overcome these challenges. This ultimate solution is and will be evolutionary progress, as shown in Figure 1.2. As stated by Akamai and Herbaut, this solution will be the optimal choice for delivering high-quality video services, bringing equal benefits to the content providers (e.g., Netflix) and network operators, reducing infrastructure costs, as a Win-win collaboration [17].

With this vCDN evolution, it will be possible to deploy the network capacity on-demand, even deeper towards the network edge, to meet the needs of peak events. This will be an ability to scale out and in the resources needed to live streaming delivery in a more granular ad efficient way, reducing costs and the time required to do so.

Akamai has already established leadership with a virtualized version of its Aura Licensed CDN deployed and managed by network operators to deliver their content.

This dissertation will later introduce the theory and the technologies behind this notion, trying to develop a simplified framework of vCDN.

# 2 Theoretical background

This chapter will be a theoretical background for the experimental work described in the following chapter.

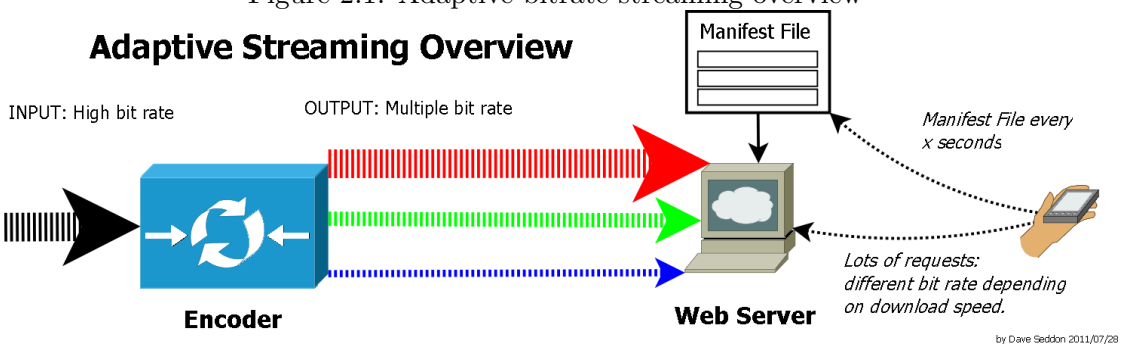
At first, there will be an introduction to internet video streaming protocols, with a short explanation of the most used one and their comparison. Then we'll introduce the ideas of Network Function Virtualization (NFV) and its complementary technology that is Software Defined Networks (SDN). Putting them together, then the concept of network orchestration will be described. It concludes a brief analysis of Docker with its container technology and ComNetsEmu, the testbed used in the experimental section.

## 2.1 Introduction and comparison of video streaming protocols

As seen in the introduction, video streaming is nowadays dominating the internet, with about 80% of the traffic dominated in 2019. This growth in video streaming demand gives birth to several challenges to provide an excellent Quality of Experience (QoE) to users, who usually have a limited network capacity and operate with mobile devices under mobile networks.

The current standard solution to this problem is Adaptive BitRate (ABR) streaming, which enables the rate adaptation during a streaming session by transcoding source in different bitrates, allowing the client to dynamically select the bitrate according to its current network conditions [15].

Figure 2.1: Adaptive bitrate streaming overview




Regardless of how the bitrate in a streaming session is managed, a key parameter is the streaming protocol, which defines how the client and server exchanges data. The choice of the streaming protocol is essential to ensure excellent QoE to the user, avoiding freezings and delay, and ensuring compatibility with endpoint devices. Given that the most frequent way to use the internet is a browser that surf pages under the HTTP protocol, the most common video streaming protocols are the Adaptive HTTP-based ones. We also have Traditional Streaming Protocols (such as RTMP and RTSP) and new emerging technologies (WebRTC, SRT) [36].

### 2.1.1 Adaptive HTTP-Based streaming protocols

HTTP-Based streams should not be considered actual streams but rather progressive downloads of little videos, called chunks, sent using a traditional web server. The most common protocols are Apple HLS, Low-Latency HLS, and MPEG-DASH.

#### Apple HLS

Apple is a major player in internet-connected devices, so its protocol is the most important one. It supports ABR and is based on HTTP, so it will playback on the majority of devices. Initially, only iOS devices supported it, but now almost everyone can play Apple HLS, ensuring accessibility to a

large audience. It has a latency that can vary from 6 to 30 seconds, and one drawback is that it prioritizes quality over latency [36].

#### Low-latency HLS

LL-HLS is the streaming standard for under-three-second streams, offering the same advantages of simplicity and quality as HLS but offering a significant improvement in latency. It's not compatible with almost every device as Apple HLS, being still an emerging specification, but having backward compatibility, every player can fall back to standard HLS [36].

#### MPEG-DASH

MPEG-DASH (Dynamic Adaptive Streaming over HTTP) is an industry-standard alternative to Apple HLS, provided by the MPEG group, and it's an open-source option. It's codec-agnostic, has the same latency as Apple HLS, but it's not supported by Apple devices [36].

### 2.1.2 Traditional streaming protocols

Traditional streaming protocols like RTMP and RTSP support low-latency streams, but not being natively supported by most endpoints, they're used for streaming to a small audience from a dedicated server. These protocols achieve such speed by transmitting the data using a firehose approach rather than requiring local download or caching. Giving their little compatibility with devices, they aren't optimized for great viewing experiences at scale. Many broadcasters choose to transport live streams to the media server using a stateful protocol like RTMP. From there, they can transcode it into an HTTP-based technology for multi-device delivery [36].

RTMP was designed by Adobe, and it uses Flash Player, but open standards and ABR streaming edged RTMP out. Despite the end of life of Flash, RTMP continues to be used by content producers. RTMP has many variants like encrypted ones or based on UDP.

RTSP is an old-school technology used for video contribution, it's not compatible out of the box for most devices but it's still a standard in many surveillance and CCTV systems.

### 2.1.3 Emerging technologies

SRT (Secure Reliable Transport) and WebRTC are emerging technologies in the video streaming industry based on UDP. UDP's advantage is that it is faster than TCP, not requiring a handshake, but it's riskier not having guaranteed transfers. These technologies promise to change the landscape.

#### Secure Reliable Transport (SRT)

SRT is an alternative to proprietary transport technologies, providing reliable streams regardless of network quality. It's a direct competitor for RTMP as a last-mile solution, but it's still being adopted as encoders, decoders, and players add support.

This protocol was designed to solve video distribution challenges over the internet, recover lost packets and preserve time behavior, and quickly become a major player in the industry. SRT is codec-agnostic, and it's being developed by leaders in video streaming like Wowza and Haivision.

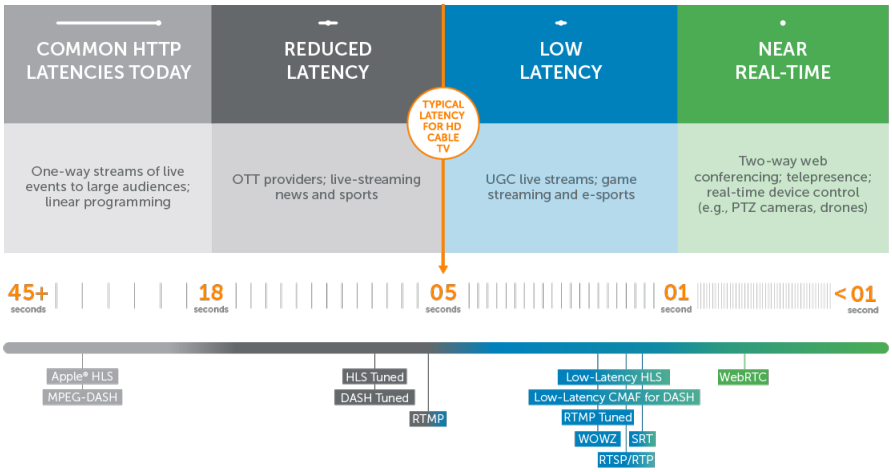
#### WebRTC

WebRTC is the speediest technology available, delivering near zero-latency audio and video from any browser. It was initially designed for chat-based applications, but it's taking place in many other use cases. The drawback of this protocol is the scalability that it's still a challenge.

### 2.1.4 Comparison between protocols

Choosing the proper video streaming protocol is difficult because each differs in specific areas like scalability, latency, and quality. RTMP and SRT are the right choices for the first-mile contribution, while both DASH and HLS are the industry standards for user playback. Figure 2.2 is a comparison between protocols giving their latency.

Figure 2.2: Comparison of streaming protocols by latency




## 2.2 Network Function Virtualization and Software Defined Networks

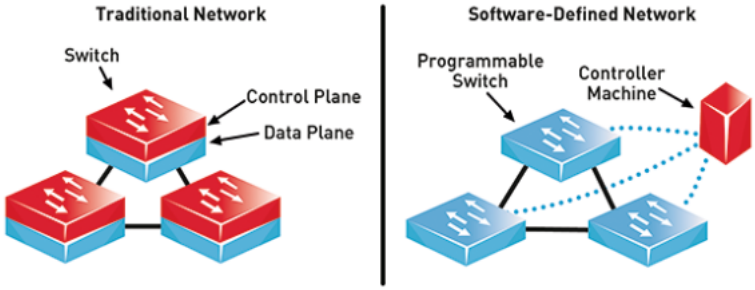
The Network Function Virtualization (NFV) and Software Defined Networks (SDN) are the most important paradigms that were developed to implement software-based virtual networks [20]. These technologies become necessary because of the missing evolution of IP-based networks that don't support new approaches like multipath communication or network slicing.

### 2.2.1 Software-defined Networking

The idea of SDN was initially suggested by the Ethane project [7]: using a centralized controller, it was possible to manage all the switches in the network, on the contrary of the classic Internet routing where it was used a decentralized approach.

This idea also started the development of OpenFlow: a standardized protocol for the messages between the SDN Controller and the network devices like routers and switches. According to the Open Networking Foundation, this idea is one of the key principles of SDN [26]: centralized management that has an overview of the network, with a directly programmable network control decoupling the data plane (where the user traffic flows) and the control plane (for the service traffic) [16]. A third plane, called the application plane, is also expected: it communicates with the control plane using APIs and telling its need to the network. Under these conditions, the traffic can be managed and adjusted more quickly and dynamically, relying on different rules and real-time parameters.

Figure 2.3: Difference between traditional and software-defined networks




As stated, three components are required to separate the traffic in the two planes: a centralized SDN controller, SDN-enabled switches, and a management protocol like OpenFlow. With this separation, all the control logic is moved to the controller, allowing the use of cheap switches that only have forwarding elements.

SDN becomes useful for different network functions and concepts: first of all, there's **Virtualization**, allowing the use of network resources without worrying about where they're located or how

they're organized. **Orchestration** is also possible, managing thousands of devices simultaneously, dynamic scaling, automation, performance, and many more.

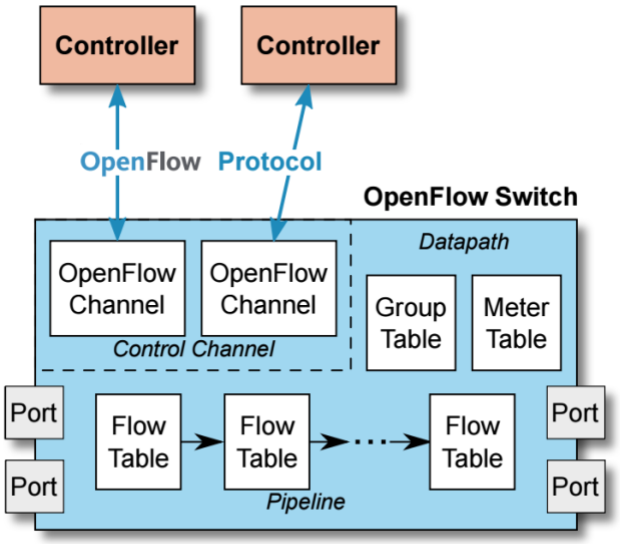
SDN allows multiple use cases for today's networks: an example is predictability, allowing schedules for traffic peaks, using different routes to balance spikes. Another example is maintenance: when a router is unavailable, the SDN controller can configure alternative routes easily without creating disruption.

#### SDN switches and OpenFlow

SDN-capable switches can be either hardware or software, and nowadays, there are many compatible switches. As software solutions, in virtualized infrastructures and data centers, the most popular implementation is Open vSwitch [29]. OVS also supports OpenFlow and will be used in the experimental part of this dissertation.

OpenFlow involves a flow table in each switch, with all the routing rules stored. When a packet arrives, the header fields are matched with the entries in the table: if any matches are found, a counter is updated, and the switch performs indicated actions; otherwise, it contact the controller. The matches can be made, for example, by source or destination MAC address, by incoming port, by VLAN IDs, by source or destination IP, protocol and many more. Examples of actions are all (send to all interfaces except incoming interface), controller, table, normal, flood.

Figure 2.4: OpenFlow switch architecture




### 2.2.2 Network function virtualization

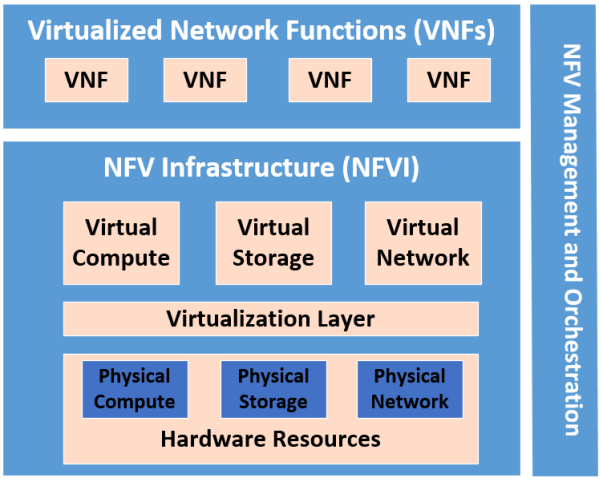
NFV, or Network Function Virtualization, is a network architecture concept that virtualizes physical networks and resources, with the objectives of decoupling software-based network functions from physical equipment, deploying these functions on demand with more flexibility. A centralized management is required, and it's independent of the underlying hardware; it allows the assignment of resources to specific services according to their current needs. This allows dynamically scaling of the service.

NFV provides efficient use of the infrastructure, increasing the freedom to deploy new services without worrying about the hardware.

The functions, or services, relies on virtual resources that are dynamically assigned to them. Resources can be clustered in computing, storage, or network resources. There's also a transversal layer responsible for managing and orchestrating of services and resources and how the functions collaborate with each other.

Network virtualization is possible by slicing the available resources: each function should have its fraction of **bandwidth** on a link and also its view of the **network topology** and the connectivity

Figure 2.5: Network Function Virtualization architecture




between them. Also, the **traffic** should be associated with one or more virtual networks, isolating a set of traffics from another. Finally, computational resources and forwarding tables must also be sliced.

## 2.3 Network orchestration and NFV-SDN Architectures

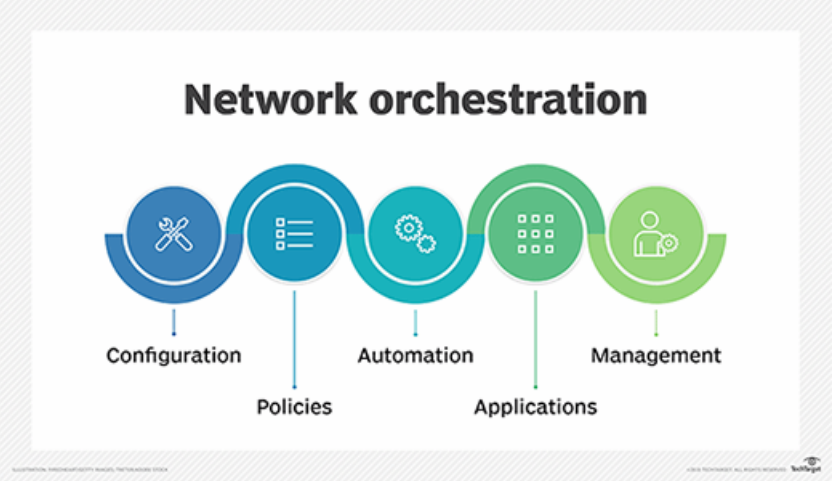
NFV and SDN are complementary technologies that do not depend on each other but have similar goals. While SDN needs new interfaces and control module applications, NFV requires moving network applications from dedicated hardware to virtual containers on commercial-off-the-shelf hardware.

Despite this indipendence, in the last years many researchers have focused on making SDN and NFV work together as a single system, following the concept of network orchestration.

According to ETSI [12], orchestration is a set of coordinated processes that automate the management and control of information systems to reach a common goal, emphasizing that orchestration could be provided in multiple functional blocks, with no priority over othere.

The Open Networking Foundation (ONF) [14] defines orchestration differently, as usage and selection of resources by orchestrator for satisfying client demands according to service level. Orchestration is considered a feature of the SDN controller, being a crucial part of SDN architecture.

Figure 2.6: Network Orchestration



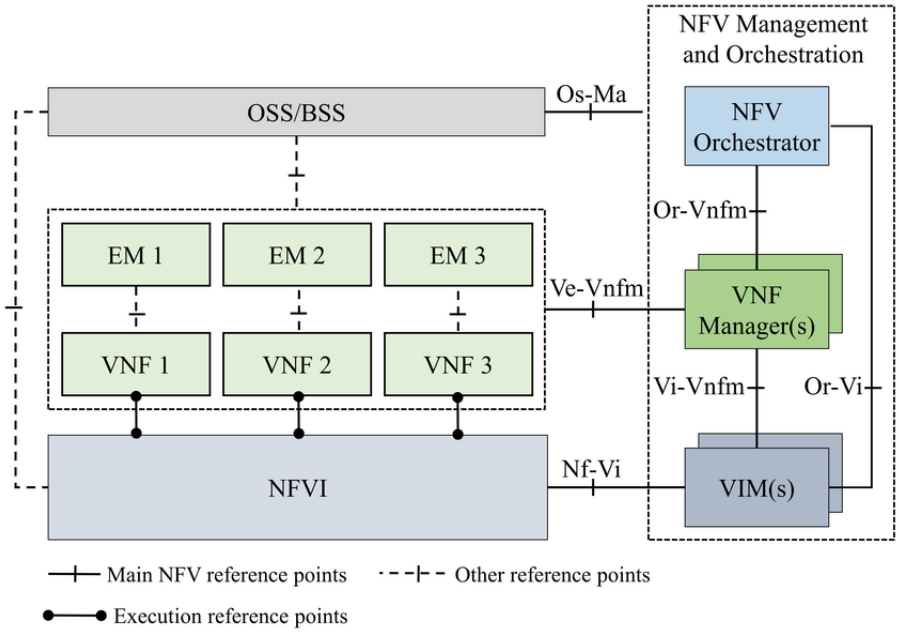

As stated by Akamai [1], with network orchestration operators achieves two important goals: **increased flexibility**, in terms of how services and resources needed are allocated automatically, and **decreased operational costs**, having a virtualized, all-IP delivery infrastructure that meets the unpredictable demands of subscribers.

### 2.3.1 Examples of NFV-SDN Architectures

With the evolution of NFV technology, many orchestration platforms emerged, having some commercial products available on the market and multiple open source initiatives. The most popular options are OpenSource MANO, hosted by ETSI; OPEN-O, hosted by the Linux foundation; CORD, governed by Open Networking Foundation; and Cloudify by Gigaspaces. The solution proposed by ETSI is the most popular and the de-facto reference NFV architecture.

**ETSI SDN-NFV MANO architecture**

Figure 2.7: ETSI NFV Architectural Framework




The ETSI architecture consists of four main blocks:

- **Network Function Virtualization Infrastructure (NFVI):** it contains the virtual and physical resources needed for the internal functions and to communications between VNFs
- **MANO:** manages tasks and automates the NFV architecture, and it also includes the VIM, the VNF manages, and the Resource Orchestrator
- **Network Management System (NMS):** it manages the virtual network and contains the Element Management (information and events referred to VNFs), and the VNFs
- **Operation/Business Support Scheme (OSS/BSS):** includes the applications used by the ISPs, primarily management and system-level applications that provide network services.

### 2.3.2 Advantages of network orchestration architectures

Network orchestration is still not a requirement in implementing software-defined environments. Still, orchestration provides a faster path to digitizing and automating, making quicker to deploy functionalities, introducing innovations, and resolving problems.

- **Faster provisioning:** provisioning of new services of applications can take only a few days, especially in a hosted environment
- **Remote deployment:** the configuration of appliances and future updates can be managed remotely and automatically, reducing the hours required by operators.
- **Performance assurance:** if there are any malfunctions, orchestration allows to re-route traffic without providing interruption to customers

- **Business benefits:** orchestration allows less-routine tasks reallocating staff to other areas. It's possible to implement automation and artificial intelligence.

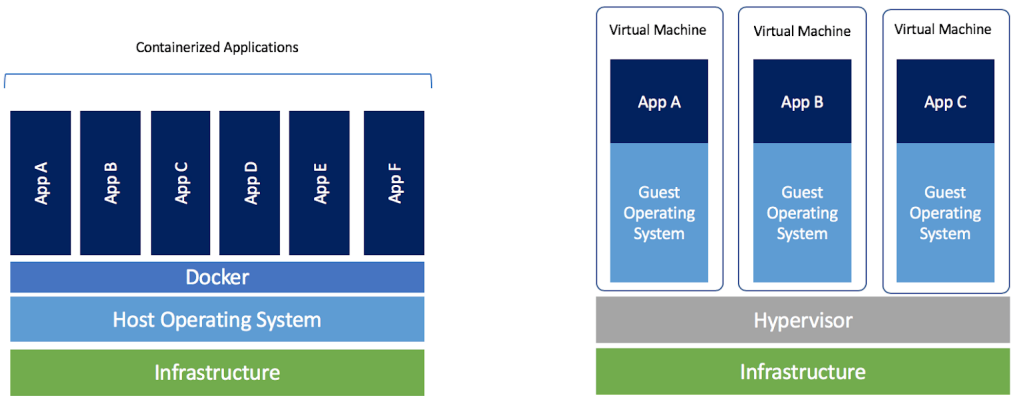
## 2.4 Containerization technology and Docker

Containerization technology is the biggest competitor to virtual machines when speaking about virtualization. Despite this, they have different use cases complementing themselves.

VMs are virtual environments that run entire operating systems. They provide more isolation, not sharing the host kernel. They're controlled by the hypervisor, responsible for gaining hardware access and control to the guest systems. This solution usually provides more resources than the application needs, having increased boot and process time.

On the other hand, containers use the same kernel, with the hypervisor program (e.g., Docker) that creates several policies to separate containers from the host OS. Containers are executed natively on the host OS using its kernel, running as a single process and using the system memory, reducing the overall resources used.

Figure 2.8: Container apps vs. VMs




Container-based applications are the preferred solution when building virtual-network functions for different reasons:

- They can run easily legacy applications
- Offers performance benefits running on bare-metal, without the overhead of any hypervisor
- Make possible higher density and resource utilization in datacenters
- The adoption for new technologies is accelerated, having isolated secure containers
- They reduce "shipping" pains, delivering code to customers easily

### 2.4.1 A container solution: Docker

Docker [10] is an open-source project that offers a set of Platform-as-a-Service (PaaS) products for developing, deploying, and executing containers. Containers in docker are instances of an image with a state of its execution. Images are executable packages containing the application, libraries, configuration files, and environment variables.

Docker is powered by a technology called **runC** [11] that offers a sandbox environment abstracting the host without rewriting the entire application, ensuring security and isolation.

## 2.5 A testbed for Virtualized Networks: ComNetsEmu

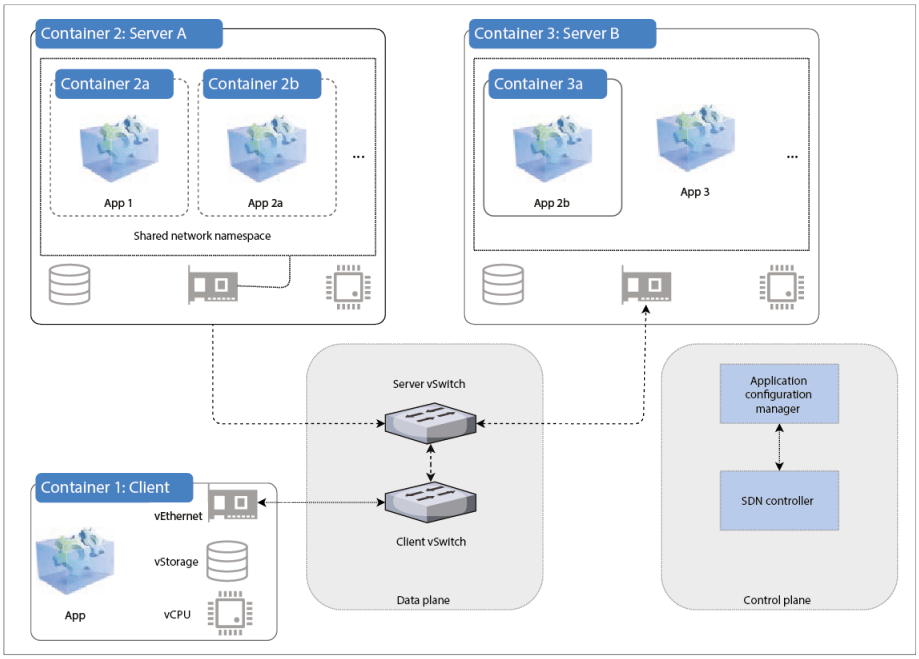
ComNetsEmu [9] is a holistic emulator of virtualized networks that provides an SDN and NFV emulation environment based on open-source packages, enabling replicable research and development on COTS hardware like a generic laptop [37].

Due to the complexity of NFV systems implementation, ComNetsEmu focuses only on the NFV Infrastructure.

ComNetsEmu relies on Mininet, a lightweight SDN emulator that enables prototyping a complete networked system using the features provided by the Linux OS. Mininet can create a virtualized network consisting of network applications, hosts, switches, and routers on a single physical machine. It also supports OpenFlow and SDN elements and provides an independent network stack and a virtual network interface (veth) for each host. This interface can be connected to virtual switches like Open vSwitch, and every link can be configured with parameters like latency and bandwidth.

ComNetsEmu deploys a Docker environment inside each Mininet host, resulting in a Docker-in-Docker nested virtualization architecture, providing isolation between Mininet and the hosts. Additional classes are provided for the orchestration of these containers. This architecture is used to mimic an actual physical host that runs Docker containers, representing scenarios where multiple VNFs have to share the limited computing resources of a single host. As a result, in ComNetsEmu is also possible to migrate VNFs between hosts to satisfy latency requirements and provide better Quality of Experience. The DIND system offers an immediate benefit with the reduced overhead compared to the deployment of entire virtual machines while maintaining ease of use and flexibility.

Figure 2.9: Architecture of ComNetsEmu



# 3 Implementation

This third chapter will report the implementation done in this research, starting from the ideas described in the first chapter and using the technologies illustrated in the second chapter.

The base system will be two NGINX apps: a video server (or transcoder) and a cache server. These apps will then be containerized with Docker and deployed in an emulated network topology using ComNetsEmu. Inside ComNetsEmu will finally be executed a service migration script that will be explained in a few moments. The chapter will end with the results of this implementation and its key points.

## 3.1 Architecture

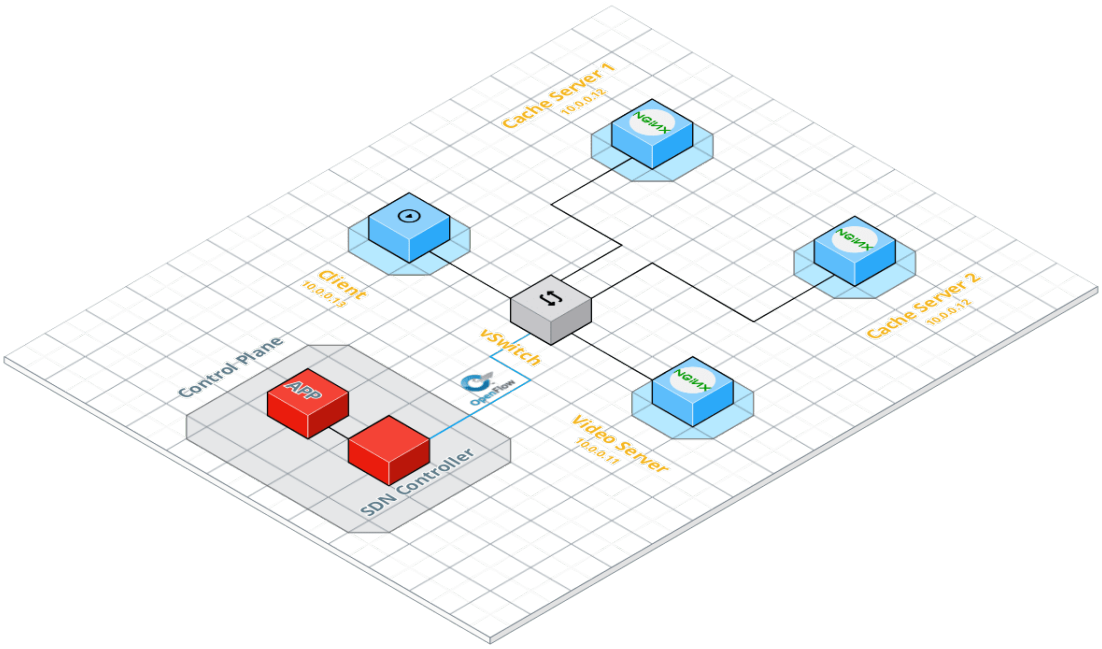
The basic concept of this project's implementation is a service migration system of a live video streaming app.

To achieve this, the client will connect to a server offering him the HTTP Live Streaming (protocol described in 2.1.1) content. Still, instead of connecting to the transcoder, it will connect to a cache that will work as a bridge between the transcoder and the client. This cache will be deployed twice in two different hosts but having the same IP and MAC addresses, and the switch will be configured with OpenFlow to send the traffic to the correct cache.

A diagram of the architecture deployed in ComNetsEmu can be found in Figure 3.1.

In the first step, the service will be migrated from a cache to another in a dumb way, using only time intervals. Then, as a next step, the migration will be done, measuring the latency between cache and client, choosing the best one.

Figure 3.1: Architecture of the system




The diagram shows that the emulated network will have four Docker-in-Docker instances (the ones in blue) containing a video server, two cache servers, and one client with a video player. The network also features a virtual switch (Open vSwitch), managed by an SDN Controller using the OpenFlow protocol (these concepts have been explained in Chapter 2). Finally, an SDN Controller is deployed

on the network's control plane, but in this experimentation its role will be replaced by a Python script running in ComNetsEmu.

## 3.2 Setup of NGINX

The first step of this methodology consisted of founding software that will act like a cache for a live video stream, a "bridge" between the server and the client that can be seen as a node of a CDN. This aspect will be a crucial point to deploy the service migration system subsequently.

After analyzing unsuccessfully other solutions like youtube-cache [39], that are no longer working due to the new YouTube policies and configurations, the choice felt on NGINX [21], using his integrated `upstream` module. NGINX was already the choice for the transcoder server, being the most used platform for HLS content delivery.

### 3.2.1 NGINX HLS server configuration

NGINX is a web server, written originally by Igor Sysoev and now maintained by Nginx, Inc. It offers high performances (it can handle more than 10,000 simultaneous connections with a low memory footprint) and can be used as a reverse proxy, load balancer and HTTP cache. According to W3Techs [34], NGINX is used by 34% of all the websites in their survey.

To deploy the NGINX server, it needed to compile NGINX from source, adding the `nginx-rtmp-module` [23], not included in the standard NGINX installation, enabling the webserver to work as a media streaming server. After having compiled and installed NGINX, it's required to modify its configuration before starting to use it. The configuration file is usually placed in `\etc\nginx\nginx.conf`. The relevant parts of this configuration are shown below.

```
rtmp {
    server {
        listen 1935; # Listen on standard RTMP port
        chunk_size 4000;

        application show {
            live on;
            hls on;
            hls_path /mnt/hls/;
            hls_fragment 3;
            hls_playlist_length 8;
            # disable consuming the stream from nginx as rtmp
            deny play all;
        }
    }
}
```

Listing 3.1: Configuration of the RTMP server, activating the HLS stream

```
types {
    application/dash+xml mpd;
    application/vnd.apple.mpegurl m3u8;
    video/mp2t ts;
}
```

Listing 3.2: Configuration of the file types provided by the HTTP server

The complete configuration for an HLS server using NGINX can be found at [28].

With this configuration, the server can eventually receive an RTMP stream on port 1935, then publish it as an HLS stream under the HTTP server. The drawback in this is that is required another device that sends an RTMP stream to the server, requiring an additional host in the emulated network. To avoid that, the solution has been sending the stream from the same host, using `FFmpeg` and a short video looped endlessly, simulating a stream.

```
ffmpeg -re -stream_loop -1 -i /root/stream.mp4 -vf drawtext="fontfile=monofonto.ttf:
    fontsize=96: box=1: boxcolor=black@0.75: boxborderw=5: fontcolor=white: x=(w-
    text_w)/2: y=((h-text_h)/2)+((h-text_h)/4): text='%{gmtime\:%H\\\\\:%M\\\\\:%S}'"
```

```
    -vcodec libx264 -vprofile baseline -g 30 -acodec aac -strict -2 -loop -10 -f flv
    rtmp://localhost/show/stream -nostdin -nostats </dev/null >/dev/stdout 2>&1 &
```

Listing 3.3: Command used for sending an RTMP stream to NGINX using FFmpeg

The command showed above takes an MP4 video looped endlessly, with the clock time overlayed (to check out if it works correctly), and sends it to the local RTMP server. The latest parameters are needed to make `FFmpeg` work on the background in the Docker container that will be deployed later; otherwise, the process will stop not having an interaction with `stdin`.

To check if the HLS stream is working, this configuration was initially deployed on a classic VM and tested using VLC Media Player on the host machine. Having this working properly, it was possible to move to the NGINX Cache configuration.

### 3.2.2 NGINX cache configuration

The best choice to deploy the two cache servers that will act as a CDN for the emulated network was using NGINX with the modules `proxy` [22] and `upstream` [24].

The first one allows passing requests to another server, while the second one is used to define groups of servers referenced by the directives used by the first one. The server is configured to activate a cache on `.m3u8` files (the chunks of the HLS stream) and store them locally to serve multiple clients passing a single request to the transcoder.

This solution can also be used as a classic load-balancer, using multiple cache servers and balancing them using a round-robin DNS technique.

Below are listed some significant code snippets of the NGINX configuration.

```
http {

    upstream hls {
        ip_hash;
        # ADD YOUR SERVERS HERE - ONE PER LINE
        server 10.0.0.11:8080;
    }

    ...
}
```

Listing 3.4: Configuration of the RTMP cache, upstream

```
server {
    listen 8080 default_server;

    add_header X-Cache-Status $upstream_cache_status;

    location ~ \.(m3u8|mpd)$ {
        proxy_cache_valid 200 302 5s;
        proxy_pass http://hls;
    }

    location / {
        proxy_pass http://hls;
    }
}
```

Listing 3.5: Configuration of the RTMP cache, enabling the server with cache on m3u8 files

To test this configuration, it's enough to create another VM, changing the upstream server with the transcoder machine's IP address, and using VLC in the host machine opening the HLS stream using the cache's IP instead of the transcoder's one.

Having these two configurations working, it is possible to convert them in Docker containers needed for the deployment in the ComNetsEmu network.

## 3.3 Software conversion into Docker images

With the NGINX configurations working, the next step before developing the service migration script in ComNetsEmu is to create two Docker images containing the NGINX services. As seen before,

containerized applications are the best choice when implementing VNFs because containers can be moved from one host to another in an easy and fast way when developing a network orchestrated system.

A third container emulating a client has also been deployed, with VLC Media Player and a browser installed. After building all three containers, they are published to the Docker hub, making their pulls more immediate in ComNetsEmu.

### 3.3.1 Building the HLS server image

Because NGINX needed to be compiled from source with the `nginx-rtmp-module` to have HLS working, the choice for this container has been to split the build into two different stages: one called `builder` and the second one as the main stage. Both the steps use `ubuntu:18.04` as the base image.

In the `builder` stage, the source code of NGINX and the module are downloaded, compiled and installed, to keep all the code separated from the primary container resulting in a cleaner environment. Once NGINX is installed, its binaries are moved to the main stage, where all permissions and configurations are created. The correct NGINX configuration seen before is copied to the container, and a bash script containing the `FFmpeg` command is set as an entry point for the image.

### 3.3.2 Building the cache and the client images

Building these two images is pretty straightforward: it's enough to update the repositories and install the required apps (like NGINX, VLC Media Player, a browser). Like the server image, the cache needs to create a user with specific permissions to make NGINX work. Also, it's necessary to copy the proper configuration file in `/etc/nginx/nginx.conf` to enable the correct functionalities.

One issue encountered when building the client image was with VLC Media Player, which cannot be run as root: using the command `sed -i 's/geteuid/getppid/' /usr/bin/vlc`, it's possible to run it without issues, although is not recommended to do that in production environments. Being this an experimental work, there are no significant problems.

## 3.4 Implementing containers in ComNetsEmu

After having built the three containers with Docker and published them to the Docker Hub (this has been done for convenience, reducing the time needed to build the container every time inside the VM), it is possible to start working with them in ComNetsEmu.

With containers, many other production solutions implement a service-migration-like system: one example is Kubernetes. Initially developed by Google, it is an open-source platform that automates operations with containers deployed in clusters, like load balancing, storage orchestrations, and self-healing. The problem with Kubernetes is that it doesn't offer a real service migration solution, to improve the user's Quality of Experience without significant service interruptions.

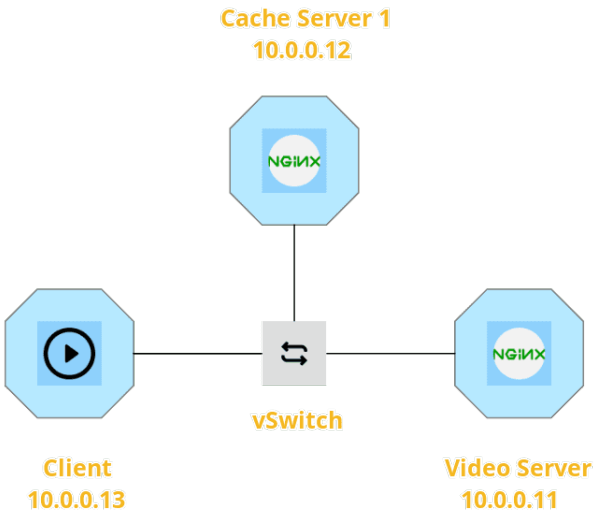
Back to ComNetsEmu, the first step before start using the containers is to create an emulated network with standard switching: 2 servers (transcoder and cache) and a client, connected to a switch, all in the same network subnet (10.0.0.0/24). This is achieved with a Python script that interacts with Containernet. The architecture of the system is shown in Figure 3.2.

When configuring the network with the script, it is also necessary to manually configure the switch interfaces and the links: in this case, all the links have a bandwidth of 1000 Mbit/s and a latency of 1 ms. With the network set up, it is possible to deploy the containers inside the virtual hosts, using the `addContainer` function provided by the `VNFManager` of the ComNetsEmu's network. It is required to provide to the function the virtual host where the container will be deployed and the container's name.

To interact with the containers, ComNetsEmu provides the function `spawnXtermDocker`, which creates an Xterm connected to the container provided as an argument. To have this working is required that the ComNetsEmu VM is configured with the correct `DISPLAY` as an environment variable and that the physical host has an X11 Server installed (XQuartz for macOS, Xorg for Linux, or VcXsrv for Windows). With the Xterms working correctly, it is possible to interact with the containers to troubleshoot issues and test the system.

When using VLC Media Player inside the client container, the X11 Forwarding should be configured in turn. In this case, the X11 server will not be accessible via `localhost`, but it's required to enter

Figure 3.2: Simplified architecture in ComNetsEmu




the physical machine's IP address and disable the Access Control List of the X11 server (on Linux or macOS, the command `xhost +` is enough).

Now it is possible to use VLC Media Player and test the system in ComNetsEmu, opening the stream provided by the cache server.

## 3.5 Service migration with ComNetsEmu

With the simplified network working, the service migration can now be deployed. The network is extended with another cache server, having the same IP address of the first cache. In addition, the SDN Controller is now required to manage the flows in the switch and achieve the service migration functions. For this experimentation, it will not be used a proper SDN controller because the Python script will replace its role.

The architecture of this step is shown in figure 3.1.

As stated before, the second cache server will have the same IP address as the first one and will also have the same MAC address. This is required to have a transparent connection from the client and cache server without requiring a reconfiguration on the player. It will be the task of the network and the SDN controller to manage the switches' flows and ensure the correct forwarding of the packets.

In the first place, the script will enable the first cache and migrate to the second one after about 60 seconds, return to the first after about 30 seconds, and make four migrations in total. Therefore the migration will not be based on network conditions to get a first confirmation about the system working without issues.

Before starting with the migrations, the network needs to be configured to have the first cache working: this is achieved with two `add-flow` actions on the switch, using the utility `ovs-ofctl` that allows the configuration of the OpenFlow table in the switch. The two flows added will send the packets coming from the client port and the transcoder to the first cache. An example is shown on Listing 3.6.

```
check_output(
    shlex.split(
        'ovs-ofctl add-flow s1 "in_port={}, actions=output:{}"'.format(
            s1_client_port_num, s1_cache1_port_num
        )
    )
)
```

Listing 3.6: OpenFlow command for adding a flow in switch 1

When it's time to migrate the service, it's essential to deploy the container on the second cache server, using the `addContainer` function as before. This is done before redirecting the flows to the second cache to give the system enough time to deploy and start the application. Then, when the server is ready to operate, the migration is completed with the `mod-flows` command, moving the packets coming from the client and the transcoder to the second cache. Thanks to their configuration, it's not needed to migrate the caches' state because they will automatically connect to the transcoder and get the stream chunks as soon as the client sends a request.

Using the Xterm and VLC Media Player as stated before, it's possible to ensure the proper functioning of the service migration: when the migration happens, and the player completes the playback of the chunks that the first cache sent to him, it starts to buffer, trying to load new chunks, and hangs for less than two seconds.

The result is clearly good, but it will be analyzed and compared to standard solutions in the following chapter. The experimentation focuses now on optimizing the system analyzing the network performances.

## 3.6 Extending the system analyzing network performances

Having the system working without significant issues, the next step in its evolution is introducing the network's status, which causes the service migration. In a real environment, the network could have fluctuations due to high traffic, congestion, or a user with a mobile device with poor signal.

With ComNetsEmu, it is not yet possible to simulate mobile links (4G/5G) that could have random changes in their quality. Still, it's possible to introduce higher latency in the connections between hosts.

Therefore, it's possible to give the two channels between the two caches and the client different latencies (e.g., 10ms for cache1, 60ms for cache2), then let the SDN Controller (in this case, the Python script) analyze the two links performances and choose the best one, migrating the server from a poor link to a better link.

This configuration is certainly not like what happens in a real network environment, but it's a starting point of what it's possible to achieve with this kind of system.

The network is configured to start with an established link between cache and client with a higher latency than a second available link. Introducing a timer (e.g., 15 seconds), every time it expires, the SDN controller will test the latency between cache and client using both the links: if the unused link has a lower latency compared to the used one, the migration will be triggered. Clearly, these two latencies will be fixed in an emulated environment, and the migration will happen only once. Still, considering a global network, where its conditions are constantly changing (link saturation, outages in the equipment), it's helpful to run this test continuously. In this way, it will be possible to keep the service operational, and the QoE (Quality of Experience) for the user will be kept good.

It is important to emphasize that this kind of analysis cannot be made with bandwidth tests like iPerf, because it will saturate the channel trying to measure the maximum achievable bandwidth [18]. A solution can be using bandwidth estimation techniques like Ochoa's ABEP (Available Bandwidth with Error Probability) [25], which uses passive measurements which does not saturate the available bandwidth, and obtains accuracy in the estimation by measuring the error probability and the channel idle time compared to the measurement period.

# 4 Final results

With the system working, it's now possible to analyze its performances, the advantages of this solution, and significant problems.

The service migration solution using SDN and NFV will be compared to traditional solutions highlighting the advantages and disadvantages of both. In addition, some graphs about the system status and performance will be shown.

## 4.1 Experimental results with service migration

The service migration system showed immediately pretty good results: when the migration happens, the client can continue with the streaming's playback without significant issues. Indeed, the player used during the tests (VLC Media Player) didn't show any alerts or closed the connection. In the following section, we'll see that with traditional systems, it is common that the player closes when it cannot fetch the stream chunks.

One minor issue that the system showed was a little of buffering a few seconds after the migration took place. HLS serves the stream sending a playlist containing a few chunks (small pieces) of the video: usually, a playlist contains about three chunks that last about 2/3 seconds, so a playlist lasts about 8/10 seconds. When the player finishes the playlist that has been received just before the migration, it tries to reconnect to the server, which will be a newly created cache. This cache, however, will not have the correct chunks that the transcoder was sending right during the migration. As a result, the player will not be able to get the proper chunks and will buffer for a few seconds, and then it will realign itself with the streaming without stops. It is not a seamless playback, but a few seconds of buffering that doesn't require the user to restart the player is still a pleasing result.

To get more detailed results, the choice fell on Apache JMeter [2], an open-source Java application designed to load test functional behavior and measure performance. To test HLS streams with JMeter, the only free tool available is HLSPlugin by Blazemeter [5], which solves the HLS complexity and display results such as media playlists and media segments with parameters such as latency, response time, and bytes received. Some significant graphs that JMeter could create are: connect time, bytes throughput, and connect time.

Figure 4.1: Overall connect times

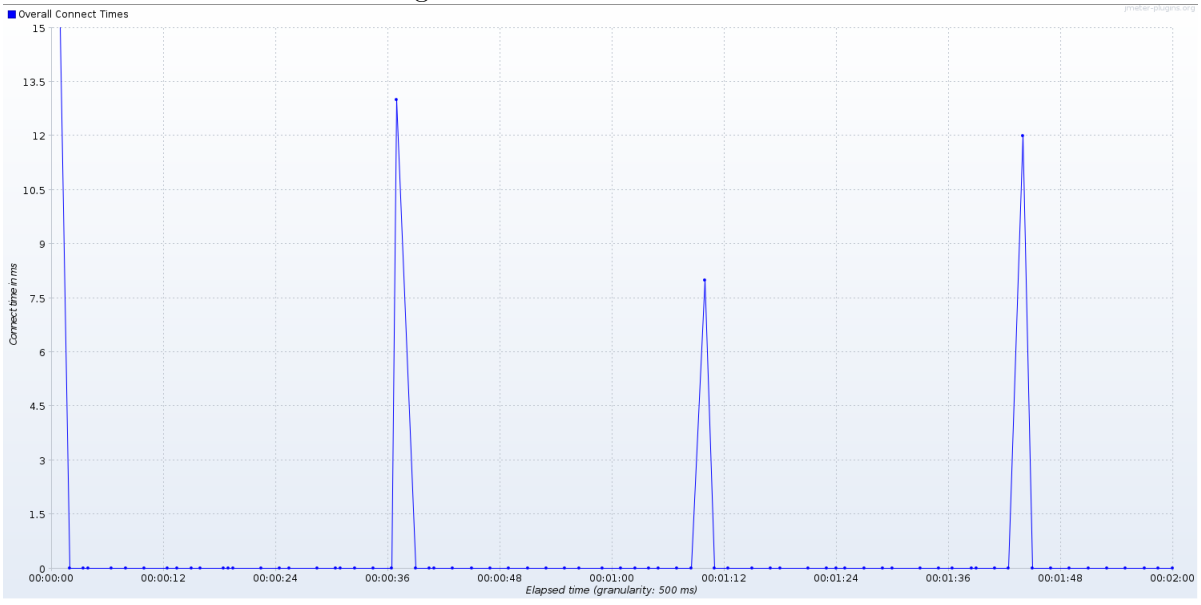
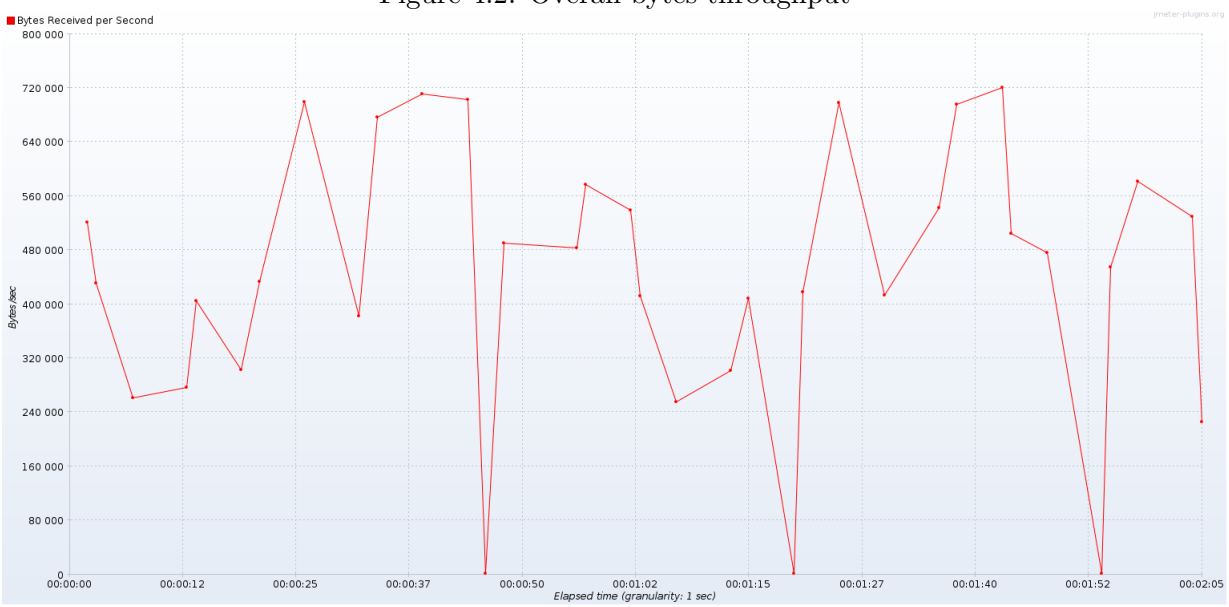


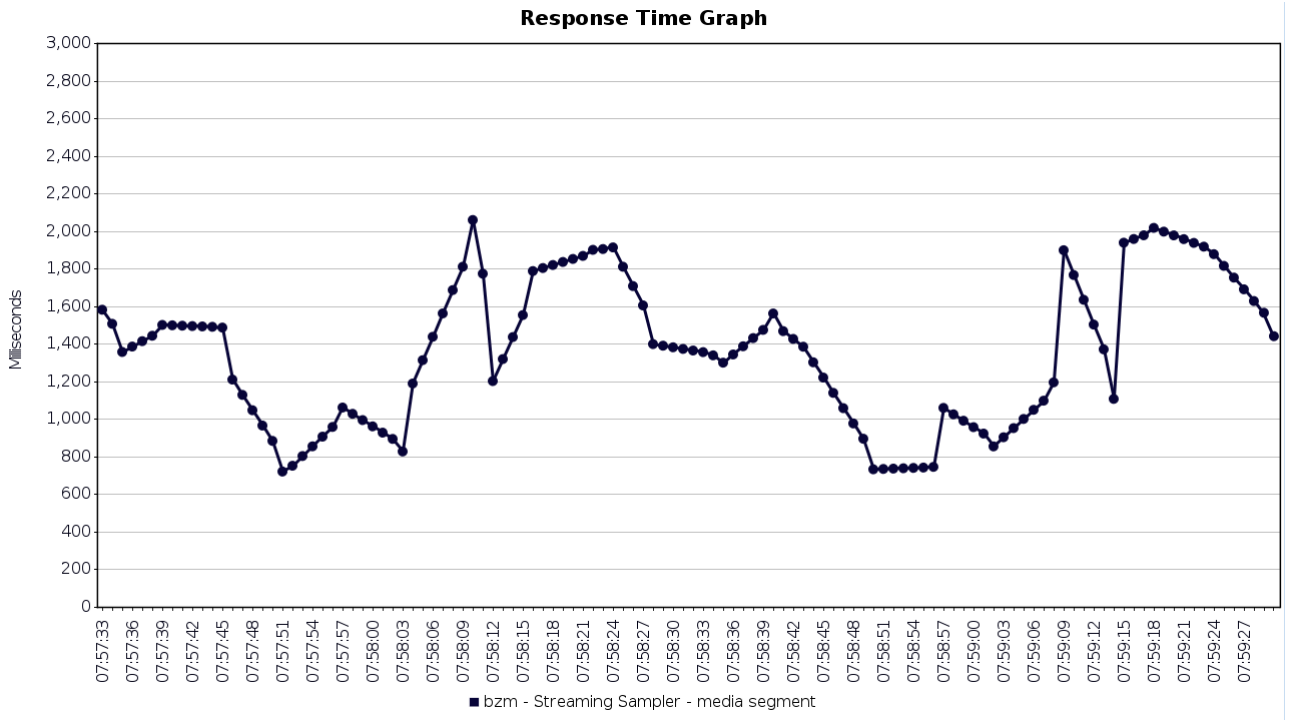
Figure 4.2: Overall bytes throughput




The first graph shows clearly what happens during the three migrations executed during the test: three spikes in the connect time to the server that represents the player trying to reconnect right when the service is migrating. It's evident that the system returns immediately in a normal state right after.

The second graph shows the received bytes throughput during the whole test: the three drops make evident when the migration occurs, causing a pause in the stream's reception. Like the first graph, the throughput returns to typical values immediately.

Figure 4.3: Overall response time




The third graph represents the response time during the whole test: two spikes during the first and the third migration shows a little bit of buffering due to the state of the new cache server. Overall the response time doesn't show significant increments during the test, making the system quite stable, as said before.

These graphs can show easily how the system is working without providing significant interruptions

and especially that restores to a normal state in a few seconds.

### 4.1.1 Complexity in an SDN configuration

One drawback in a service migration system that takes advantage of Software Defined Networks and virtualization is that the SDN needs to be appropriately configured. Especially in production environments, the network is managed by the controller (like Ryu), and its configuration requires experience and trained staff that knows how it works and how to set it up properly.

The SDN configuration was simplified and delegated to the Python script instead of an SDN controller in this work. However, knowledge in virtualization technology (VMs, Docker) and SDN (e.g. OpenFlow commands) is required.

This leads to a need for more specialized staff and more investment in training and equipment for production environments, but then the network can be managed and improved a lot compared to traditional systems.

### 4.1.2 Computing resources and time required for service migration

During the development of the system, particular attention has been given to the computing resources available and the bottlenecks due to the virtualized environment. The primary ComNetsEmu VM has been configured with four vCPUs and 8 GB of RAM, which should be enough for a video streaming service.

It should be noted that, because of the architecture of ComNetsEmu, the result is a three-level virtualization: the principal VM, created with Vagrant and Virtualbox, contains an emulated network of Containernet (fork of Mininet that allows using Docker containers in the topology) hosts. In this way, every host can be deployed as a container with specific functions, but with ComNetsEmu, this host contains another Docker instance with the proper service (in this case, the transcoder or the cache server). As said in section 2.5, this allows a mimic of an actual physical host that runs Docker containers, representing multiple VNFs.

This three-level virtualization leads to a significant bottleneck, especially when deploying video streaming services, requiring not a few resources for encoding and playback.

In addition, when the migration happens, the CPU's usage increases a lot: a Docker container is being deployed in real-time, and two significant flows (`transcoder-cache` and `cache-client`) are being moved to the second cache. Being these two flows based on pretty big links (1 Gbps per link), it's clear how much computational resources are required.

Initially, the transcoder was based on 720p video, and in the moment of the service migration, the CPU became saturated, causing the playback to lag and several times to stop, making the analysis difficult. The video source was replaced with a reduced quality version (360p) for lowering the resources needed to solve this problem. This allowed the migration to happens without stops caused by the increased CPU usage.

#### Time required for migration

An additional measure analyzed to evaluate the system performances has been the time between the beginning and the end of the migration, including deploying the container and modifying of the flows.

The measure doesn't include the time for the cache server to come back online because the only possible measure would be a ping. The issue with ping is that the cache server always replies, even right after the migration, and could have very high results. This, of course, will only ruin the calculation for the time required, so it's better to consider only the "software" time to re-deploy the container and modify the flows.

Measuring the time required for several migrations, there's an average of 300 milliseconds with 48 `ms` of standard deviation, that's a pretty good value for this kind of system.

To reduce even more this parameter, it's possible to let the secondary cache container always deployed in the host, but in a real environment may not always be possible: if other services require the resources, it's a waste to leave the container deployed so that the system will lose one of its main advantages. Also, in production, it's possible that the streaming service needs to be scaled JIT (Just In Time) without predictions, probably due to a rapid increase in viewers for special events.

## 4.2 Comparison with traditional ABR systems

When comparing a service migration system for video streaming to traditional approaches, like adaptive bitrate ones, many advantages come out.

In the first place, as already said previously, in a traditional solution that must serve thousands of users without knowing any forecasts, it is necessary to reserve several servers almost wholly dedicated to video streaming. This leads to a waste of resources that could be used to more critical services and waste money to maintain the system and energy to power it. When using virtualized networks, this problem is eliminated: the service can be scaled at will following the requests in real-time, reducing the wastage of resources and energy.

Another issue that traditional systems arise is the QoE (Quality of Experience) for the user playing back the stream. Adaptive bitrate allows to reduce the stream quality automatically if the bandwidth is not good, but when there's a link in the internet backbone that has problems like high latency or packet loss, due for example, to saturation, ABR can't do anything. Software Defined Networks and virtualization can solve this issue: if the client is connected to the server with poor links, the SDN Controller, which constantly analyzes the network, will "move" the client to a link with better parameters. This happens transparently to the user because all the analysis and operations will occur in the ISP network, where the client cannot operate. In conclusion, thanks to the SDN "magic", the user will enjoy an almost constantly excellent Quality of Experience without interruptions.

## 4.3 Advantages and issues with containers

During this project's implementation, container technology has played a crucial role and has shown many benefits: at first, containers are fundamental when deploying virtualized networks and VNFs (Virtual Network Functions).

When a service is "containerized", it can be deployed to users in seconds, thanks to their reduced startup time compared to bare-metal systems or virtual machines. Containers allow the development of microservices of an application, fine-tuning the resources allocated, and providing more stable services. Thanks to this granularity, containers are lightweight and enable hosts to run many of them at the same time.

Also, updates can be rolled out more easily and fastly, and in case of issues, it's possible to roll back almost instantly. Using an orchestrator that manages all the containers, every operation can be done in seconds without manual interventions. Without containers, a real-time service migration will not be possible without causing significant interruptions.

In conclusion, a drawback in containers technology is the little overhead required by the orchestrator that manages every container, but that's still less than a virtual machine hypervisor.

# 5 Conclusions

Network virtualization and Software Defined Networks are increasingly becoming a standard for modern, high-performance, and scalable networks.

In this dissertation, we've used these technologies to provide a starting point in improving live video streaming services, that as stated in 1.3, are becoming more and more popular to deliver content to worldwide users, and in a few years, they will be the majority of the global traffic.

Using ComNetsEmu, a holistic testbed/emulator for Software Defined Networks, we developed an experimental environment that uses containers and the concept of Virtual Network Functions (VNFs) to implement a live video streaming service, optimized migrating a cache between different servers in the network. Analyzing the performances, it's possible to highlight the best server relatively to the client and move the service to it. This is a breakthrough compared to traditional adaptive bitrate systems, which can suffer from link saturation and high latencies, having the client always connected to the same server.

Using SDNs, the incoming connection from the client will be moved to different servers dynamically without the user knowing and without requiring its intervention. Analyzing this system with proper tools has been pointed out that the interruption for the migration lasts only a few seconds because the client will continue to receive packets from the same address, and it will need very little time to realign with the stream.

Obviously, this research has been limited due to the given time, but many other features and future developments can be made. The first idea is to scale the system making it capable of self-configure according to how many users try to connect to the service. A service manager will automatically increase the number of caches in the network, placing them in strategic points and moving clients to balance the available resources. As showed in the implementation, the network will be constantly monitored, and users will be moved between caches to ensure an excellent QoE.

This network monitoring can be likewise improved in future developments: instead of using only a latency test, bandwidth estimation techniques could be introduced, and also many other client or server measures can be added. Thinking about mobile networks (4G/5G), interesting parameters to optimize the service can be signal strength or device computing power.

Another improvement is adapting the system making it capable of serving every kind of stream, caching, for example, YouTube and Netflix. This will reduce the used bandwidth in primary links and will give the users an improved quality. Because of the restrictive functionalities of these platforms, this idea is still not possible, but instead of using proprietary systems like Google Edge Network or Netflix Open Connect, the use of ISP-wide cache services will be a win-win collaboration for both ISP and CDN operators.

Alternatively, an evolution of this system could become a commercial product, sold to major operators like TV networks: the service can provide on-demand content or live channels to a large group of users, that in the years ahead will switch from cable/DTT television to the Internet to watch contents. A dashboard for system engineers could also be added in a commercial product: a network map showing real-time the clients and their connections to the caches. The engineer will be able to fine-tune the parameters that trigger the migration, add or remove servers manually, schedule operations based on planned events, and get alerts if anything goes wrong.

With the advent of 5G Networks, this kinds of system will become more popular, and the architecture provided in this dissertation could have an exciting enhancement. Thanks to Mobile-Edge Computing, an emergent technology for 5G that creates a standardized and open environment with computational resources and storage using the virtualization paradigm, it's possible to add functionalities and improve the QoE placing the services as close as possible to users.

In conclusion, hoping this work will be a starting point for many other developments, it is clear that the global network and user demand are evolving. Modern tools and paradigms will allow developers

and ISPs to improve their skills and provide final users a secure, fast, and efficient global network, ready to evolve and scale with the growing demand of services more and more connected. The key remains in imagination and willingness to pursue more remarkable milestones.

### Appendix - Project Repository

The complete repository of this project will be available later inclusive of documentation at the following page: `https://github.com/davideparpinello/VideoFlux`

# Bibliography


[1] Akamai. The Case for a Virtualized CDN (vCDN) for Delivering Operator OTT Video. *Akamai*, 2018.

[2] Apache JMeter™. `https://jmeter.apache.org/`.

[3] Netflix to cut streaming quality in Europe for 30 days. `https://www.bbc.com/news/technology-51968302`.

[4] Video marketing statistics: The state of video marketing in 2021. `https://biteable.com/blog/video-marketing-statistics/`.

[5] Blazemeter / HLSPlugin. `https://github.com/Blazemeter/HLSPlugin`.

[6] Netflix Revenue and Usage Statistics (2021). `https://www.businessofapps.com/data/netflix-statistics`.

[7] Martin Casado, Michael Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. ETHANE: Taking Control of the Enterprise. *Computer Communication Review - CCR*, 37:1–12, 10 2007.

[8] Cisco Annual Internet Report (2018–2023) White Paper. `https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html`.

[9] ComNetsEmu public repo. `https://git.comnets.net/public-repo/comnetsemu`.

[10] Docker. `https://www.docker.com`.

[11] Introducing runC: a lightweight universal container runtime. `https://www.docker.com/blog/runc/`.

[12] Experiential networked intelligence (ENI) ETSI ISG. Context-Aware Policy Management Gap Analysis Disclaimer, Technical Report. *ETSI*, 2018.

[13] COVID-19 Pushes Up Internet Use 70% And Streaming More Than 12%, First Figures Reveal. `https://www.forbes.com/sites/markbeech/2020/03/25/covid-19-pushes-up-internet-use-70-streaming-more-than-12-first-figures-reveal/`.

[14] Open Networking Foundation. Framework and Architecture for the Application of SDN to Carrier networks, Technical Report. *Open Networking Foundation*, 2016.

[15] Guanyu Gao and Yonggang Wen. Video transcoding for adaptive bitrate streaming over edge-cloud continuum. *Digital Communications and Networks*, 2020.

[16] Evangelos Haleplidis, Kostas Pentikousis, Spyros Denazis, Jamal Salim, David Meyer, and Odysseas Koufopavlou. RFC 7426: Software-Defined Networking (SDN): Layers and Architecture Terminology. *IRTF*, 01 2015.

[17] Nicolas Herbaut, Daniel Negru, Yiping Chen, Pantelis A. Frangoudis, and Adlen Ksentini. Content Delivery Networks as a Virtual Network Function: A Win-Win ISP-CDN Collaboration. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, 2016.

[18] iPerf - The ultimate speed test tool for TCP, UDP and SCTP. `https://iperf.fr/`.

[19] G. Mann. 4k technology is the tipping point in sight? *Broadband, Journal of the SCTE*, 36(3):66–70, 2014.

[20] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Raouf Boutaba. Network Function Virtualization: State-of-the-Art and Research Challenges. *IEEE Communications Surveys Tutorials*, 18(1):236–262, 2016.

[21] Nginx server. `https://nginx.org/`.

[22] Module ngx http proxy module. `http://nginx.org/en/docs/http/ngx_http_proxy_module.html`.

[23] nginx-rtmp-module. `https://github.com/arut/nginx-rtmp-module`.

[24] Module ngx http upstream module. `http://nginx.org/en/docs/http/ngx_http_upstream_module.html`.

[25] Martha Hernandez Ochoa, Mario Siller, Hector A Duran-Limon, and Liliana Duran-Polanco. Access network selection based on available bandwidth estimation for heterogeneous networks. *Computer Standards & Interfaces*, 78:103544, 2021.

[26] Open Networking Foundation. `https://opennetworking.org/`.

[27] Mukaddim Pathan. *Cloud-Based Content Delivery and Streaming*. John Wiley & Sons, Ltd, 2014.

[28] Setting up HLS live streaming server using NGINX + nginx-rtmp-module on Ubuntu. `https://docs.peer5.com/guides/setting-up-hls-live-streaming-server-using-nginx/`.

[29] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[30] On-Demand Viewing Growing Much Faster Than Live, Says Conviva. `https://www.streamingmedia.com/Articles/News/Online-Video-News/On-Demand-Viewing-Growing-Much-Faster-Than-Live-Says-Conviva-133418.aspx`.

[31] COVID-19 Software Industry Statistics. `https://www.trustradius.com/vendor-blog/covid-19-software-industry-data-and-statistics`.

[32] More Than 500 Hours Of Content Are Now Being Uploaded To YouTube Every Minute. `https://www.tubefilter.com/2019/05/07/number-hours-video-uploaded-to-youtube-per-minute/`.

[33] YouTube Says 2019 Coachella Week 1 Live Streams Hit Record High, Preps for Second Weekend. `https://variety.com/2019/digital/news/youtube-2019-cocachella-record-views-1203193219/`.

[34] Usage statistics of nginx. `https://w3techs.com/technologies/details/ws-nginx`.

[35] 75 Staggering Video Marketing Statistics for 2021. `https://www.wordstream.com/blog/ws/2017/03/08/video-marketing-statistics`.

[36] Streaming Protocols: Everything You Need to Know. `https://www.wowza.com/blog/streaming-protocols`.

[37] Zuo Xiang, Sreekrishna Pandi, Juan Cabrera, Fabrizio Granelli, Patrick Seeling, and Frank H. P. Fitzek. An Open Source Testbed for Virtualized Communication Networks. *IEEE Communications Magazine*, 59(2):77–83, 2021.

[38] You know what's cool? A billion hours. `https://blog.youtube/news-and-events/you-know-whats-cool-billion-hours`.

[39] Youtube-cache. `https://code.google.com/archive/p/youtube-cache/`.