Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

MULTICAST CONTENT REPLICATION AND CACHING IN A VIDEO DELIVERY NETWORK WITH MPEG-DASH

TRABAJO FIN DE MÁSTER

Ramón Alonso García

2019

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

MULTICAST CONTENT REPLICATION AND CACHING IN A VIDEO DELIVERY NETWORK WITH MPEG-DASH

Autor
Ramón Alonso García

Director
Luis Bellido Triana

Departamento de Ingeniería de Sistemas Telemáticos

2019

Resumen

El vídeo está por todas partes. En la era digital, la reproducción de contenidos de vídeo ha pasado a ser parte un acto cotidiano dentro de nuestras vidas. En cualquier lugar, en cualquier momento. Existe un amplio ecosistema de dispositivos de usuario. El uso intensivo, ubicuidad, expectativas del usuario de streaming en alta calidad, reproducción continua y una mejor experiencia de usuario, desafían el dimensionamiento de la red y los protocolos utilizados para adaptarse a estos requisitos. Este proyecto presenta una solución para reducir el uso de la red y mejorarla experiencia de usuario en la reproducción de contenidos de vídeo.

El objetivo del proyecto es proponer un escenario de reproducción de vídeo bajo demanda utilizando MPEG-DASH sobre una red IPTV mejorada con el multicast de segmentos de video a servidores cache distribuidos dentro de la red gestionada. MPEG-DASH se adapta perfectamente a este escenario y mejora la experiencia de usuario y reduce los requisitos que se necesitan para dimensionar la red. La utilización de esta solución es transparente para el usuario final ya que continúa recibiendo el stream unicast con el protocolo MPEG-DASH.

Para cumplir con este objetivo, el enfoque adoptado es en primer lugar analizar el estado del arte en las redes IPTV, en particular en el diseño de red y en los protocolos multicast y unicast utilizados en el streaming de vídeo. A continuación, varios proyectos de código abierto se han evaluado para construir el escenario práctico. Es entonces cuando con todos esos elementos creamos una red IPTV simulada. Finalmente, se definen los grupos multicast y los componentes multicast se han diseñado, desarrollado y desplegado en los servidores Origen y Cache. El escenario propuesto es un ejemplo en ejecución.

Utilizando la distribución multicast a elementos de red más cercanos a los usuarios finales se permite una entrega más rápida y se reduce el ancho de banda utilizado dentro de la red. El uso de protocolos multicast en las redes de distribución y agregación de un servicio IPTV, junto con protocolos unicast de streaming adaptativo (MPEG-DASH) proporcionan una solución para hacer frente a los futuros retos del streaming de vídeo.

Abstract

Video is everywhere. In a digital era, the playout of video contents has become part of a common action in our lives. Anywhere, anytime. A big ecosystem of user devices exists. This intensive usage, ubiquity, user expectations of high-quality streaming, continuous playout and better user experience challenges the network dimensioning and protocols usage to fit those requirements. This project presents a solution to reduce network usage and improve user experience in the playout of video contents.

The objective of the project is to propose a video-on-demand scenario using MPEG-DASH over an IPTV network enhanced with multicasting of media segments to Cache servers distributed within the managed network. MPEG-DASH adapts perfect to this scenario and improve user experience and reduce the requirements needed for dimensioning the network. Using this solution is transparent for the final user as they continue receiving a unicast stream of MPEG-DASH.

To meet this objective, the approach taken is to first analyze the state of the art in IPTV networks, focus in the network design and the multicast and unicast protocols used in video streaming. Then several open-source projects are evaluated to build the practical scenario. At that time, with all these elements a simulated IPTV network is created. Finally, the multicast groups are defined, and the multicast components designed, developed and deployed in Origin and Cache servers. The proposed scenario is a running example.

Using multicast delivery to network elements closer to end users enables fast delivery and reduces bandwidth usage within the network. Multicast protocols in the distribution and aggregation networks of an IPTV service, along with adaptive bitrate streaming unicast protocol (MPEG-DASH) provide a solution to face with future challenges in video streaming.

General Index

Resumen i
Abstract iii
General Index v
Table of Figures vii
Acronyms ix
1 Introduction 1
2 State of the art in IPTV Video Streaming 3
2.1 IPTV Video Delivery Networks 6
2.1.1 IPTV Backbone Technologies 7
2.1.2 Network Architecture in IPTV 9
2.1.3 Network Factors Associated with Deploying IPTV 12
2.2 Media Streaming 13
2.2.1 Broadcast over an IPTV Network using Multicast 13
2.2.2 Unicast of Video-on-Demand over IP delivery networks 17
2.2.3 Dynamic Adaptive Streaming over HTTP: MPEG-DASH 19
3 Open-source Software in the Context of Video Streaming 21
3.1 OSS Building Blocks for Video Streaming 21
3.1.1 Content Processing: Video Coding Tools 21
3.1.2 Delivery and Caching 27
3.1.3 DASH Clients 28
3.1.4 JGroups: Reliable Messaging 30
3.2 Tools for Configuration Management and Testing 34
3.2.1 Virtual Networks over LinuX (VNX) 35
3.2.2 Ansible 35
4 Use Case 37

4.1 Multicast MPEG-DASH in Managed Networks 37
4.2 Building the scenario 38
4.2.1 The Video Delivery Network (VNX scenario) 38
4.2.2 Video Content Processing component 42
4.2.3 The Origin Server 49
4.2.4 The Mid-Tier and Edge-Tier Cache Servers 51
4.3 Detailed design of multicast components 54
4.3.1 Design of the Multicast component of the Origin server 55
4.3.2 Design of the Multicast Receiver component of the Cache Servers 59
4.4 Running the scenario 62
5 Conclusions and Future Work 65
5.1 Conclusions 65
5.2 Future Work 66
Bibliography 67

Table of Figures

Figure 1 Evolution of the number of subscribers to pay TV 4
Figure 2 Households with hired services 4
Figure 3 IPTV Value Chain 5
Figure 4 IPTV Core-Networking Infrastructure 7
Figure 5 Multi-service Network Architecture 10
Figure 6 Metropolitan Network of an IPTV service over a managed network 11
Figure 7 MPEG-DASH Deployment Architecture 19
Figure 8 Apache Traffic Control 28
Figure 9 JGroups protocol stack 34
Figure 10 Components and Protocols 37
Figure 11 The Video Delivery Network 39
Figure 12 VNX scenario 42
Figure 13 Video content processing component 43
Figure 14 GOP and I, B and P frames 46
Figure 15 MPEG-DASH segmenting 48
Figure 16 The Origin Server 50
Figure 17 Mid/Edge-Tier Cache Servers 52
Figure 18 Multicast solution in MPEG-DASH video delivery 54
Figure 19 Origin Server design 55
Figure 20 JGroups in Multicast components 58
Figure 21 Fragmentation in the Quality Representation Channel 58
Figure 22 MediaSegment class 60
Figure 23 Multicast MPEG-DASH media segments 61

Acronyms

ATM: Asynchronous Transfer Mode

AVC: Advanced Video Coding

CDN: Content Delivery Network

CNMC: Comisión Nacional de los Mercados y la Competencia

CO: Central Office

DASH: Dynamic Adaptive Streaming over HTTP

DRM: Digital Rights Management

DVB: Digital Video Broadcasting

DVR: Digital Video Recorder

EPG: Electronic Programming Guide

EVC: Ethernet Virtual Connections

GMS: Group Membership

GOP: Group of Pictures

HD TV: High-definition Television

HEVC: High Efficiency Video Coding

HFC: Hybrid Fiber-Coaxial

IGMP: Internet Group Management Protocol

HTTP: Hypertext Transfer Protocol

IPTV: Internet Protocol Television

iTV: Interactive TV

IP: Internet Protocol

JSON: JavaScript Object Notation

LIB: Label Information Base

LSR: Label Switch Router

LSP: Label Switched Path

MEF: Metro Ethernet Forum

MPD: Media Presentation Description

MPEG: Moving Picture Experts Group

MPLS: Multiprotocol Label Switching

NGN: Next Generation Network

nPVR: Network Personal Video Recording

ONT: Optical Network Terminal

OSS: Open-Source Software

QoS: Quality of Service

PIM: Protocol Independent Multicast

RFC: Request for Comments

RTP: Real Time Protocol

RTSP: Real Time Streaming Protocol

SDH: Synchronous Optical Networking

STB: Set-Top Box

STM: Synchronous Transport Module

TCP: Transmission Control Protocol

TDM: Time Division Multiplexing

UDP: User Datagram Protocol

UHD: Ultra High Definition

VNX: Virtual Networks over linuX

VoD: Video on Demand

YAML: YAML Ain't Markup Language

1 Introduction

Video and multimedia applications are major drivers for the next generation of video services and network design. IPTV networks emerge as a solution to improve the user experience when consuming video contents connected to the network, everywhere, both at home and on the road. The network convergence is a reality. The new challenges that the users expect from the service provider, and also for the network provider, requires enhancements to current state of the art in video streaming. Nowadays multicast is used to broadcast live HD TV channels to a wide audience and unicast to provide video-on-demand contents and innovative services as network personal video recording (nPVR). Recent new 4K resolution (and 8K in the near future) requires a strict consistently bandwidth requirements (speeds of around 25 Mbps). Additionally, users expect QoS to be improved, and the network needs more scalability, resiliency, capabilities, and capacity.

Multicast is a proven protocol used for broadcasting TV channels and unicast protocols are using efficiently in current networks. Adaptive bitrate streaming protocols, like MPEG-DASH, enhance current user experience adapting to network changes and to the big ecosystem of user devices available.

This project proposes an enhanced solution for video delivery on IPTV networks, using MPEG-DASH as streaming protocol, adding multicast components to bring media segments closer to end users based on some metrics coming from user feedback. This enables fast delivery and reduces the bandwidth used within the network. So, both multicast and unicast delivery is joined together to enrich the video delivery with the advantages of both worlds.

This general objective is put in practice a network and software design of the solution, and an implementation to demonstrate its feasibility. The IPTV network is design and simulated with the tool VNX developed inside the ETSIT.

The remainder of the document is structured starting with the state of the art in IPTV video streaming. This section briefly presents the current market in IPTV, the design and architecture the network, and some notes about network dimensioning. Multicast and unicast media streaming protocols are described, and the section ends with a summary of Dynamic Adaptive Streaming over HTTP (DASH).

Next section covers open-source software that could be used to build a practical scenario for video streaming in an IPTV network. The section is divided in OSS building

blocks for video streaming (i.e. content processing, delivery and caching, DASH clients and communications library) and tools for configuration management and testing.

Section 4 is dedicated to the practical use case. This is the main part of the project where the proposed solution is described in detail. All the elements that build the scenario (i.e. the network, Origin server, Mid/Edge-Tier Cache servers, the multicast components and groups) are analyzed. Some design details are included in this section and the specific elements implemented for project

The document ends with some conclusions and future work.

2 State of the art in IPTV Video Streaming

Internet protocol television (IPTV) is a collection of technologies in computing, networking, and storage combined together to deliver a rich set of services and high-quality television content through Internet protocol (IP) networks [1]. IPTV is becoming a platform that is changing the way we access information and entertainment and is currently seen as a part of the triple play and quadruple play bundles that are typically offered from network operators worldwide. This IPTV service is provided through the managed network of the network operator. IPTV experience is being continuously enhanced with improvements in the underlying networks and computing systems.

Unlike IPTV, Internet TV (aka Internet video or unmanaged IPTV) is any video delivered over the public Internet to personal computers and some dedicated boxes while relying on the same core base of IP technologies. The Internet TV approach is different from the classical IPTV approach in that it leverages the public Internet to deliver video for end users without using secure dedicated managed private networks as IPTV. Consequently, Internet TV cannot ensure a guarantee for the TV quality or experience since some of the IP packets may be delayed or lost since best-effort delivery is used by the Internet. However, Internet TV has no geographical limitations as long as Internet connections exist.

The adoption of broadband Internet access by many households in turn has become a very powerful motivation for consumers to start subscribing to IPTV services. People's homes and lifestyles are evolving, adopting a range of new technologies. Digital entertainment devices such as gaming consoles, multiroom audio systems, set-top boxes (STB), and the migration of high-definition (HD) TV to 4K UHD TV has increased consumers demand on entertainment and multimedia services, all allowing for continuous evolution of home networks. Finally, some businesses and commercial drivers are also playing an important role in the evolution of IPTV technology. The increased competition is forcing many telecommunication companies to offer IPTV services to their subscribers and to extend these service by including broadband Internet connections and digital telephony forming what is known as the triple-play bundle (that is currently experience a high penetration rate through billion of subscribers worldwide) as well as the quadruple-play bundle through the addition of wireless Internet connectivity.

Spain's market authority, *Comisión Nacional de los Mercados y la Competencia (CNMC)*, produces a report about the Telecommunications and Audiovisual Markets [2]. The report presented (figures until 2017) that there was a continuous growth in the number

of pay TV subscribers with an increase of 8.7% and 534 thousand net additions, reaching a new maximum with 6.7 million households subscribed to the pay TV service (Figure 1). The highest growth was obtained by IPTV services, which with 575+ thousand net additions, closed the year with 4 million subscribers.

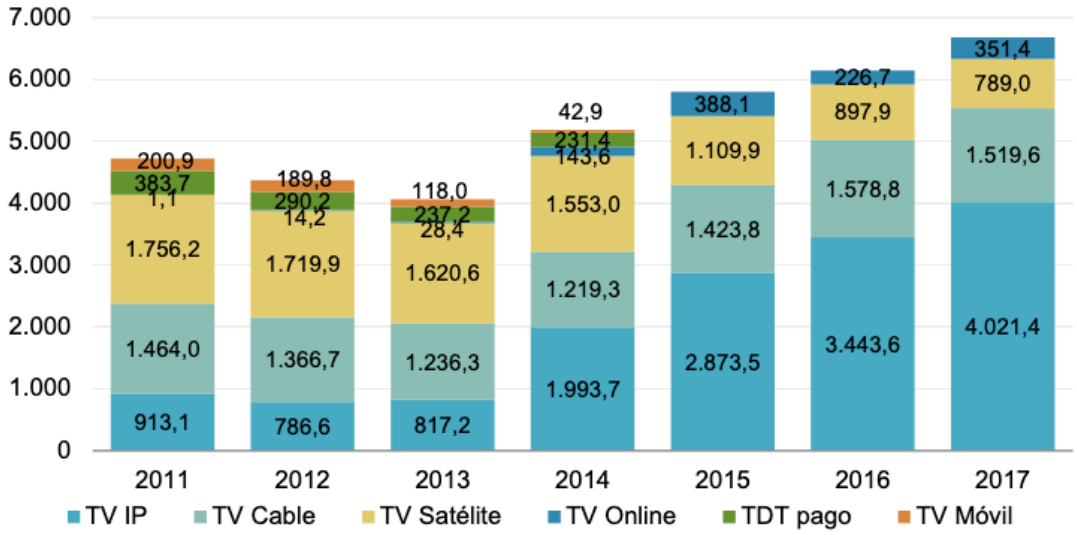

Figure 1 Evolution of the number of subscribers to pay TV

Note that the penetration of the pay TV is less (31,1%) that those that have fixed telephone, mobile and Internet (i.e. 48%).

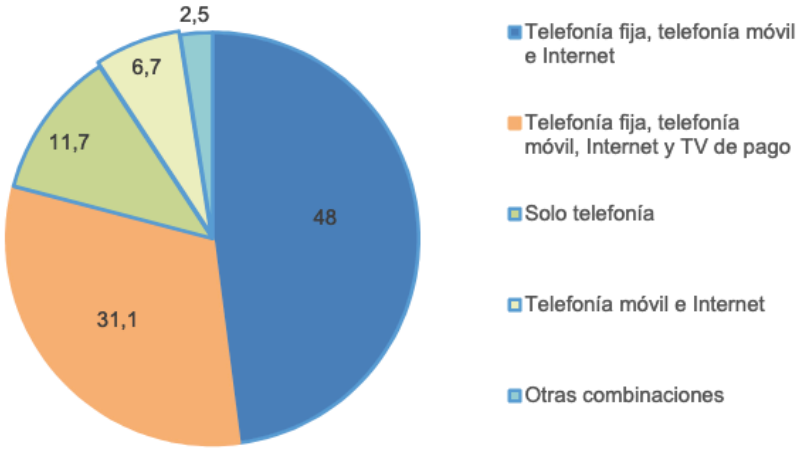

Figure 2 Households with hired services

So, IPTV market seems to have a brilliant future, that can be powered with enhance video quality (4K video resolution), new services for final users (some of them currently available but many more to come) and prices could go down due to hard competition between different providers. Basic bundles could be at a competitive price and premium bundles with innovative services could have additional subscriptions.

The major challenges that emerging video services requests are higher scalability, higher capacity, higher quality of service (QoS), stronger interactivity, dealing with heterogeneity, security, …

The IPTV value chain spans over four domains or major players:

- The *customer domain* presenting services to the end user
- The *network provider domain* allowing the connection between the customer domain and the service provider domain.
- The *service provider domain* which is responsible for providing customers with the streaming service.
- The *content provider domain* that owns or is licensed to sell contents.

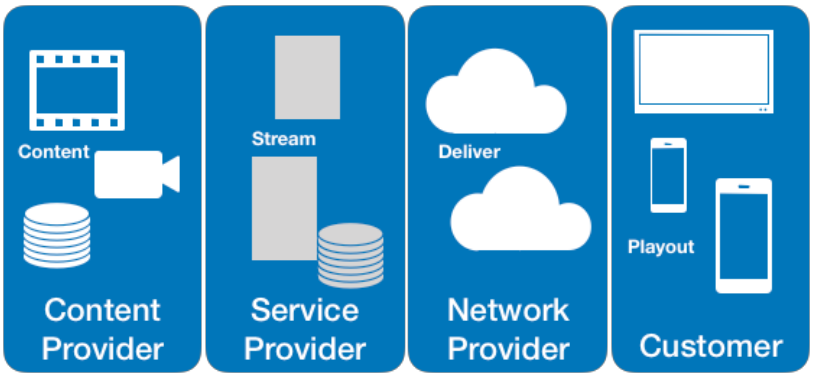

Figure 3 IPTV Value Chain

In the IPTV value chain, the telecommunications operators used to be in the two central boxes of Figure 3. But the evolution of the market makes that those operators span to also the edge of this chain, i.e. to content providers (i.e. creating video contents as films, series…) and providing to customers specific value-added devices (e.g. set-top boxes, video conference sets, home bases, smart speakers…).

In the following subsections, it is described some of the key aspects of video delivery over managed networks, it cannot be considered a complete reference as only are presented those aspects that are applied in the use cases of this project:

- *IPTV video delivery networks*, describing video delivery over managed networks.
- *Media Streaming*, video is broadcasting using different networks and protocols, in this section it will be described broadcast in IPTV networks with multicast protocols and an introduction to adaptive streaming MPEG-DASH.

2.1 IPTV Video Delivery Networks

IPTV networks are deployed over copper, fiber, wireless, and HFC networks to deliver digital TV services over IP networks, including real-time broadcast TV services. IPTV leverages IP technologies to deliver an enhanced TV experience in the context of Triple Play (video, voice and data) services:

- *Whatever*: diversity of content and customization of experience.
- *Wherever*: in a multiscreen environment, content can be played out in the TV set, in mobility with a smartphone or tablet.
- *Whenever*: with content on demand (VoD) and personal/operator recordings (recordings stored in the network done by the operator, i.e. catch-up TV, or by the user specifically recording a program), user can watch what they want at any moment independently of when it was broadcasted.

IPTV supports an everyday increasing set of features, here is some of them:

- *Digital broadcast channels*, traditional TV channels broadcasting a scheduled defined live programming.
- *Video-on-Demand (VoD)*, contents available to the user as requested, access provided by a subscription or by purchasing an individual content.
- *Digital Video Recorder (DVR, PVR)*, programming broadcasted in TV channels can be recorded by the user or the operator. Network based Personal Video Recording (*nPVR*) is a feature that allows a user to record a program in the operator's network sharing a single copy for all the users of the service.
- *Interactive TV applications (iTV)*, the television should be interactive allowing the client comment, recommend, or even buy an on-screen product.
- *Electronic Programming Guide (EPG)*, information of the programming broadcasted in a channel. This metainformation can be broadcasted to the final user using multicast protocols.
- *Targeted or Advanced Advertising*, presenting to every single user the ads most appropriate to their profile.

Many separate areas of IPTV systems are covered by standards, from networking to video encoding, in an effort to bring interoperable IPTV systems to the market.

A typical IPTV infrastructure consists of three major building blocks, content acquisition, content distribution, and content consumption, constructed in a hierarchy of national, regional, and local coverage, to customer premises.

The following subsections discuss first the backbone technologies used in an IPTV deployment, to continue with the network architecture of a regional hierarchy (as the elements and protocols used are more concrete). Finally, a discussion of the network factors that need to be considered in a commercial IPTV service (e.g. dimensioning, reliability, QoS…) is analyzed.

2.1.1 IPTV Backbone Technologies

The backbone or core of an IPTV networking infrastructure is required to carry large volumes of video content at high speed between the IPTV data center and the last mile broadband distribution network. There are several different types of backbone transmission standards that provide multipath and link protection capabilities that are necessary to ensure high reliability capabilities. Each standard has a number of specific features including data transfer speed and scalability. Three of the major backbone transmission technologies used in IPTV network infrastructures are ATM over SDH, IP over MPLS and metro ethernet. As illustrated in Figure 4 these core networking technologies provide connectivity between the central IPTV data center (head-end) and the access networks.

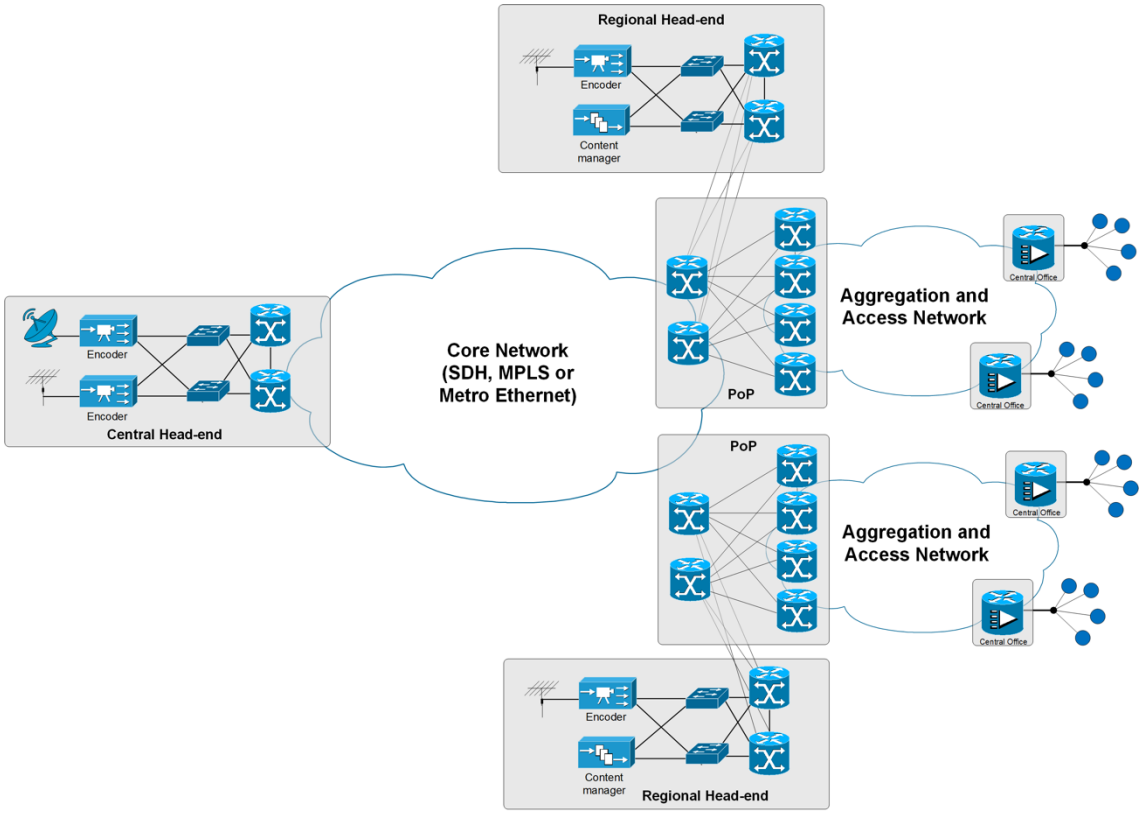

Figure 4 IPTV Core-Networking Infrastructure

ATM and SDH

ATM can support demanding applications such as IPTV that require high bandwidth and low transmission delays. ATM will operate over different network media including coaxial and twisted pair cables; however, it runs at its optimum speed over fiber optic cable. A physical layer called Synchronous Digital Hierarchy (SDH) is typically used by a number of telecommunication carriers to transport ATM cells over the backbone network.

SDH is a protocol that provides high speed transmission using fiber optic media. The available transmission rates go from 155 Mbps (STM-1) to 40 Gbps (STM-156).

SDH uses time division multiplexing (TDM) to send multiple data streams simultaneously. With TDM, the SDH network allocates bandwidth to a certain portion of time on a specific frequency. The preassigned time slots are active regardless of whether there is data to transmit.

In the context of and IPTV networking environment the SDH equipment receives a number of bits streams and combines into a single stream, which then is sent out onto the fiber network using a light emitting device. The rates of the combined input rates will equal the rate outputted from the SDH device.

IP and MPLS

A number of larger telecommunication companies have started to deploy the Internet Protocol in their core networks. Although IP was never originally designed with features such as QoS and traffic segregation capabilities, the protocol works quite well in these environments when combined with a technology called *Multiprotocol Label Switching* (*MPLS*). An MPLS enabled network supports the efficient delivery of various video traffic types over a common networking platform.

An MPLS platform is design and built using advanced *Label Switch Routers* (*LSRs*). These *LSRs* are responsible for establishing connection-oriented paths to specific destinations on the IPTV network. These virtual paths are called *Label Switched Paths* (*LSPs*) and are configured with enough resources to ensure the smooth transition of IPTV traffic through an MPLS network. The use of *LSPs* simplifies and speeds up the routing of packets through the network because deep packet inspection only occurs at the ingress to the network and is not required at each router hop.

The other main function of *LSRs* is to identify network traffic types. This is achieved by adding a MPLS header onto the beginning of each IPTV packet. This header is added at the ingress LSR and removed by the egress LSR as it leaves the MPLS core network.

While the IPTV traffic traverses across MPLS enabled routers a number of local tables called Label Information Bases (LIBs) are consulted to determine details about the next

hop along the route. In addition to examining the table, a new label is applied to the packet and forwarded onward to the appropriate router output port. Other added benefits of MPLS networks include their support for high levels of resilience when a failure occurs.

Metro Ethernet

Another technology, which may be deployed in the core network, is Metro Ethernet. An alliance of leading service providers, equipment vendors, and other networking companies called the Metro Ethernet Forum (MEF) is responsible for establishing specifications for integrating Ethernet technologies into high capacity backbone and core networks. The key technical and operational characteristics of Metro Ethernet based core networks include:

- It meets the various requirements that are typical of a core networking technology, namely, resilience, high performance, and scalability.
- Some of the modern Metro Ethernet networking components can operate at speeds up to 100 Gbps across long geographical distances. This provides service providers with an ideal platform for efficiently delivering new value-added services such as IPTV to geographically dispersed regional offices.
- It implements a sophisticated recovery mechanism in the event of a network link failure thereby, ensuring that services such as IPTV are unaffected by the outage.
- Metro Ethernet technologies support the use of connection oriented virtual circuits that allow IPTV service providers to guarantee the delivery of high-quality video content within the network core. These dedicated links are called Ethernet Virtual Connections (EVCs).

In addition to the above characteristics, the low delay and packet loss features of Metro Ethernet make it an ideal core networking technology for carrying IPTV services.

2.1.2 Network Architecture in IPTV

In Figure 5 is depicted a high-level network architecture of an IPTV service deployed on a managed network of an operator offering a triple play service. The IPTV network elements (i.e. head-end, distribution/aggregation networks, access network and home network) plays different major functions.

In the *head-end* is located the processing and aggregation of video content including video servers broadcasting H.264/H.265 encoding. The head-end or IPTV central data center receives content from a variety of sources including local video, content aggregators, content producers, cable, terrestrial and satellite channels. Once received, a number of different hardware components ranging for encoders and video servers to IP

routers and dedicated security hardware are used to prepare the video content for delivery over an IP based network. Basically head-end collects video content from different sources and prepares it for delivery over the video network. Additionally, a subscriber management system is required to manage IPTV subscriber profiles and payments. The physical location of the IPTV data center will be dictated by the networking infrastructure used by the service provider.

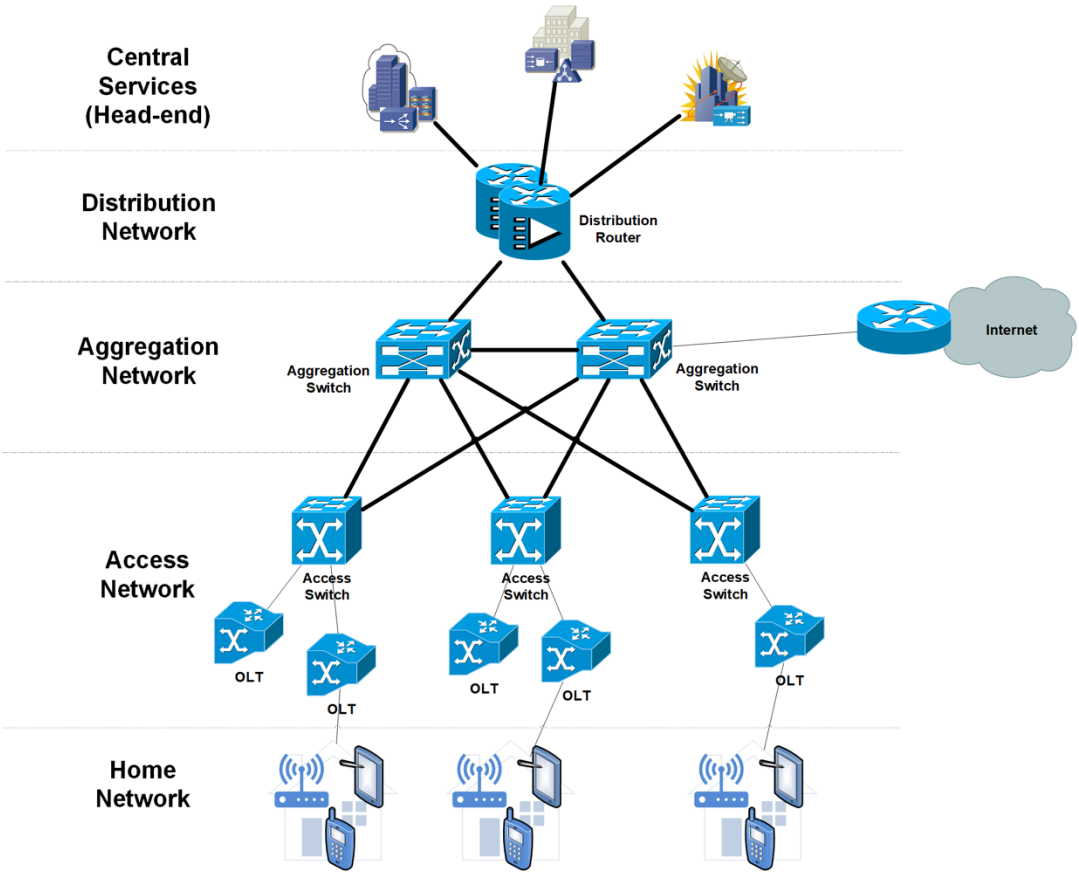

Figure 5 Multi-service Network Architecture

The *access network* provides the last mile delivery from Central Offices (CO) to users located at home network. The key attributes for this network are cost-effective, scalable, and field-proven solutions. The access network leverage installed equipment and infrastructure.

A *home network* connects a number of digital (multiscreen) devices within a small geographical area. For video streaming, a home network includes a residential ONT and router with an optional set-top box supporting MPEG-2/MPEG-4 for HD and 4K videos.

In Figure 6, network architecture is depicted in more detail focusing in a regional zone of the IPTV hierarchy. The major components of a typical IPTV architecture include the following [1]:

- *Acquisition servers* (*A-servers*) encode video and add DRM metadata, located in the same site of VoD and IPTV elements.
- *Distribution servers* (*D-servers*) provide caching and QoS control, located in central offices of the distribution, aggregation and access networks.
- *VoD servers* retain a library of encoded VoD content to provide VoD services.
- *IP routers and switches* route IP packets and provide fast reroute in case of routing failures.
- *Residential equipment* bundles services at home, including ONT, router and set-top box (STB). A STB is a device that displays output to a television set that is connected to the access network of the IPTV service.

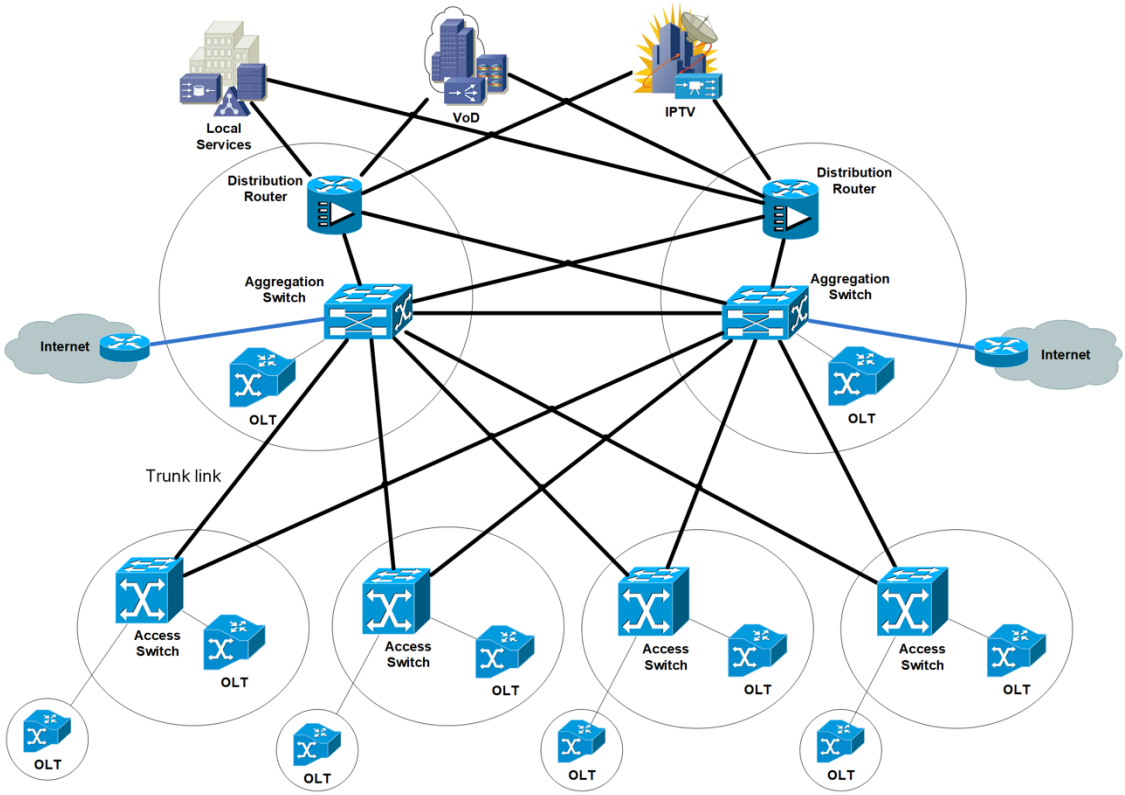

Figure 6 Metropolitan Network of an IPTV service over a managed network

This network architecture is going to be created in a simulated environment to broadcast content using MPEG-DASH adding a multicast component to improve network usage, this will be described later in this document.

2.1.3 Network Factors Associated with Deploying IPTV

A number of network factors need to be considered before commercially launching IPTV services.

Network Dimensioning

To support the transport of video, IPTV distribution networks need to feature high bandwidth carrying capacities. The total bandwidth required to implement IPTV services depends on a couple of factors:

- *The number of IPTV multicast channels on offer*. A single copy of each channel is sent from the IPTV data center on to the distribution network. Once the channel is streamed onto the networking infrastructure the multicast process handles the copying of channels and routing to individual IPTV subscribers. Consider a service provider who is offering its 3 million subscriber a package of 100 HD IP broadcast TV channels. Assumed that the provider is using H.264 to compress the channels, this generally translates to a bandwidth requirement of at least 4 Mbps for each broadcast channel. In the scenario where at least one subscriber is accessing each channel at a particular instance in time then the Next Generation Network (NGN) core distribution network will require 400 Mbps of bandwidth capacity. This is the bandwidth requirement from the IPTV central data center to each of the regional offices. At these central offices techniques such as IGMP snooping can be used to reduce the bandwidth required over the local access section of the network.
- *Inclusion of VoD service*. The dimensioning of the network is further complicated by the addition of VoD service. This type of service uses the unicast transport protocol to stream the content between the IPTV consumer devices and the on-demand video server. This mode of operation consumes a large amount of bandwidth and the network needs to accommodate this level of network traffic. Considering the same network of 3 million end users and assume that at a particular instance in time there is 5% of the subscriber base accessing VoD contents. And assuming again that the H.264 compression standard is used, this translates to a peak usage on the network of 600 Gbps (3 mill x 5% x 4 Mbps). This is a significant requirement for the core network.

Reliability

The IP networking infrastructure needs to be reliable in the event of device failures. There should be no single point of failure that could interrupt the delivery of IPTV services, both multicast or unicast streaming. Redundant links should be used wherever possible.

Fast Responsiveness

The network needs to support minimum response times associated with channel zapping (refers to changing from one channel to another during a TV viewing experience).

Predictable Performance

The nature of video bit rate streams is variable due to the differing scene complexities, which are delivered to an IPTV access device on a frame-by-frame basis. Therefore, it is difficult to predict the exact requirements of a video transmission until the service is operating in real-time. IPTV operators have to bear this in mind and assign appropriate network resources to cope with variable bit rate streams.

Level of QoS

Due to the fact that most IPTV services operate over a private managed IP network, it is advisable to implement a QoS policy when delivering video content to paying subscribers. A QoS system preserves a video signal and reduce the probability of impairments as it gets transmitted over long distances. It allows operators to provide services that require strict performance guarantees such as VoD and IP multicast. It comprises of a number of network techniques and supporting protocols that guarantee IPTV subscribers a specific level of viewing quality.

2.2 Media Streaming

In this section media streaming is described focusing in an IPTV service over a managed network. Multicast and unicast protocols are used for broadcasting TV channels and video-on demand respectively. Then an emerging protocol for adaptive streaming over HTTP is discussed as it is the proposed solution to deploy in an IPTV network along with multicast replication in the practical use case.

2.2.1 Broadcast over an IPTV Network using Multicast

Multicast refers to the technique of transmitting a single video signal simultaneously to multiple end users. All viewers receive the same signal at the same time but there are no separate streams for each recipient. It provides an efficient way to support high bandwidth, one-to-many video streams on a network.

Groups and membership form the basis of how multicasting operates. In the context of an IPTV deployment, each multicast group is a broadcast TV channel and its members equate to the various user's devices that are tuned into and viewing that channel. Thus, each IPTV channel is only streamed to the IP set-top boxes that want to view the channel. This keeps bandwidth consumption relatively low and reduces the processing burden on the server to a small fraction of that found under the unicast one-to-one communication system.

In Figure 6, only a single copy is sent from the video server to the distribution router. This router makes two copies of the stream and sends them to the regional central offices via dedicated IP connections. This approach significantly cuts down the number of IP connections and video streams traversing the network. This is typically used by service providers to broadcast live IPTV programming and is an efficient technique for utilizing an existing IP infrastructure. Multicast does not utilize an upstream path for communication between STBs and the broadcast server. The multicast of IPTV content is considered to be more sophisticated when compared to using the unicast or broadcast communication models.

IP multicast is also widely used in the delivery of broadcast TV services over IP networks. There are a number of reasons for this. First and foremost, it significantly reduces the amount of bandwidth required to transmit high quality IPTV content across a network. This is because only a single copy of every video stream needs to be sent for the requesting devices. Not only does multicast reduce the bandwidth requirements of the network but the processing power of the content server can also be kept relatively low because it only transmits one copy of an IPTV stream at a time. In contrast, a unicasting based networking environment is required to simultaneously support the transmission of multiple video streams to multiple end users and only high-performance servers can done this task. Multicasting does however have some drawbacks including:

- *Trick mode is not supported*. Multicast does not allow subscribers to rewind, pause or fast-forward the video content.
- *Limited flexibility*. When IPTV subscribers turn on their TVs, they can only join the viewing when the particular channel is already in progress.
- *Routers need to support multicast*. Service providers who want to deliver live IP broadcast television over their networking infrastructure need to ensure that all routers between the central data center and client devices are multicast enabled.
- *Increases the workload and processing requirements of routers*. Routers play a prominent role in the transmission of IPTV content across the network. In addition to forwarding traffic to the correct output ports, routers also need to handle additional tasks such as replicating video streams and keeping track of multiple copies of video packets. Processing the various tasks associated with IP multicasting adds a significant burden to the workload of IP routers.
- *Support needs to be consistent between source and destination*. All of the components between the IPTV content source and the client device are required to support IP multicasting technology. This is problematic for Internet TV providers who have limited or no control on how devices are configured on the public Internet.

- *Blocking of IP multicast traffic*. Security devices such as firewalls are often configured by IT network managers to block IP multicast protocols. This is not a major issue for operators that own the networking infrastructures; but it should be kept in mind when deploying IPTV services across the public Internet.

The deployment of a multicast system is based on a distributed networking architecture. The logical and physical components needed to deliver IP multicast services can be categorized as:

- IGMP devices
- Multicast groups and addressing
- IPTV multicast protocols
- Multicast transport architecture

IGMP devices

A multicast-enabled host is configured to send and/or receive multicast data. There are two categories of devices used in an end-to-end IGMP communications transaction:

- An *IGMP host* is any client or server connected to an IPTV network.
- *Multicast routers* also called IGMP routers are a key component of an IPTV networking infrastructure. Routers on an IPTV network fall into two broad categories, distribution and aggregation routers (could be also be deployed as switches). The distribution router(s) directly interface with the source content servers. All of the IPTV channels are available at the distribution router. The aggregation router is positioned downstream in the network and closer to the end user. Only the channels that are viewed by client devices connected to the aggregation router are available at this point on the network. Multicast IP routers are an integral and essential component used in the delivery of IPTV broadcast channels. They are connected to the IPTV transmission network and support the functions of receiving multicast IPTV content, managing and processing IGMP messages, keeping the routing tables updated and replication of IPTV streams.

IPTV Multicast Protocol: Internet Group Management Protocol (IGMP)

The IGMP protocol version 3 RFC 3376 (updated by RFC 4604) operates between a host and its directly attached router (or first-hop router). IGMP provides the means for a host to inform its attached router that an application running on the host wants to join a specific multicast group. Given that the scope of IGMP interaction is limited to a host and its attached router, another protocol is required to coordinate the multicast routers

throughout the network, so that multicast datagrams are routed to their final destinations. This latter functionality is accomplished by network-layer multicast routing algorithms. Network layer multicast in the internet consists of two complementary components: IGMP and multicasting routing protocols.

There are several types of IGMP messages, carried (encapsulated) within an IP datagram, with an IP protocol number of 2:

- The *membership_query* message is sent by a router to all hosts on an attached interface to determine the set of all multicast groups that have been joined by the hosts on that interface.
- The *membership_report* message is the respond by the hosts to the membership_query sent by a router. This message can also be generated by a host when an application first joins a multicast group without waiting for a *membership_query* message from the router.
- The *leave_group* optional message, that when explicitly sent by a host indicates the attached router that the host leaves the multicast group. When not explicitly sent the host, the router infers that a host is no longer in the multicast group if it no longer responds to a membership_query message with the given group address. This is an example of what is called soft state in an Internet protocol.

Multicast Routing Protocols

The delivery of video over an IP networking infrastructure generally uses a number of routing protocols and technologies. These routing technologies are multicast distribution trees, multicast distribution protocols and multicast forwarding techniques. As the practical scenario uses PIM as multicast distribution protocol a brief description follows.

In large IPTV deployments, multiple multicast routers are required to deliver multiple TV broadcast channels to a spatially diverse and large number of client devices. Protocols such as *Protocol Independent Multicast* (*PIM*) are used to build multicast distribution trees that route IPTV video content through a high-speed broadband network. PIM defines a small collection of multicast routing protocols that are optimized for delivering different types of services. There are four PIM variants:

- *PIM dense mode* (*PIM-DM*) operates on the principle of flooding multicast packets to all routers on the network. Routers that do not have multicast IP group members connected to their interfaces send a leave or prune message to the source of the packets. Once the source receives the prune/leave message, the transmission of multicast packets to that part of the network ceases. This flooding of unwanted multicast traffic consumes valuable

bandwidth, that's why this PIM variant is rarely deployed across an IPTV network.

- *PIM sparse mode* (*PIM-SM*) defines how routers interact with each other to build and maintain different types of multicast distribution trees. PIM-SM is based on the assumption that viewers of a particular channel are sparsely distributed throughout an IPTV network. The delivery of TV channels under the PIM-SM protocol is done so using the pull mode content delivery mechanism and only client devices that have explicitly requested to view a channel will be forwarded the associated IP video traffic. This is a useful technique for preserving bandwidth; however, the time taken to join a broadcast stream may cause a slight delay since IGMP instructions need to be issued to the nearest upstream multicast router that is processing the requested channel. PIM-SM scales quite well for large network infrastructures.
- *PIM source specific multicast* (*PIM-SSM*) is a routing protocol that operates at level 3 and has been derived from PIM-SM. It supports the deployment of SSM, a delivery model supported by IGMPv3, which allows client devices to explicitly specify the channels they want to receive.
- *Bidirectional PIM* (*BIDIR-PIM*) is slightly different to PIM-SM. Lack of support for encapsulation and source trees are two of the main differentiators of this PIM based routing protocol. BIDIR-PIM is considered to be quite a useful protocol in cases where scalability is required. It does however suffer from instances of delays when deployed across large IP networks.

Of the four PIM variants PIM-SM is the most popular multicast distribution protocol used to support deployments of IPTV. This is the protocol used in the practical use case of this project.

2.2.2 Unicast of Video-on-Demand over IP delivery networks

In unicast, every video stream is sent to a single client device. Therefore, if more than one IPTV end user desires to receive the same video content, each client device will need a separate unicast stream. This uses a considerable network resources, even though each client is watching the same content. Each of these streams will flow to the destinations across the high-speed IP network. The principle of implementing unicast over an IP network is based on delivering a dedicated stream of content to each end user. From a technical perspective this configuration is quite easy to implement; however, it does not make effective use of the bandwidth on the network.

When multiple IPTV end users decide to access the same IPTV channel at the same time a number of dedicated IP connections are established across the network. For

example, if five IPTV subscribers access a specific IPTV broadcast channel the same number of IP connections are established, and the video server needs to provide an active IP connection to every IPTV subscriber that requires access to the channel. These connections are routed to their destination points. In this unicast environment the need for multiple IP connections leads to a requirement for very high capacity network links. This method of transporting IP video is well suited to on-demand services such as network based personal video recording (nPVR) and VoD, where each subscriber receives a unique stream.

nPVR is a technology that allows consumers to record video programming and play back at their convenience. The storage of video content and manipulation for nPVR takes place at the IPTV data center. In effect, it centralizes the functions of a DVR. Thus, users can use standard trick modes (i.e. fast-forwarding, rewinding and pausing) and recording controls.

The nPVR delivery mechanism eliminates the costs associated with supplying subscribers with hard-disk enabled STBs. nPVR does have a drawback, namely, the large amount of storage space required at the service provider's IPTV data center.

In addition to a high capacity of the broadband network, the deployment of VoD services also require a number of other logical and physical blocks: streaming server(s), IP transport protocols and an interactive client application.

Besides the various processes that are executed in IPTV data centers, a number of high capacity video servers are also deployed to allow the delivery of VoD services to various types of client devices. The main function of these VoD streaming servers is to retrieve and deliver on demand video content to a distribution network.

The size and capabilities of the video server varies, however most popular will support advanced streaming capabilities, resilience, high capacity storage capabilities, monitoring, scalability, multiple formats support, interoperability and real-time ingesting of content.

The hardware architecture of a streaming VoD server consists of technologies for storage, processing and memory, network connectivity, and software. The on-demand content gets archived by the storage subsystem and is retrieved by the processing and memory subsystem once a request for a video asset is received; the networking subsystem the packetizes the video data into IP packets and streams over the network.

RTSP (Real Time Streaming Protocol) along with RTP (Real Time Protocol) are the most prevalent protocol for streaming VoD content. RTSP enables client devices to establish and control the flow of video streams, and a separate RTP over UDP connection

is established to carry encoded video content. In this way RTSP and RTP work closely together to deliver IPTV content across the network.

2.2.3 Dynamic Adaptive Streaming over HTTP: MPEG-DASH

In the previous sections IPTV multicast to broadcast live TV channels and unicast of video-on-demand have been discussed. Moving forward in unicast delivery, a new type of HTTP-based streaming referred to as Dynamic Adaptive Streaming over HTTP (DASH) appears on the scene. In this section MPEG-DASH is presented as it is the protocol used to enhance the video streaming in the practical scenario.

In DASH, the video is encoded into several different versions, with each version having a different bit rate and, correspondingly, a different quality level. The client dynamically requests chunks of video segments of a few seconds in length from the different versions. It is the client who selects which chunks request at a given time, selecting high-rate version when the amount of available bandwidth is high. The client selects different chunks one at a time with HTTP GET request messages. Figure 7 represents the basic components involved in a scenario of streaming video using MPEG-DASH [3].

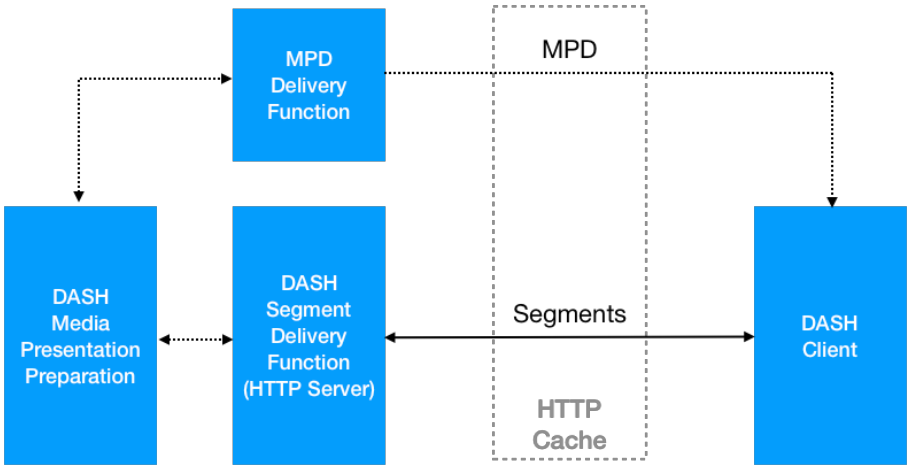

Figure 7 MPEG-DASH Deployment Architecture

With DASH, each video version is stored in the HTTP server, each with a different URL. The HTTP server also has a manifest file, which provides a URL for each version along with its bit rate. The client first requests the manifest file and learns about the various versions. The client then selects one chunk at a time by specifying a URL and a byte range in an *HTTP GET* request message for each chunk. While downloading chunks, the client also measures the received bandwidth and runs a rate determination algorithm to select the chunk to request next. Naturally, if the client has a lot of video buffered and if the measured receive bandwidth is high, it will choose a chunk from a high-rate version. And naturally if the client has little video buffered and the measured

received bandwidth is low, it will choose a chunk from a low-rate version. DASH therefore allows the client to freely switch among different quality levels. Since a sudden drop in bit rate by changing versions may result in noticeable visual quality degradation, the bit-rate reduction may be achieved using multiple intermediate versions to smoothly transition to a rate where the client's consumption rate drops below its available receive bandwidth. When the network conditions improve, the client can then later choose chunks from higher bit-rate versions.

By dynamically monitoring the available bandwidth and client buffer level, and adjusting the transmission rate with version switching, DASH can often achieve continuous playout at the best possible quality level without artifacts (e.g. frame freezing or skipping) created in the client player. Furthermore, since the client (rather than the server) maintains the intelligence to determine which chunk to send next, the scheme also improves server-side scalability. Another benefit of this approach is that the client can use the HTTP byte-range request to precisely control the amount of prefetched video that it buffers locally.

3 Open-source Software in the Context of Video Streaming

For the use case of this project several open-source projects have been analyzed and used. In this section those projects are described, describing in more detail those components that will be used in the development of the practical scenario. These OSS projects speed up the development of the scenario, providing tools and libraries well-tested and with high-quality functions that could not be done from scratch in a short time. The documentation of all of them has been widely used to deploy, configure and run all the components.

This section is split into two parts:

- *Building blocks for video streaming*, open-source projects covering different aspects of delivering video over a network.
- *Tools for configuration management and testing*, where it is presented the tools for the configuration and provision of the components that integrates the scenario as well as simulating the video network.

3.1 OSS Building Blocks for Video Streaming

The following building blocks are used for the video streaming use case:

- *Content processing*, to create the desired formats to deliver the video to the viewing devices of the end users.
- *Delivery and caching*, HTTP servers for delivering and caching the video content, e.g. MPEG-DASH files.
- *DASH clients* used for the playout by final users.
- *JGroups*, Java library for reliable messaging.

3.1.1 Content Processing: Video Coding Tools

The first step in video delivery is to generate the files (i.e. manifest, segments files…) to be streaming to the final users. This functionality is performed using the open-source software described in this section:

- *FFmpeg*
- *GPAC*

FFmpeg

FFmpeg is a free software cross-platform multimedia framework to decode, encode, transcode, mux, demux, stream, filter and play audio and video. *FFmpeg* includes the following command line tools:

- *ffmpeg* is a command line tool to convert multimedia files (audio or video) between formats. It can also capture and encode in real-time from various hardware and software sources (e.g. a TV capture card).
- *ffplay* is a simple media player based on Simple DirectMedia Layer (SDL) and the FFmpeg libraries.
- *ffprobe* is a simple multimedia stream analyzer to display media information.

The official page with *FFmpeg* documentation is located at:

https://ffmpeg.org/ffmpeg.html

The *ffmpeg* command line tool is used in this project to produce the different encoded medias, a brief discussion of the options of this tool are included here.

Main options

-i url :

Input file URL.

-c[:stream_specifier] codec
-codec[:stream_specifier] codec

Select an encoder (when used before an output file) or a decoder (when used before an input file) for one or more streams. codec is the name of a decoder/encoder or a special value copy (output only) to indicate that the stream is not to be re-encoded.

Video options

-b[:stream_specifier] bitrate

Set bitrate in bit/s. Default value is 200K.

-r[:stream_specifier] fps

Set frame rate (Hz value).

-vcodec codec

Set the video codec. This is an alias for -codec:v.

-vf filtergraph

Create the filtergraph specified by filtergraph and use it to filter the stream. This is an alias for -filter:v, see the -filter option.

-g gop

Set the group of pictures (GOP) size. Default value is 12.

Audio options

-acodec codec

Set the audio codec. This is an alias for -codec:a.

Ffmpeg filters

Using filtering with FFmpeg we can draw a test string from a specified file on top of a video. This option accepts a lot of parameters but here only present the simple use of it.

An example of use is:

```
drawtext="fontfile=FreeSerif.ttf: text='This is a test'"
```

This option draws a 'Test' with font FreeSerif using the default values for the optional parameters.

An example to generate a new media from

$ ffmpeg -i bbb_sunflower1080p_30fps_normal.mp4
-c:v libx264 -c:a copy -b:v 2M -r 24
-profile:v baseline -level 3.0 -g 24 -vf scale=1280:720 bbb_sunflower_repMM1.mp4

Input file is bbb_sunflower1080p_30fps_normal.mp4 (-i option in command).

The option codec -c:v libx264 -c:a copy encodes all video streams with libx264 and copies all audio streams.

The video bitrate of the output file is set to 2Mbit/s (-b:v 2M).

The frame rate of the video is 24 fps (-r 24).

The output video is created with the highest compatibility with ancient devices (-profile:v baseline -level 3.0).

GOP length (the distance between two consecutive I-frames) is 24 (-g 24). Filter option for video -vf scale=1280:720 resizes the video to 1280:720.

The *ffprobe* command line tool gather information from multimedia streams and prints it out, e.g. it can be used to check the format of the container used by a multimedia stream and the format and type of each media stream contained in it. *ffprobe* output is designed to be easily parsable by a textual filter and consists of one or more sections of a form defined by the selected writer, which is specified by the print_format option. In the example below the JSON format is used for printing.

Executing the ffprobe command line tool on the original *mp4* file of the previous example the following output is printed out:

```
$ ffprobe -hide_banner -print_format json bbb_sunflower_1080p_30fps_normal.mp4
{
Input #0, mov,mp4,m4a,3gp,3g2,mj2, from 'bbb_sunflower_1080p_30fps_normal.mp4':
  Metadata:
    major_brand     : isom
    minor_version   : 1
    compatible_brands: isomavc1
    creation_time   : 2013-12-16T17:44:39.000000Z
    title           : Big Buck Bunny, Sunflower version
    artist          : Blender Foundation 2008, Janus Bager Kristensen 2013
    comment         : Creative Commons Attribution 3.0 -
http://bbb3d.renderfarming.net
    genre           : Animation
    composer        : Sacha Goedegebure
  Duration: 00:10:34.53, start: 0.000000, bitrate: 3481 kb/s
    Stream #0:0(und): Video: h264 (High) (avc1 / 0x31637661), yuv420p, 1920x1080
[SAR 1:1 DAR 16:9], 2998 kb/s, 30 fps, 30 tbr, 30k tbn, 60 tbc (default)
    Metadata:
      creation_time   : 2013-12-16T17:44:39.000000Z
      handler_name    : GPAC ISO Video Handler
    Stream #0:1(und): Audio: mp3 (mp4a / 0x6134706D), 48000 Hz, stereo, s16p,
160 kb/s (default)
    Metadata:
      creation_time   : 2013-12-16T17:44:42.000000Z
      handler_name    : GPAC ISO Audio Handler
    Stream #0:2(und): Audio: ac3 (ac-3 / 0x332D6361), 48000 Hz, 5.1(side), fltp,
320 kb/s (default)
    Metadata:
      creation_time   : 2013-12-16T17:44:42.000000Z
      handler_name    : GPAC ISO Audio Handler
    Side data:
      audio service type: main

}
```

The official page with FFmpeg documentation is located at:

https://ffmpeg.org/ffmprobe.html

In the use cases, when transcoding the original file to different representations we will use this tool to check out the generated files.

GPAC

GPAC is an open-source multimedia framework used for research and academic purposes. The project covers different aspects of multimedia, with a focus on presentation technologies and on multimedia packaging formats such as MP4.

GPAC provides three sets of tools based on a core library called *libgpac*:

- A multimedia player, called *MP4Client*,
- A multimedia packager, called *MP4Box*,
- And some server tools included in *MP4Box* and *MP42TS* applications.

The two first tools are used in this project, *MP4Client* as a DASH client and *MP4Box* to create the DASH segments.

GPAC is cross-platform and it is written in ANSI C for portability reasons.

MP4Box is the multimedia packager in GPAC used for performing manipulations on multimedia files like AVI, MPG, TS, but mostly on ISO media files (e.g. MP4, 3GP).

MP4Box can be used:

- for manipulating ISO files like MP4, 3GP: adding, removing, multiplexing audio, video and presentation data (including subtitles) from different sources and in different formats,
- for encoding/decoding presentation languages like MPEG-4 XMT or W3C SVG into/from binary formats like MPEG-4 BIFS or LASeR,
- for performing encryption of streams,
- for attaching metadata to individual streams or to the whole ISO file to produce MPEG-21 compliant or hybrid MPEG-4/MPEG-21 files,
- for preparation of HTTP Adaptive Streaming content,
- and packaging and tagging the result for streaming, download and playback on different devices (e.g. phones, tablets) or for different software (e.g. iTunes).

MP4Box can be used to repackaging existing content to compliant ISO media files (MP4, 3GP, 3G2, OMA DCF), but MP4Box does not re-encode audio, video and still image content, external tools shall be used for this purpose (e.g. *ffmpeg*).

MP4Box is used to prepare files for different delivery protocols, e.g. HTTP downloading, RTP streaming or adaptive streaming (MPEG-DASH). Adaptive

streaming MPEG-DASH is used in this project as video delivery protocol, an example to create the DASH manifest and associated files is:

$ MP4Box -dash 2000 bbb_sunflower_repMM1.mp4

MP4Box can be used to generate content conformant to MPEG-DASH specification (ISO/IEC 23009-1). The common options supported by MP4Box (used in the context of the project) are:

-dash duration

Enables DASH segmentation of input files with the given segment duration.

-frag dur_in_ms

Specifies the duration of subsegments in ms. This duration should always be less than the segment duration. By default, the subsegment duration is the DASH duration, i.e. there is only one subsegment per segment.

-out filename

Specifies the output file name for MPD. All segments will be produced in the same directory as the MPD.

-profile name

Specifies the target DASH profile: OnDemand, live, main, simple, full, dashavc264:live, dashavc264:onDemand.

-segment-name name

Sets the segment name for the generated segments. If not set (default), the segments are concatenated in output file. The segment names can furthermore be configured by using a subset of the SegmentTemplate identifiers: $RepresentationID$, $Number$, $Bandwidth$ and $Time.

-segment-ext name

Sets the segment extension. Default is m4s, null means no extension.

-base-url string

Sets the base url at MPD level. Can be used several times for multiple URLs.

-single-segment

Uses a single segment for each representation. Set by default for OnDemand profile.

-single-file

Uses a single file for each representation.

A complete list of the options supported for MPEG-DASH delivery protocol in the *MP4Box* tool can be found typing in the command shell *MP4Box -h dash*.

PP4Client is described in the section 3.1.3 when discussing DASH clients.

3.1.2 Delivery and Caching

Origin servers and caching servers are used to stream video files hosted in some kind of storage to consumers over the Internet. These servers also allow content to be cached local to the viewer in order to deliver the best experience to every consumer.

There are many different open-source technologies used for streaming, including *Apache Traffic Server* (*http://trafficserver.apache.org/*), and *NGINX* (*https://www.nginx.com*) to name a few.

For the practical use case, the Origin server is deployed using *Apache HTTP Server*. Caching function is provided by *Apache Traffic Server*. Both components are projects (included in the http category) from *The Apache Software Foundation*, that at the time of writing this document were celebrating 20 years of community-led development. Moreover, the *Apache HttpComponents Client* is used to build the multicast components in the Origin and Cache servers.

The *Apache HTTP server* is a free and open-source cross-platform web server, release under the terms of Apache License 2.0. *Apache Traffic Server* (ATS) is a high-performance web proxy cache that improves network efficiency and performance by caching frequently-accessed information at the edge of the network. This brings content physically closer to end users, while enabling faster delivery and reduced bandwidth use.

Another open-source project in the context of video streaming analyzed for this project is the *Apache Traffic Control* (Figure 8). Traffic Control is a CDN control plane. It is made up of a suite of applications which are used to configure, manage, and direct client traffic to a tiered system of HTTP caching proxy servers (aka cache servers). In this suite the caching software chosen is Apache Traffic Server.

The following figure taken from the documentation of the *Apache Traffic Control* depicts the overall architecture and components included in the solution.

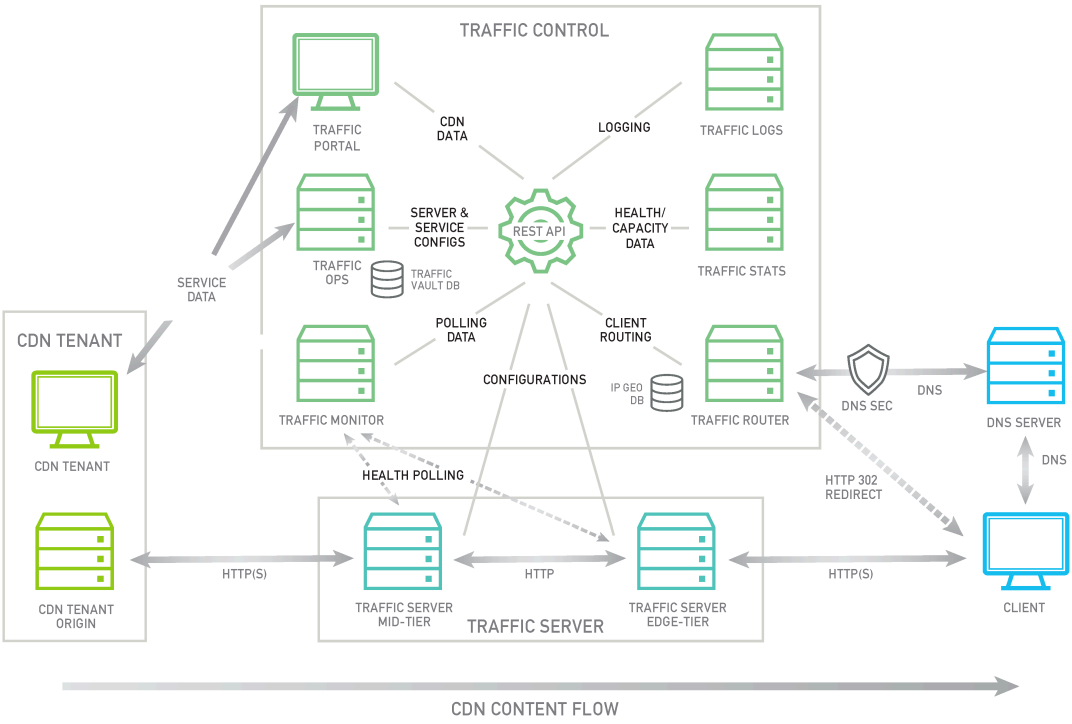

Figure 8 Apache Traffic Control

Let's describe some of the components of the Traffic Control. *Traffic Router* is used to route client requests to the closest healthy cache server by analyzing the health, capacity, and state of the cache servers according to a health protocol and relative geographic distance between each Cache Group and the client. *Traffic Monitor* does health polling of the cache servers on a very short interval to keep track of which servers should be kept in rotation. *Traffic Stats* collects and stores real-time traffic statistics aggregated from each of the cache servers. This data is used by the *Traffic Router* to assess the available capacity of each cache server which it uses to balance traffic load and prevent overload.

3.1.3 DASH Clients

The content playout by final users is done by two open-source DASH clients: *Dash.js* is a reference client implementation for the playback of MPEG-DASH via Javascript and compliant browsers, and *MP4Client* is the multimedia player included in GPAC framework.

Dash.js

Dash.js is an initiative of the DASH Industry Forum to establish a production quality framework for building video and audio players that play back MPEG-DASH content using client-side JavaScript libraries leveraging the Media Source Extensions API set as defined by the W3C.

The core objectives of this project are to build an open source JavaScript library for the playback of DASH which:

- Is robust in a real-world production environment.
- Has the best performing adaption algorithms.
- Is free for commercial use.
- Is both codec and browser agnostic.
- Implements best practices in the playback of MPEG-DASH.
- Supports a wide array of features including in-band events, multiple-periods and cross-browser DRM.

All code in the *Dash.js* project is covered by the BSD-3 license. This permissive license allows redistribution and use in source and binary forms, with or without modification, without cost or any license fees.

As an example, open your favorite Internet navigator and connect to a server where the DASH reference player is available, for example:

http://vnet.video-player.tv/dash.js-2.9.2/dash-if-reference-player/index.html

Then in the DASH player insert the MPD file in the stream text box:

http://vnet.origin1.net/video-rep/videobbb.mpd

The video starts playback and you can check out video buffer level and bitrate (kbps).

You can find additional documentation of Dash.js its Wiki page:

https://github.com/Dash-Industry-Forum/dash.js/wiki

MP4Client

GPAC provides a highly configurable multimedia player available in command-line, GUI and browser plugins. For this project the command-line option is used. *MP4Client* is capable to play most video or audio formats and support most of the existing delivery protocols, additionally it focuses on graphics, animations and interactivity technologies. The GPAC player is supported on Windows platforms, on Linux, MacOS, iOS and Android. The *MP4Client* GPAC command line player used is version 0.5.2-DEV.

When starting the *MP4Client* you can check the options available in command line hitting the key 'h'.

As an example, to playout an MPEG-DASH content, start the *MP4Client*, insert command key *o* to connect to server and *q* to stops playback:

```
$ MP4Client
Loading GPAC Terminal
Hit 'h' for help

o
Enter the absolute URL
http://vnet.origin1.net/video-repo/videobbb.mpd
Service Connected

q
```

Note that the video start playback in a separate window.

You can find additional documentation of this player in the official site of GPAC:

https://gpac.wp.imt.fr/player/

3.1.4 JGroups: Reliable Messaging

In this section we discuss JGroups as it is the Java library used to implement the multicast communication between Origin and Cache servers.

At the beginning of the project several options were taken into consideration to implement the multicast of media segments from Origin server to Mid-Tier and End-Tier Caches, e.g. different Java libraries, modules in Python and node.js,… but finally Java and JGroups was selected to implement multicast communication as it provides reliable messaging (one-to-one or one-to-many) between the systems and mainly because using JGroups prevents to implement several functions in the server and client. Time is an important factor in the project and using JGroups permits the development effort focus on main tasks.

JGroups is licensed under the Apache License 2.0. The version used of Groups is 4.0.19.

JGroups is a reliable group communication toolkit written in Java. It is based on IP multicast, but extends it with reliability and group membership. Unicast and multicast communications are supported by JGroups. This project needs multicast communication for the transmission of media segments from Origin to Cache servers.

For multicast communication, where one sender sends a message to many receivers, IP multicast extends UDP: a sender sends messages to a multicast address and the receivers have to join that multicast address to receive them. Like in UDP, message transmission is unreliable, packets may be lost, and there is no notion of membership (who has currently joined the multicast address).

The most powerful feature of JGroups is its flexible protocol stack, that can be adapted to exactly match the requirements and network characteristics of the system. JGroups

comes with a large number of protocols (additionally anyone can write their own), for example:

- Transport protocols: UDP (IP Multicast) or TCP
- Fragmentation of large messages
- Reliable unicast and multicast message transmission. Lost messages are retransmitted
- Failure detection: crashed nodes are excluded from the membership
- Flow control to prevent slow receivers to get overrun by fast senders
- Ordering protocols: FIFO, Total Order
- Membership
- Encryption
- Compression

Using a typical protocol stack configuration, we are going to discuss the properties of the different protocols. This configuration should be adapted to the requirements of the project.

```
<config xmlns="urn:org:jgroups"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="urn:org:jgroups
http://www.jgroups.org/schema/jgroups.xsd">
    <UDP
            mcast_port="${jgroups.udp.mcast_port:45588}"
            ucast_recv_buf_size="5M"
            ucast_send_buf_size="640K"
            mcast_recv_buf_size="5M"
            mcast_send_buf_size="640K"
            max_bundle_size="64K"
            ip_ttl="${jgroups.udp.ip_ttl:2}"
            enable_diagnostics="true"

            thread_pool.min_threads="2"
            thread_pool.max_threads="8"
            thread_pool.keep_alive_time="5000"/>

    <PING />
    <MERGE3 max_interval="30000"
            min_interval="10000"/>
    <FD_SOCK/>
    <FD_ALL/>
    <VERIFY_SUSPECT timeout="1500"  />
    <BARRIER />
    <pbcast.NAKACK2 use_mcast_xmit="true"
```

```
                    discard_delivered_msgs="true"/>
        <UNICAST3 />
        <pbcast.STABLE stability_delay="1000" desired_avg_gossip="50000"
                       max_bytes="4M"/>
        <pbcast.GMS print_local_addr="true" join_timeout="2000"/>
        <UFC max_credits="2M"
             min_threshold="0.4"/>
        <MFC max_credits="2M"
             min_threshold="0.4"/>
        <FRAG2 frag_size="60K"  />
        <pbcast.STATE_TRANSFER />
    </config>
```

Transport: UDP

This is the transport protocol. It uses IP multicasting to send messages to the entire cluster, or individual nodes.

Initial membership discovery: PING

This is the discovery protocol. It uses IP multicast (by default) to find initial members. Once found, the current coordinator can be determined, and a unicast JOIN request will be sent to it in order to join the cluster.

Merging after a network partition: MERGE3

If a cluster gets split for some reason (e.g. network partition), this protocol merges the sub clusters back into one cluster.

Failure Detection: FD_SOCK, FD_ALL, VERIFY_SUSPECT

The task of failure detection is to probe members of a group and see whether they are alive. When a member is suspected of having failed, then a SUSPECT message is sent to all nodes of the cluster. It is not the task of the failure detection layer to exclude a crashed member (this is done by the group membership protocol, GMS), but simply to notify everyone that a node in the cluster is suspected of having crashed.

FD_ALL is a failure detection based on simple heartbeat protocol. Every member periodically multicast a heartbeat.

FD_SOCK is a failure detection protocol based on a ring of TCP sockets created between cluster members, but not using heartbeat messages.

VERIFY_SUSPECT verifies that a suspected member is really dead by pinging that member one last time before excluding it and dropping the suspect message if the member does respond.

Reliable Message Transmission: NAKACK2, UNICAST3

NAKACK2 provides reliable delivery and FIFO properties for messages sent to all nodes in a cluster. Message reliability guarantees that a message will be received. If not, the receiver(s) will request retransmission. FIFO guarantees that all messages from sender P will be received in the order P sent them.

UNICAST3 provides reliable delivery and FIFO properties for point-to-point messages between a sender and a receiver, i.e. same as NAKACK2 for unicast messages.

Message stability: STABLE

Deletes messages that have been seen by all members (distributed message garbage collection).

Group Membership: GMS

Group membership takes care of joining new members, handling leave requests by existing members, and handling SUSPECT messages for crashed members, as emitted by failure detection protocols.

Flow Control: MFC and UFC

Flow control takes care of adjusting the rate of a message sender to the rate of the slowest receiver over time. This is implemented through a credit-based system, where each sender has *max_credits* credits and decrements them whenever a message is sent. The sender blocks when the credits fall below 0, and only resumes sending messages when it receives a replenishment message from the receivers.

Fragmentation: FRAG2

Fragments large messages into smaller ones and reassembles them back at the receiver side for both multicast and unicast messages.

State Transfer: STATE_TRANSFER, BARRIER

STATE_TRANSFER ensures that state is correctly transferred from an existing member (usually the coordinator) to a new member.

BARRIER is used by some of the state transfer protocols, as it lets existing threads complete and blocks new threads to get both the digest and state in one go.

We can depict protocol stack in the following diagram:

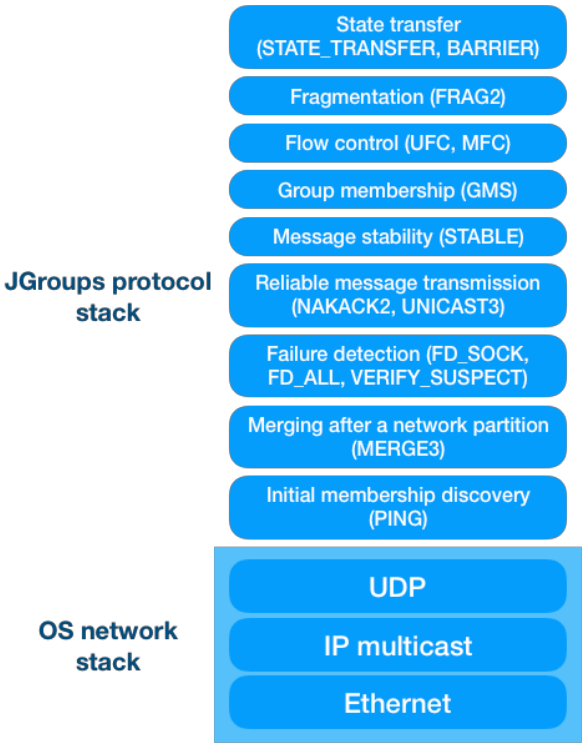

Figure 9 JGroups protocol stack

As an example, a user might start with a stack only containing IP Multicast. To add loss-less transmission, he might add the NACKACK protocol (which also eliminates duplicates). Now the messages sent from a sender are always received by the recipients, but the order in which they will be received is undefined. Therefore, the user might choose to add the FIFO layer to impose per/sender ordering. If ordering should be imposed over all the members, then the SEQUENCER protocol (providing total order) may be added. The group membership (GMS) protocol provides group membership: it allows the user to register a callback that will be invoked whenever the membership changes, e.g. a member joins, leaves or crashes. In the latter case, a failure detector protocol is needed by the GMS to announce crashed members. If new members want to obtain the current state of existing members, then the STATE_TRANSFER protocol has to be present in this custom-made stack. Finally, the user may want to encrypt messages, so he adds the ENCRYPT protocol.

All the information above is an excerpt of JGroups Manual located at:

http://www.jgroups.org/manual4/index.html

3.2 Tools for Configuration Management and Testing

In this section we describe two helpful tools used to test the overall scenario (simulating all the elements that compose the use case) and configure all the components:

- *Virtual Networks over LinuX (VNX)*, used to create the simulated environment of a video delivery network deployed in a managed network of an IPTV operator.
- *Ansible*, open-source project for provisioning, configuration and application deployment.

3.2.1 Virtual Networks over LinuX (VNX)

VNX is a tool to create a testbed environment with networks and systems simulating a real environment running in commodity hardware. This simulated environment can be used to test and evaluate network configurations, protocols development and system implementations. This tool has been developed by the Telematics Engineering Department (DIT) of the Technical University of Madrid (UPM). VNX is distributed under the GNU General Public License (GNU GPL) which guarantees users the freedom to run, study, share and modify the software.

This tool is based in virtualization techniques available in Linux operating system and allows running several virtual machines or containers on a host. This combined with the use of emulated virtual networks in the host can create scenarios of virtual networks even with external connections. The number of virtual machines that can be managed by a host is limited by the resources available and used by the scenario, e.g. network traffic, memory and disk space. In any case, if the scenario grows one could split the scenario in several hosts and interconnect them all through a common network.

The phases used when creating a scenario of virtual networks are design, specification, creation of the scenario and running the system. First, the design of the scenario is done defining the networks elements and systems included in the network. A complete network configuration connecting all the systems is needed. In this first phase an initial IP network addressing plan is defined. Once this design is written down, it is manually translated to the VNUML (xml) language using a graphical editor (VNUML GUI) or writing an XML file. Then the scenario is created invoking VNX command line tools. Finally, you are ready to go and interact with the scenario, running the systems and protocols that need to be tested.

3.2.2 Ansible

Ansible is an open-source project widely used for provisioning, configuration and application deployment [4]. A running system is a combination of different factors, not only the version of the software that is running, but also its configuration, the operating system of the hosts is running and even its physical location. If any of these elements change then it's a different system. The configuration management is a way of handling changes in a system using a well-known method so that the system maintains its integrity over time.

Ansible uses YAML files as its main source of information at run time. YAML is a data representation language that is commonly used for configuration. YAML is very easy to read and use. Ansible is written entirely in Python. With Python installed, no additional language dependencies on the machines that you need to manage, but also there are no additional dependencies at all. Ansible works by running commands via SSH, so there's no need to install any server software. Ansible uses an "agentless" model where changes are pushed out to machines on demand.

Ansible performs automation and orchestration via Playbooks. Playbooks are a YAML definition of automation tasks that describe the state of each individual components of the system. Ansible Playbooks consist of series of `plays´ that define automation across a set of hosts, known as the `inventory´. Each `play´ consists of multiple `tasks´, that can target one, many, or all of the hosts in the inventory. Each task is a call to an Ansible module – a small piece of code for doing a specific task. These tasks can be simple or complex. Ansible includes hundreds of modules, ranging from simple configuration management, to managing network devices, to modules for maintaining infrastructure on every major cloud provider.

Core included modules for Ansible are written in a manner to allow for easy configuration of desired state – they check that the task that is specified actually needs to be done before executing it. For example, if an Ansible task is defined to start a web server, configuration is only done if the web server is not already started. This desired state configuration, sometimes referred to as `idempotency', ensures that configuration can be applied repeatedly without side effects, and that configuration runs quickly and efficiently when it has already been applied.

Ansible is used in this project for the configuration management of the different elements of the system, i.e. origin server, mid-tier and edge-tier caches, routers… several examples of use will be shown in the practical section.

4 Use Case

4.1 Multicast MPEG-DASH in Managed Networks

The main objective of this project is to simulate an IPTV network build on an operator's managed network to stream media contents using MPEG-DASH. Then this scenario is improved adding Cache servers to increase network reliability and performance, and so improving user experience.

The use case was designed and developed incrementally starting from creating the network, then adding open-source components for web server and caches (deployment and configuration), and finally adding the multicast function to improve the video delivery. As the scenario is simulated testing could be done in all the phases of the project.

The components and protocols involved in the scenario are depicted in Figure 10.

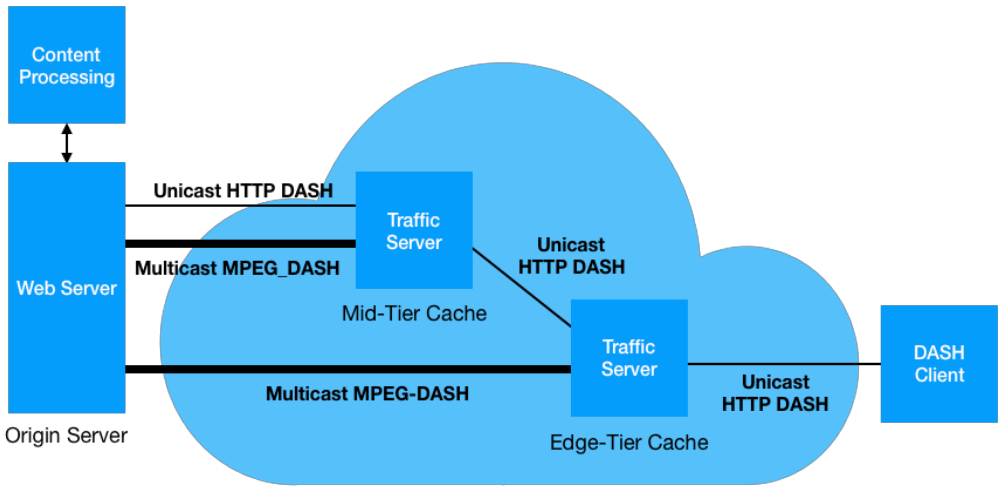

Figure 10 Components and Protocols

The components and tools used to create the scenario are all open-source projects:

- *VNX*, tool designed and developed in DIT of the ETSIT. Excellent tool to quickly create a network with all the components, allowing fast prototyping and testing of complex scenarios in commodity hardware.
- *Apache Web Server*, playing the role of Origin server in the scenario, easy to configure, it provides a generic web server that adapts perfectly for the streaming of video segments of MPEG-DASH files.

- *Apache Traffic Server* is a modular, high-performance proxy server. It plays the role of Mid-Tier and Edge-Tier caches for the media segments of MPEG-DASH.
- *DASH clients*, open-source applications for the playout of the video by the end users.

The following section 4.2 describes the creation of the scenario from the network definition to the installation and configuration of all the open-source components included in it. Section 4.3 is dedicated to the design and description of the multicast components added to improve the performance of the scenario. This section ends with the description of how to run the scenario (section 4.4).

4.2 Building the scenario

The development of the practical scenario is divided in the following phases:

- *network simulation (VNX scenario)*, define and create the IPTV network used to deploy the components,
- *content processing*, manipulation of the original content to create the files needed for the delivery using MPEG-DASH,
- *configuration of Origin and Cache servers*, configuration of *Apache Web Server* and *Traffic Server* using Ansible to meet the requirements of the scenario, and
- *the multicast components*, design and development of the multicast components in the Origin to the Cache servers.

4.2.1 The Video Delivery Network (VNX scenario)

The first stop in the journey is to create the video network. As a real environment is not available, the video network is simulated with virtual machines running in commodity hardware created using the tool VNX. The scenario simulates a managed network of an IPTV operator. To reduce the number of containers started, the network simulated is a reduced to the minimum set of elements that illustrates the scenario.

The simulated scenario consists of the elements depicted in Figure 11 described from top to the edge of the network:

- *Origin server* (*os1*), receives and stores video contents from the packager or transcoder and then serves those contents. This content is served directly to users (unicast MPEG-DASH) or multicasting media segments to Cache servers.
- *Distribution network* (*Net1*) deployed with one Aggregation Ethernet Switch and a Distribution Router (i.e. *r1* node). The Mid-Tier Cache server (*c1* node) is located in this network.

- *Access networks*, deployed with one Access Ethernet Switch, create the networks *Net21* and *Net22*. The Edge-Tier Cache servers (*c21* and *c22* nodes) are located in this network.
- *Hosts* (*h21-1*, *h22-1*, …) to simulate the final users running MPEG-DASH players. The host where all the containers are running is also used to run a player.

The following figure depicts the simulated network architecture:

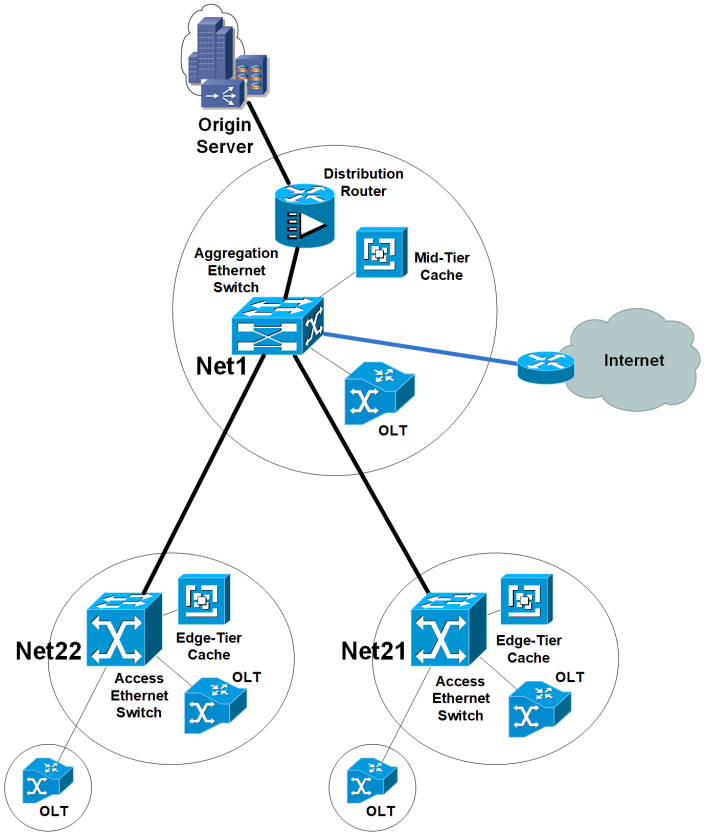

Figure 11 The Video Delivery Network

The scenario is defined in an VNX network specification file, you can check it out at:

https://gitlab.com/irst-workgroup/vnet-sandbox/blob/master/vnx/scenarios/vnet_cache_scenario/vnet_cache_scenario.xml

The Origin server *os1* is created in the VNX file with the following Linux container and configuration:

```
<!-- Origin Server -->
<vm name="os1" type="lxc" arch="x86_64">
  <filesystem type="cow">
    /usr/share/vnx/filesystems/rootfs_lxc_vnet
  </filesystem>
  <mem>512M</mem>
  <shareddir root="/shared">video-repo</shareddir>
```

```
  <if id="1" net="Net0" >
    <ipv4>10.1.0.2/24</ipv4>
  </if>
  <route type="ipv4" gw="10.1.0.1">default</route>
</vm>
```

The root file system used for the Origin server is the same for all the systems of the scenario. In the Wiki of the project in *GitLab* is explained what it is installed and how to reproduce the root file system from scratch. This node has a shared directory with the host where the video repository with the MPEG-DASH files are located. In the interfaces section (<if> tag) the Origin server is connected to network *Net0* with IP address 10.1.0.2. The default gateway is set to Distribution Router *r1*, i.e. 10.1.0.1.

The distribution (*Net1*) and access networks (*Net21* and *Net22*) are simulated with Open vSwitch nodes, the following lines in the VNX file creates those networks and the links between them:

```
<net name="Net0" mode="openvswitch" />
<net name="Net1" mode="openvswitch" />
<net name="Net21" mode="openvswitch" >
  <connection name='link21to1' net='Net1'>
  </connection>
</net>
<net name="Net22" mode="openvswitch" >
  <connection name='link22to1' net='Net1'>
  </connection>
</net>
```

The Distribution Router *r1* is created as a Linux container connecting the networks *Net0* and *Net1* with an IP address both two interfaces. IP forwarding is enabled in this host to enable routing capabilities (using the forwarding element):

```
<!-- Distribution router -->
<vm name="r1" type="lxc" arch="x86_64">
  <filesystem type="cow">
    /usr/share/vnx/filesystems/rootfs_lxc_vnet
  </filesystem>
  <if id="1" net="Net0">
    <ipv4>10.1.0.1/24</ipv4>
  </if>
  <if id="2" net="Net1">
    <ipv4>10.1.1.8/24</ipv4>
  </if>
  <forwarding type="ip"/>
</vm>
```

Two Cache servers are deployed in the scenario: *Mid-Tier* and *Edge-Tier Cache servers*, the former deployed in the distribution network and the latest in the access network.

The Mid-Tier Cache server (*c1*) is deployed as a Linux container connected to the network *Net1* (i.e. the distribution network) with the IP address 10.1.1.1. The default route is set to the distribution router *r1*.

```
<!-- Mid-Tier Cache Server (Apache Traffic Server) -->
<vm name="c1" type="lxc" arch="x86_64">
  <filesystem type="cow">
    /usr/share/vnx/filesystems/rootfs_lxc_vnet
  </filesystem>
  <mem>512M</mem>
  <if id="1" net="Net1" >
    <ipv4>10.1.1.1/24</ipv4>
  </if>
  <!-- Route to the Distribution Router -->
  <route type="ipv4" gw="10.1.1.8">default</route>
</vm>
```

The Edge-Tier Cache (*c2*) is configured with the following container:

```
<!-- Edge-Tier Cache Server (Apache Traffic Server) -->
<vm name="c2" type="lxc" arch="x86_64">
  <filesystem type="cow">
    /usr/share/vnx/filesystems/rootfs_lxc_vnet</filesystem>
  <mem>512M</mem>
  <if id="1" net="Net21" >
    <ipv4>10.1.1.2/24</ipv4>
  </if>
  <!-- Route to the Distribution Router -->
  <route type="ipv4" gw="10.1.1.8">default</route>
</vm>
```

To simulate clients executing MPEG-DASH players a set of Linux containers are deployed in the scenario, here is the configuration of one of them:

```
<!-- Dash client located in Net21 -->
<vm name="h21-1" type="lxc" arch="x86_64">
  <filesystem type="cow">
    /usr/share/vnx/filesystems/rootfs_lxc_vnet
  </filesystem>
  <if id="1" net="Net21" >
    <ipv4>10.1.1.5/24</ipv4>
  </if>
  <route type="ipv4" gw="10.1.1.8">default</route>
</vm>
```

The scenario created can be checked using the graphical map tool included with VNX executing the following command:

```
$ sudo vnx -f vnet_cache_scenario.xml -v --show-map
```

The output image generated is:

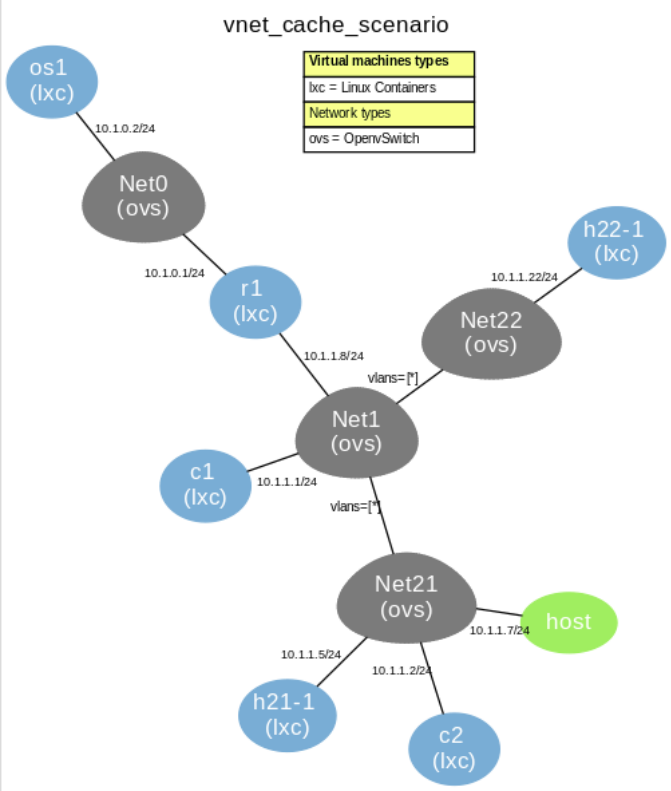

Figure 12 VNX scenario

4.2.2 Video Content Processing component

The *Video Content Processing component* creates and stores the files needed by the Video Server to broadcast contents using MPEG-DASH, allowing users to playout the contents smoothly from different devices (i.e. different resolutions), everywhere (i.e. networks with different bandwidth), and every time.

The original video content is provided in some container format, with a certain video codec, and including one or more audio tracks. This video should be prepared for MPEG-DASH playout. H.264/AVC will be used for video stream within segmented mp4 containers.

For the use case, this component is performed manually in the host executing commands in the shell and storing the output files in a directory, that is shared with the container running the Origin Server. Figure 13 represents the steps that model this component:

- *Transcoding*, using *ffmpeg* command line tool, this step is optional and is done depending on the format of the original content.
- *Create the MPEG-DASH representations*, using *ffmpeg* command line tool this element creates the different representations of the content.

- *Segmenting*, using GPAC *MP4Box* command line tool, file is segmented in chunks.
- *Store the output files in the media repository* (in this case a local filesystem), i.e. the MPEG-DASH related files.

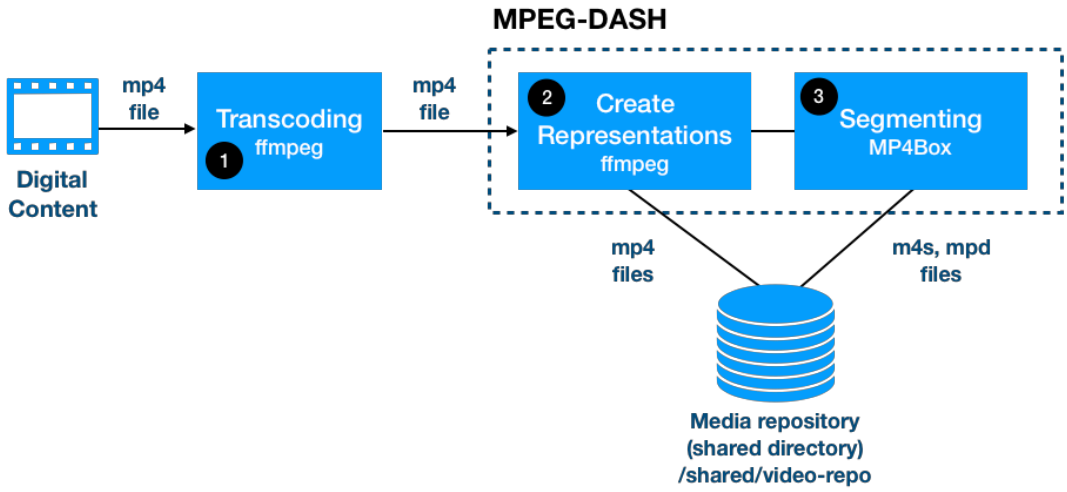

Figure 13 Video content processing component

The digital content used for the use case is the movie '*Big Buck Bunny*', licensed under Creative Commons-BY, license that enables the free distribution of work, allowing the right to share, use, and build upon the work of the author. This movie is available in several different sizes and formats. The video is encoded with x264 and contain both AC-3 5.1 surround sound track as well as a normal mp3 stereo track. The videos come in three different categories: hardware-assisted stereoscopic 3D, color-coded stereoscopic 3D and finally the 2D version. The latest is the video category used for this use case, i.e. 2D version. These files can be downloaded here:

http://bbb3d.renderfarming.net/download.html

Transcoding

This optional component can be skipped if the source original video is already in the expected format, but probably in general this will not be the case. The *ffmpeg* command is used to create desired file.

Transcoding, at a high level, is taking already-compressed (or encoded) content, decompressing (decoding) it, and then somehow altering and recompressing it. As an example, the audio and/or video format (codec) might be changed from one to another, such as converting from an MPEG2 source (commonly used in broadcast television) to H.264 video and AAC audio (the most popular codecs for streaming). Other basic tasks could include adding watermarks, logos or other graphics to the video.

Create MPEG-DASH Representations

The MPEG-DASH *Media Presentation Description* (*MPD*) is an XML document containing information about media segments, their relationships and information necessary to choose between them, and other metadata that may be needed by clients.

Following is a brief description of the most important parts of the MPD, starting from the top level (periods) and going to the bottom (segments).

Periods, contained in the top-level MPD element, describe a part of the content with a start time and duration. Multiple *Periods* can be used for scenes or chapters, or to separate ads from program content.

Adaptation Sets contain a media stream or setoff media streams. In the simplest case, a Period could have one *Adaptation Set* containing all audio and video for the content, but to reduce bandwidth, each stream can be split into a different *Adaptation Set*. A common case is to have one video *Adaptation Set*, and multiple audio *Adaptation Sets* (one for each supported language). *Adaptation Sets* can also contain subtitles or arbitrary metadata.

Representations allow an Adaptation Set to contain the same content encoded in different ways. In most cases, *Representations* will be provided in multiple screen sizes and bandwidths. This allows clients to request the highest quality content that they can play without waiting to buffer, without wasting bandwidth on unneeded pixels. *Representations* can also be encoded with different codecs, allowing support for clients with different supported codecs, or to provide higher quality *Representations* to newer clients while still supporting legacy clients (providing both H.264 and H.265, for example). Multiple codecs can also be useful on battery-powered devices, where a device might choose and older codec because it has hardware support (lower battery usage), even if it has software support for a newer codec.

Media segments are the actual media files that the DASH client plays, generally by playing them back-to-back as if they were the same file. Media segments locations can be described using BaseURL for a single-segment Representation, a list of segments (SegmentList) or a template (SegmentTemplate).

For the use case the following representations are created:

Representation Number	Bitrate	Resolution (width : height)
1	250Kbps	640:360
2	500Kbps	1024:576
3	1Mbps	1280:720
4	2Mbps	1280:720

To create the representation 4, i.e. **2Mbps** for video bitrate, resolution **1280:720**, with **H.264** video encoding, a frame rate of **24 fps**, and a **GOP** length of **24**, the following *ffmpeg* command is executed:

*$ **ffmpeg** -i input_file.mp4*
*-c:v **libx264** -c:a copy -b:v **2M -r 24***
*-profile:v baseline -level 3.0 **-g 24 -vf scale=1280:720** bbb_sunflower_repMM1.mp4*

Just some comments about the options used in the generated media about MPEG compression (type of pictures), GOP, ...

The MPEG standard specifically defines three types of pictures:

- *Intra-frames* (*I-frames*). An *I-frame* is encoded as a single image, with no reference to any past of future frames. They are self-contained and used as a foundation to build other types of frames.
- *Forward predicted frames* (*P-frames*). A *P-frame* is a predicted frame and is based on past I-frames. It is not actually an encoded image and contains motion information that allows to rebuild the frame. *P-frames* require less bandwidth than I-frames, which is particularly important for IPTV based networks.
- *Bi-directional predicted frames* (*B-frames*). A *B-frame* is a bidirectional frame made up from information from both *I-frames* and *P-frames*. The encoding for *B-frames* is similar to *P-frames*, except that motion vectors may refer to areas in the future reference frames. *B-frames* occupy less space that *I-frames* or *P-frames*. So, relating back to an IPTV environment a stream that contains a high density of *B-frames* will require less bandwidth compared to a digital stream built with a high density of *I-* and *P-frames*. Even though *B-frames* help to minimize the bandwidth requirements of MPEG video streams, there is one main drawback: time delays. The time delays are created because the player will have to wait to examine two reference frames before it is capable of processing the *B-frame* details.

GOP (*Group of Pictures*) is related only to a video content and it is one of the important parameters of DASH media. Incorrectly set GOP may affect the overall quality of the media; it affects the quality of the playback in adaptive streaming scenarios, when the media player switches between different quality levels or it may event prevent the media player from playing such media at all.

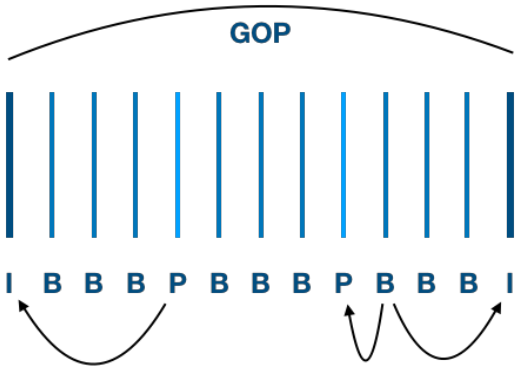

Figure 14 GOP and I, B and P frames

Compressed H.264 (aka AVC) and H.265 (aka HEVC) media consists of frames of different types: I-, P- and B-frames. The frames are organized in a specific order. Let's say we have a sequence of frames (Figure 14): **I** B B B P B B B P B B B **I**. The GOP structure starts with and I-frame so the GOP length (distance between two consecutive I-frames), in this case, is equal to 12.

There are many reasons why GOP is important, and it's essential to consider them all while creating DASH media for adaptive streaming, some of them are:

- **Precision of seeking in a media player.** Media players can start playback only from I-frames, since this type of frames is less compressed and contains the most of information. So, if the GOP length is big, I-frames will be far away from each other, and a user of the media player won't be able to navigate to a place between these I-frames. Instead, the media player will jump to a nearest I-frame. This effect may be undesirable, if the precision of seeking is important. By making the GOP length smaller, the precision of seeking may be increased.
- **Seamless quality level switching.** In adaptive streaming scenarios, media may have multiple quality levels, i.e. representations in MPEG-DASH. Depending on network conditions or bandwidth, the highest quality that is possible is not playout in the media player of the user, seamlessly switching between representations. But the media player can start the playback of another representation only starting from an I-frame, so, in order to make it possible to switch seamlessly, I-frames in different representations must have the same locations. Otherwise, there will be noticeable "jumps" in the playback or even the playback may be stopped at all.
- **Media segmentation.** The media player does not retrieve the whole media (with all its representations). Instead, it retrieves only the segments that needs at the current moment, and those that it can actually retrieve without stopping

the playback. In order to make seamless switching possible, segments must start with an I-frame, and duration of segments must be equal.

- **Media server load (number of segments requests).** If the GOP length is very small, then there will be a lot of segments. This is good if the network conditions change very fast – in this case, the media player will be able to faster respond to such changes. But this also has a negative impact on a media server load, since the media player will be performing a lot of segment requests. By increasing the segment duration, it will be possible to decrease the media server load, but the media player won't be able to switch to another quality levels very fast.
- **Playback latency.** Since the GOP length may control the segment duration, shorter segments may be loaded by the media player faster. As a result, the playback starts also faster.

To finish GOP description, mention that there are two types of GOP:

- **Closed GOP.** It starts with an *I-frame* and doesn't reference frames from other surrounding GOPs, i.e. the final *B-frame* does not require an *I-frame* from the next GOP for decoding.
- **Open GOP.** It doesn't start with an *I-frame*, but it references the last frame from a previous GOP, i.e. open GOPs do require the *I-frame* contained in the next GOP.

Although open GOP increases the compression efficiency, it also increases the complexity of decoders. As a result, not all media players support it. That's why it's safer to use the closed GOP.

Speaking of GOP, the only important thing to consider is the number of frames per second (FPS). In the use case the original file there are 30 fps, with a segment duration of 2 seconds, this means that an I-frame will be located 30 fps * 2 secs = 60 frames apart from each other (a segment will start from an I-frame and will contain 60 frames). GOP length in this case is 60.

H.264 uses profiles and levels for its encoding and decoding. A profile defines the complexity of the encoding process, i.e. a profile represents a sub-set of the encoding techniques available in H.264. This is useful to target decoders with very different decoding capabilities (memory and processing power). H-264 has the following profiles among others:

- *Baseline or simple*. No *B-frames* are used when using this simple profile.

- *Main profile*. This profile uses all three frame types I, P and B. This profile is commonly used for deploying multicast IPTV services over a broadband network.
- *High profile*. This profile is used for broadcast DVB HDTV.

To complement these variations in picture quality, H.264 has also subdivided profiles into levels. These levels are used to define constraints on picture size, frame rate, bit rate, and buffer size for each of the defined profiles. Different profile and level combinations have already been established for different products and for different applications of digital compression.

Segmenting

When working with adaptive streaming formats such as MPEG-DASH, one point of decision is how long to generate the media segments of the content. The segmentation of the content is necessary, as it enables the switching between the different qualities during the streaming session.

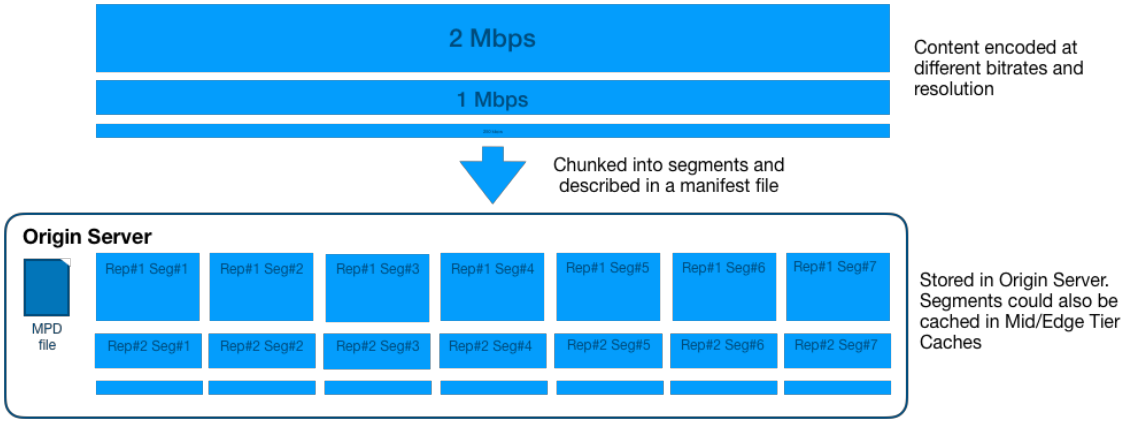

Figure 15 MPEG-DASH segmenting

This question about the optimal segment length has not an easy answer and depends on the environment (i.e. fixed access or mobile access), the type of content (e.g. cartoon or special effects movie),… Short segments are good to adapt quickly to bandwidth changes and prevent stalls, but longer segments may have a better encoding efficiency and quality.

To enable seamless switching between different quality representations of MPEG-DASH, it is required to maintain a fixed I-frame positions in the video, e.g. after 48 frames, an I-frame has to be set in a 24 frames-per-second (fps) video and segment length of 2 seconds. This is necessary to guarantee I-frames at the beginning of each segment, which is needed to be able to switch representations between different segments. By doing so at the beginning of a new segment, the decoder does not need any references to previous frames or segments and therefore the new segment can have frames in

different resolutions, bitrates or framerates. Fixed I-frame positions can be achieved by restricting the group-of-picture (GOP) size of the encoder to the segment size of the content.

From the network perspective, there are a lot of influencing factors to be considered. Longer segment lengths may cause stalls using wireless internet connection with high bandwidth changes, but short segment lengths may result in poor streaming performance due to overhead produced by requests and the influence of the network delay.

The chunk sizes of MPEG-DASH used for this use case will be 2 seconds, which is a good compromise between encoding efficiency and flexibility for stream adaption to bandwidth changes.

To create the segments the multimedia packager *MP4Box* is used. Assumed that working with four representations, a target DASH full profile, and a segment length of 2 seconds, and example of segmenting is:

$ ***MP4Box*** *-dash* ***2000*** *-profile* ***full***
-segment-name dash_%s_ -out ***videobbb.mpd***
./../bbb_sunflower_repMM1.mp4 ./../bbb_sunflower_repMM2.mp4
./../bbb_sunflower_repMM3.mp4 ./../bbb_sunflower_repMM4.mp4

This command generates the MPD and segment files for the delivery of the content using MPEG-DASH.

Store output files in the Media Repository

The resulting files are stored in the *Media Repository* located at the local file system of the host. For the simplicity of the scenario the files are stored in the host file system and this directory is shared with the container where the Origin Server is deployed. In a real scenario the *Video Content Processing component* could be located in another server and a mechanism to upload the files to the Origin server should be needed. Additionally, the files could be stored in the server in a database (e.g. *Ceph*). This could be a future enhancement to this use case described in section 5.

4.2.3 The Origin Server

The *Origin server* is deployed using the open-source Apache Web Server. The root file system *vnx_rootfs_lxc_ubuntu64-16.04-v025* used to create the scenario already included that server installed. The Apache Web Server provides the functions of a web server and it adapts perfectly to provide MPEG-DASH files, i.e. media representation files and media segments to users running a DASH client.

The components of the Origin server are depicted in Figure 16. The Content Processing component is described in section 4.2.2. The Multicast component is described in section 4.3.1.

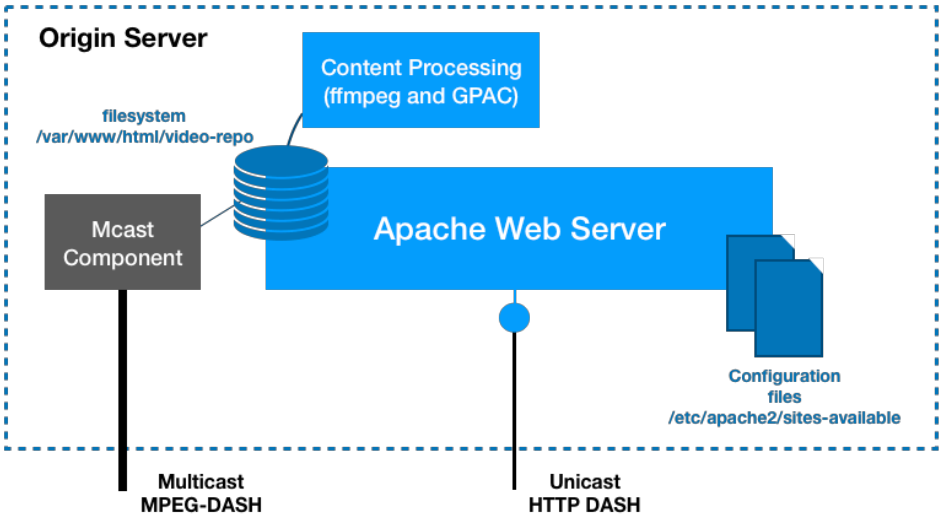

Figure 16 The Origin Server

The Origin server is the main source of the video files. Those files are ingested from a Content Processing component that transcodes the original files to the representations exposed by the server. Those files are stored locally in the Media Repository (a filesystem in the server).

The tasks needed to configure the Apache Web server to play the role of an Origin server are:

- Create a symbolic link to the *Media Repository*, implemented as a shared directory and this link points to that repository.
- Enabled the *mod_headers*. This is an Apache module that provides directives to control and modify HTTP request and response headers.
- Add a rule in the server to include the *Cache-Control* header in the responses with status code 200 using the module *mod_headers*.

This configuration is done using Ansible with the following tasks included in the playbook *vnet-sandbox.yml*:

```
- hosts: os1
  tasks:
    - name: Add multicast static route
      become: yes
      command: route add -net 224.0.0.0 netmask 240.0.0.0 eth1
```

```
- name: Enabled mod_headers
  become: yes
  apache2_module:
    name: headers
    state: present

- name:
  become: yes
  lineinfile:
    path: /etc/apache2/sites-available/000-default.conf
    insertbefore: '</VirtualHost>'
    line: Header append
      Cache-Control s-maxage=600 "expr=%{REQUEST_STATUS} == 200"
    state: present

- name: Create symbolic link to video repository
  become: yes
  file:
    src: /shared
    dest: /var/www/html/video-repo
    owner: root
    group: root
    state: link

- name: Restart Origin Server (apache2)
  become: yes
  service:
    name: apache2
    state: started
```

4.2.4 The Mid-Tier and Edge-Tier Cache Servers

Mid/Edge Tier Cache servers are deployed using open-source *Apache Traffic Server*, adding a custom multicast component to push media segments in local database cache. The root file system *vnx_rootfs_lxc_ubuntu64-16.04-v025* used to create the scenario does not have *Apache Traffic Server* already installed. The root file system (*vnx_rootfs_lxc_vnet_ubuntu64-16.04-v025-1*) created from that one adds the installation of the *Apache Traffic Server*. In the Wiki of the project in GitLab is explained what it is installed and how to create the root file system from scratch.

The Origin server (including the multicast component) is depicted in Figure 17. The *Apache Traffic Server*, acting as an HTTP reverse proxy, exposes an HTTP port to provide MPEG-DASH contents to users. The content is provided from local DB when a cache hit and forwarded to next level in case a cache miss. Additionally, a Multicast Receiver Component is included to improve the scenario, it is discussed in detail in next section.

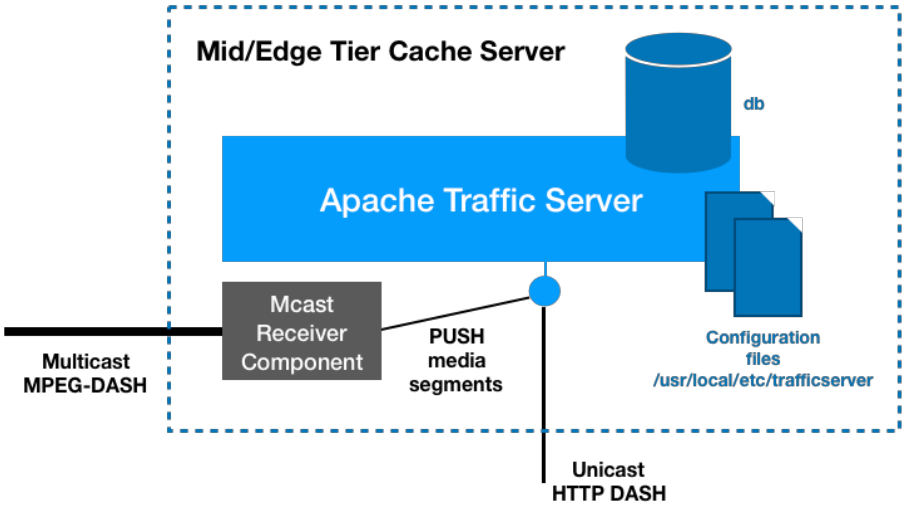

Figure 17 Mid/Edge-Tier Cache Servers

This section focuses on the configuration of the *Apache Traffic Server*. The specific configuration for the Mid/Edge Tier Cache servers are:

- *Enable Reverse Proxying*, the Apache Traffic Server plays the role of a cache server in the scenario. To do that, the server is configured as a reverse proxy, to intercept the request from the clients to the origin server and respond in the content is stored in the local cache, on the contrary the request is forwarded to origin server (or the next level of caching).
- *Configure Traffic Server to accept PUSH requests*, those requests are needed to allow the Multicast Receiver Component to store objects in the local cache of the Traffic Server.
- *Adding caching rules in the Traffic Server*, e.g. MPD files are not cached, or allow all HTTP methods coming from our network…
- *Add a rule for the URL mapping*, a rule is created for each of the Cache servers. Request coming to Edge-Tier Cache with a miss in local cache is forwarded to Mid-Tier Cache. And the latest in case of a miss in a requested segment is forwarded to Origin Server.

This configuration is done using Ansible with the following tasks included in the playbook *vnet-sandbox.yml*:

```
- hosts: caches
  tasks:
    - name: Enable Reverse Proxying
      become: yes
      lineinfile:
        path: /usr/local/etc/trafficserver/records.config
        regexp: '^CONFIG proxy.config.url_remap.pristine_host_hdr'
```

```
      line: 'CONFIG proxy.config.url_remap.pristine_host_hdr INT 1'

  - name: Enable Reverse Proxying - proxy.config.http.server_ports 80
    become: yes
    lineinfile:
      path: /usr/local/etc/trafficserver/records.config
      regexp: '^CONFIG proxy.config.http.server_ports'
      line: 'CONFIG proxy.config.http.server_ports STRING 80 80:ipv6'

  - name: Configure Traffic Server to accept PUSH requests
    become: yes
    lineinfile:
      path: /usr/local/etc/trafficserver/records.config
      regexp: '^CONFIG proxy.config.http.push_method_enabled'
      line: 'CONFIG proxy.config.http.push_method_enabled INT 1'

  - name: Caching rule in Traffic Server - mpd files not cached
    become: yes
    lineinfile:
      path: /usr/local/etc/trafficserver/cache.config
      line: 'dest_ip=10.1.1.1 suffix=mpd action=never-cache'

  - name: Caching rule in Traffic Server - remove deny actions IPv4
    become: yes
    lineinfile:
      path: /usr/local/etc/trafficserver/ip_allow.config
      regexp: '^src_ip=0.0.0.0(.*)ip_deny'
      line: 'src_ip=0.0.0.0-255.255.255.255 action=ip_allow method=ALL'

  - name: Caching rule in Traffic Server - remove deny actions IPv6
    become: yes
    lineinfile:
      path: /usr/local/etc/trafficserver/ip_allow.config
      regexp: '^src_ip=::(.*)ip_deny'
      line: 'src_ip=::-ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff
             action=ip_allow method=ALL'
```

The rule for the URL mapping is different in every cache server. An example of this task for the *Mid-Tier Cache server* is:

```
- hosts: c1
  tasks:
    - name: Add rule to URL mapping config file
      become: yes
      lineinfile:
        path: /usr/local/etc/trafficserver/remap.config
        line: map http://10.1.1.1/video-repo/ http://10.1.0.2/video-repo/

    - name: Restart Apache Traffic Server (Mid-Tier Cache)
      become: yes
      command: /usr/local/bin/trafficserver restart
```

4.3 Detailed design of multicast components

Once the *Apache Web Server* and *Traffic Server* are deployed and configured, a multicast component is added to both of them to improve the bandwidth usage on the managed network.

In the *Origin server* a *Multicast component* is added to send the media segments of most popular content (grey-box in Figure 16) to the cache servers deployed in operator's network.

In the *Mid/Edge-Tier Cache servers* a *Multicast Receiver component* is added to receive the multicast and store the media segments in local cache (grey-box in Figure 17). From this point the *Cache server* can provide the media segments from local cache instead of forwarding the request to the upper node, i.e. another *Cache server* or finally the *Origin server*.

Those new components are depicted together in Figure 18.

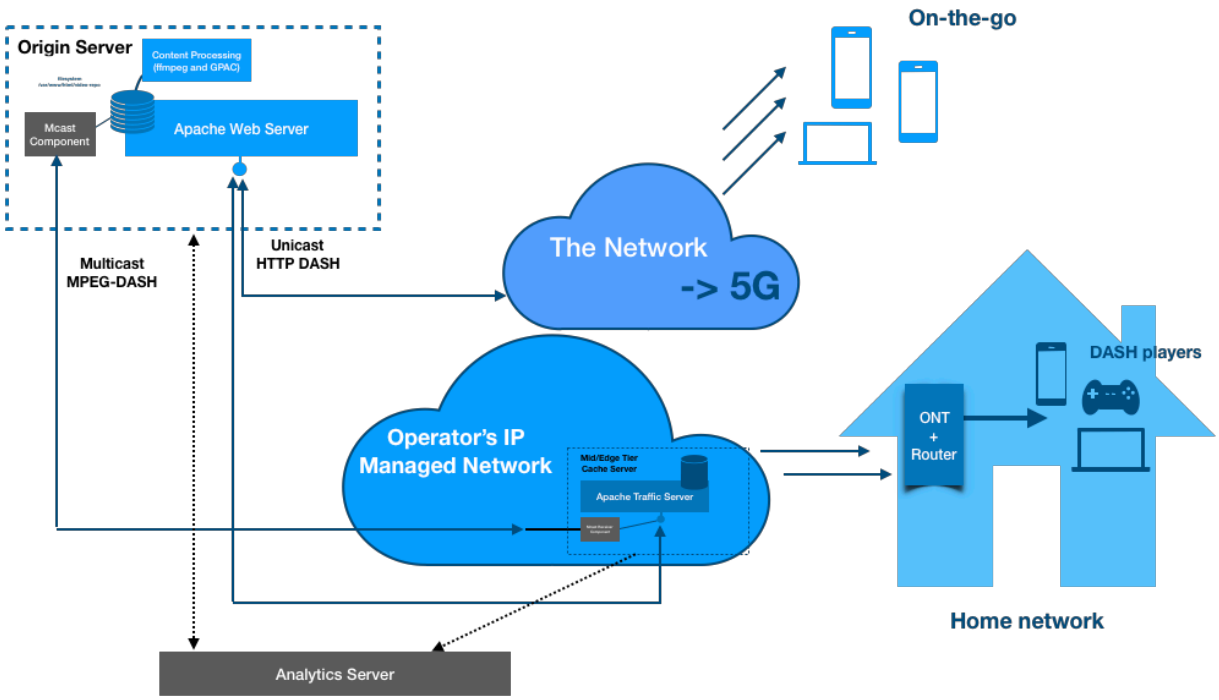

Figure 18 Multicast solution in MPEG-DASH video delivery

The *Origin server* will multicast the most popular contents to the *Cache servers* deployed in the managed network that are closest to the final user, based on some well-known metrics or expected consumption of the contents. The final user will receive MPEG-DASH in home network video streaming from operator's managed network. The multicast components are deployed in the managed network, but the *Multicast Receiver component* can be deployed also in-home network to provide a better user experience

allowing to watch the same content in different terminals synchronized without different delays. The Network in the figure represents the 5G network that could also benefit from having those components closest to the final users to watch contents on-the-go.

4.3.1 Design of the Multicast component of the Origin server

The *Origin server* includes a multicast component to send the most popular contents to network caches. This component is instructed by the Analytics component based on some metrics provided by the user's activity. This latest component is out-of-scope in the project and the *Multicast component* sends the media components that has manually configured.

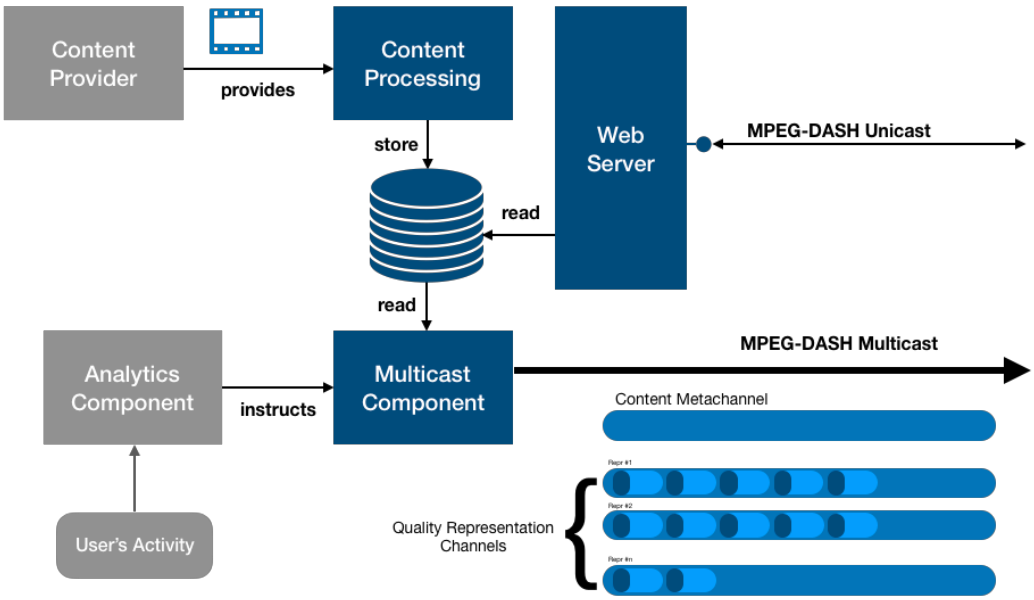

Figure 19 Origin Server design

There are two types of multicast groups:

- *Content Metachannel*, contains metadata information of the multicast groups delivering media segments, e.g. multicast IP, content broadcasted in the group, …
- *Quality Representation Channel*, these channels transport media segments for a given representation. Media segments are fragmented in files of fixed size including a custom header to allow reassemble in the cache servers.

Content Metachannel

The *Content Metachannel* includes information about the *Quality Representation Channels* multicast from the Origin server. For each channel the following information is included:

- *Channel identifier*,
- *Multicast address* of the channel where the media contents are broadcasted. This will be used by the *Cache servers* to join the multicast group and receive the media segments of the contents.
- *MPEG-DASH properties* of the contents included in this channel, for example segment duration, video bitrate, video frame rate, profile, GOP length, …
- *List of contents* multicast in the channel, including metadata of the contents broadcasted:
    - *Base URL*, URL where all the segments of a content are located in the Origin server, e.g. *http://vnet.origin1.com/video-repo/videoBBB/*.
    - *Format for the naming of segments*, this should be similar to the value of the parameter *-segment-name* used in the *MP4Box* when creating the representations, e.g. *dash_$Name$_$ReprID$_$Number$.m4s*.
    - Number of segments in the representation.
    - Additionally, can be added any information related to the content (theme, parental rating, …).

This metachannel is used to communicate with the Cache servers the information of the content being multicast from Origin server to Cache servers. This is a simple solution instead of having an individual communication with every Cache server.

An example of the metadata sent in this channel is:

```
{
  "version": 1.0,
  "source": "vnet.origin1.com",
  "channels": [
    {
      "id": "q-repr-1",
      "mcast_address": "224.10.10.11",
      "dash_properties": {
        "profile": "full",
        "segment_duration": "2s",
        "video_bitrate": "2M",
        "video_frame_rate": "30fps",
        "GOP": 60
      },
      "contents": [
        {
          "base_url": "http://vnet.origin1.com/video-repo/videoBBB/",
          "segments_format": "dash_videoBBBrepMM4_%d.m4s"
        },
```

```
            {
              "base_url": "http://vnet.origin1.com/video-repo/videoBee/",
              "segments_format": "dash_videoBeerepMM4_%d.m4s"
            }
          ]
        },
        {
          "id": "q-repr-2",
          "mcast_address": "224.10.10.12",
          "dash_properties": {
            "profile": "full",
            "segment_duration": "2s",
            "video_bitrate": "4M",
            "video_frame_rate": "30fps",
            "GOP": 60
          },
          "contents": [
            {
              "_comment": "Contents come here..."
            }
          ]
        }
      ]
    }
```

Quality Representation Channel

The *Quality Representation Channels* transport the media segments of a representation of a content transmitted using MPEG-DASH. These media segments (i.e. objects in the context of the Apache Traffic Server) are stored in the local database to send them back to a client device in case of a cache hit.

The implementation of these channels is done in the class *QualityReprChannel.java*. This class creates a new thread for each *Quality Representation Channel* and use internally *JGroups* as the communication library. When the channel is started a configuration file *vnet-qualityrepr-<MCAST_GROUP>.xml* is read to configure the *JGroups* communication layer. The configuration parameters included in this file are described in section 3.1.4. Only a subsection of them is needed for this use case. The design of the Multicast components is done to be able to replace the communication library (i.e. *JGroups* library) by a custom implementation that should add reliability, flow control and failure detection to the Multicast component. As this development could take considerable effort, a decision of using *JGroups* was done to speed up the development of the multicast components. This task could be scheduled as future work.

For the multicast components the following protocols from the JGroups stack are used (Figure 20): initial membership discovery, failure detection, reliable message transmission, message stability, group membership and flow control.

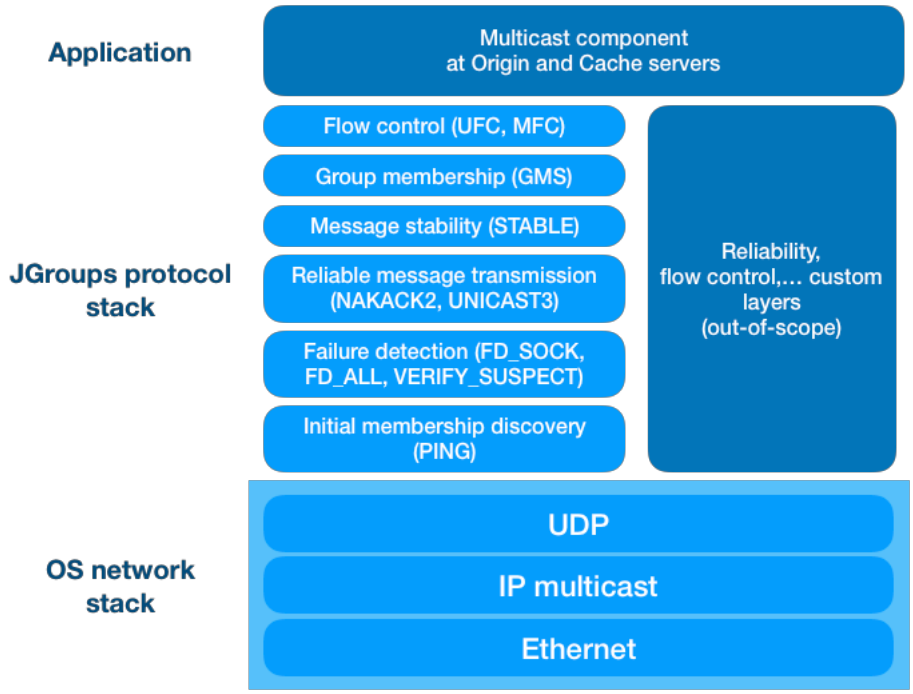

Figure 20 JGroups in Multicast components

The *Quality Representation channel* sends the media segments created for the representation of a content, i.e. the *m4s* files generated by the *Content Processing component*. Those files are stored in the *Media Repository* (file system in the host) in the Origin server. The multicast component reads those segments from the file system and sends them as messages fragmented into the multicast group.

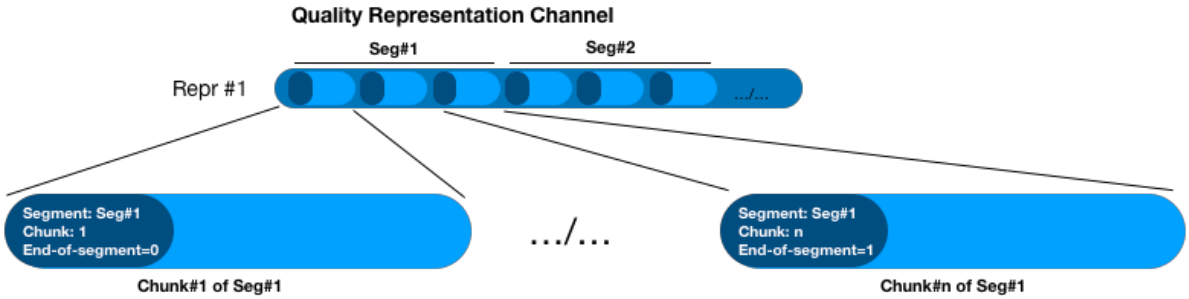

Figure 21 Fragmentation in the Quality Representation Channel

Although the *JGroups* library provides fragmentation capabilities in its protocol stack, this function has been implemented for the use case. The segments are divided in chunks, and every chunk has a header indicating the segment, the chunk number, and whether it is the final chunk of the segment (end-of-segment). This is depicted in Figure 21. Note that *JGroups* provides reliable transfer and no additional information is identified to be included. The custom header is implemented in the class *VnetMediaSegmentHeader*. This class extends *Header* class of JGroups to add the

parameters needed by the implementation (i.e. identification of the segment, the end-of-segment flag, and potentially any other).

4.3.2 Design of the Multicast Receiver component of the Cache Servers

The *Multicast Receiver component* of the Cache servers implements two main functions, read and store:

- *Read the multicast groups*, i.e. the *Content Metachannel* and *Quality Representation Channels*, in order to get the information of the multicast groups with contents (from the *Content Metachannel*), and the media segments of the popular contents from those multicast groups.
- *Store the media segments* in the local cache of the *Apache Traffic Server*.

Once a valid segment is received (an *MD5 checksum* can be used to check it out), the segment media file is stored in local cache. This cache consists of a high-speed object database called *object store* that indexes cache objects according to URLs and their associated headers. Pushing an object into the local database is a key function in the multicast component that it is explained in the next section.

Pushing an object into the Cache

Apache Traffic Server accepts a custom HTTP request method *PUSH* to put an object into the local cache. If the object is successfully written to the cache, then *Traffic Server* responds with a *200 OK* HTTP message; otherwise a *400 Malformed Pushed Response Header* is returned.

Pushing the object to the local cache is done using the *Apache HttpClient* (from the *Apache HttpComponents* project). A new class is created for the custom HTTP request method *PUSH*, called *HttpPush* extending *HttpEntityEnclosingRequestBase*. This HTTP method is mandatory to store objects in local cache.

A class called *MediaSegment* is created to manage the storage of the media segments, pushing the objects to the *Apache Traffic Server*. In addition to the method *PUSH* the request must include all the HTTP headers that would have been included if the media segment were returned by the Origin server in a *200 OK* HTTP response. The examples included in the official documentation (using *curl* tool) of the *Apache Traffic Server* were used to initially understand the request to store an object.

Once the media segments are written in the object store, the *Apache Traffic Server* will return those segments from it local cache instead of forwarding the request to the upper layer in case a cache hit.

The following is an excerpt of the code that implements the *MediaSegment* class.

```
public class MediaSegment {

    private HttpEntity fileEntity;

    public void setFileEntity(String pathname) throws Exception {
        File f = new File(pathname);

        this.fileEntity = new FileEntity(f);
    }

    public void storeInCache(String cacheIpAddress, String mediaSegmentPath) throws Exception {

        HttpClient client = HttpClientBuilder.create().build();

        URI uri = new URIBuilder()
                .setScheme("http")
                .setHost(cacheIpAddress)
                .setPath("/video-repo" + mediaSegmentPath)
                .build();

        HttpPush request = new HttpPush(uri);

        request.setHeader( name: "Accept", value: "*/*");
        request.setHeader( name: "Content-Type", value: "application/x-www-form-urlencoded");
        request.setHeader( name: "Expect", value: "100-continue");

        String body = "HTTP/1.1 200 OK\n" +
                "Server: ATS/9.0.0\n" +
                "Accept-Ranges: bytes\n" +
                "Cache-Control: s-maxage=600\n" +
                "Age: 51\n" +
                "Connection: keep-alive\n" +
                "\n";

        HttpEntity entity = new StringEntity(body);
        InputStream s1 = entity.getContent();
        long l1 = entity.getContentLength();

        InputStream s2 = this.fileEntity.getContent();
        long l2 = this.fileEntity.getContentLength();

        SequenceInputStream final_body = new SequenceInputStream(s1, s2);
        InputStreamEntity inputStreamEntity = new InputStreamEntity(final_body, length: l1 + l2);
        inputStreamEntity.setChunked(false);

        request.setEntity(inputStreamEntity);

        System.out.println(request.toString());
        System.out.println(Arrays.toString(request.getAllHeaders()));

        HttpResponse response = client.execute(request);
        System.out.println(response.toString());

    }
}
```

Figure 22 MediaSegment class

The *MediaSegment* class has no constructor and the media segment is set using the setter function *setFileEntity()*, a *FileEntity* object is created from it. When the component wants to store the object into local cache the method *storeInCache()* is called. The URI to push the object ① is the IP address where the *Cache server* is listening adding the path of the segment is the Origin server. This is the index of the cached object. Then ② the *PUSH* HTTP request is created adding the headers *Accept*, *Content-Type* and *Except*. The body of the request is the *200 OK* HTTP response from the Origin server with their associated headers ③. It is important to add the Cache-Control header in the response. An *HttpEntity* is created with these associated headers ④. The final body is created with these two entities, i.e. headers and media segment ⑤.

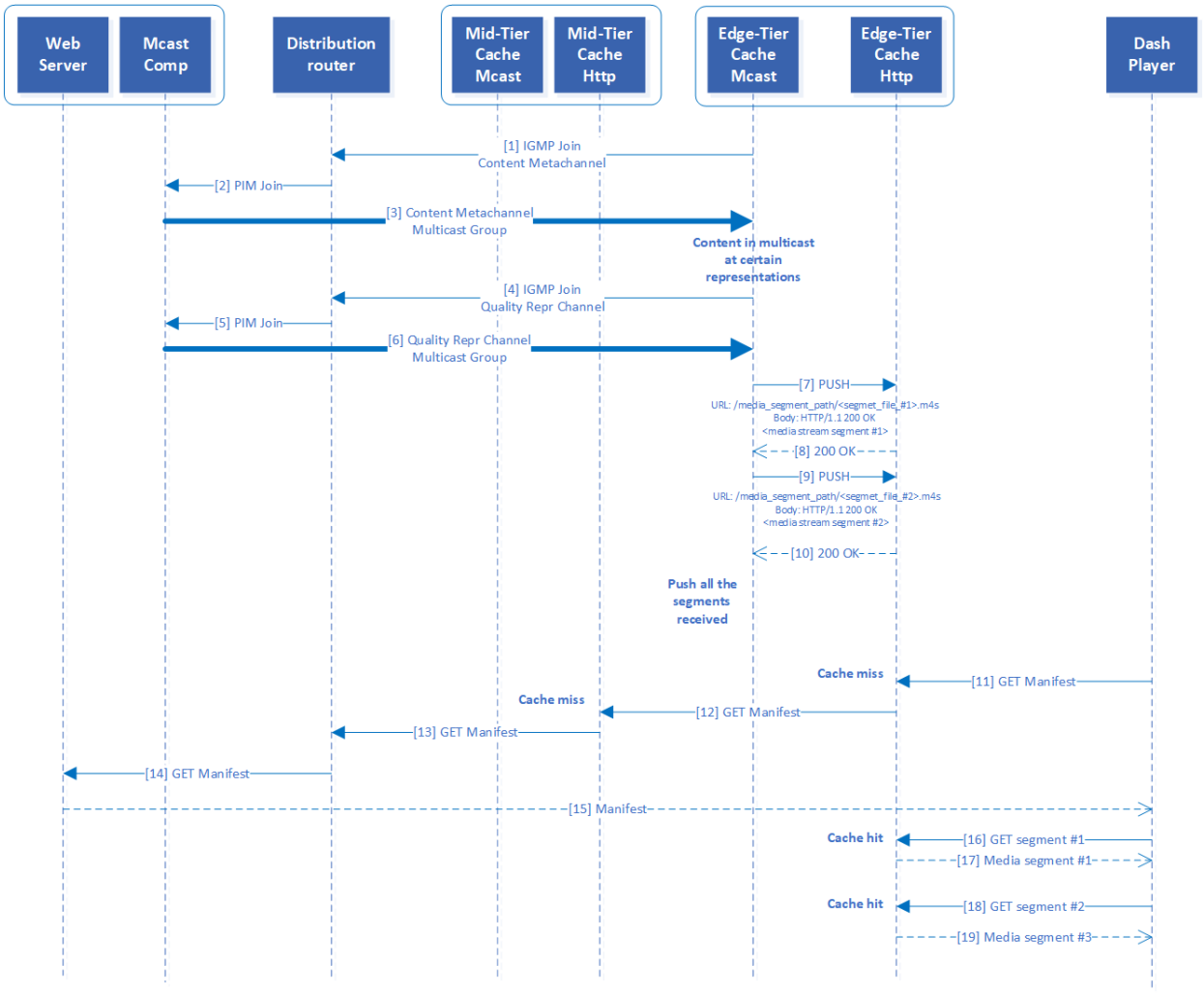

Figure 23 Multicast MPEG-DASH media segments

The sequence diagram in Figure 23 shows the messages exchanged in the network from the *Origin server* to the *Cache server* to multicast the media segments and store those objects in the local cache. And then, returned the cache objects from local cache when a client device sends a request for that media segments.

In the example scenario the *Edge-Tier Cache server* joins the *Content Metachannel* [1] and receives information about the *Quality Representation Channels* (i.e. multicast groups) that are multicasting representations of contents. When reading this metachannel at some point it detects that a popular content is available in multicast at certain representations (in this case only one, assumed the representation with higher bandwidth requirements), so the *Multicast Receiver Component* joins the *Quality Representation Channel* [4] indicated in the metachannel. When the multicast with the media segments is received [6], the multicast component of the *Edge-Tier Cache* stores all the objects in the local database sending a *PUSH* request ([7] and [9] in the figure) to the *Apache Traffic Server* (as previously described in section 4.3.2). The objects are stored correctly in the local database for its future use. When a client starts to play a content

and asks for its Manifest file [11], in this case the request gets to the *Origin server*, as it was not previously cached in any server. When the client device requests the media segments of the content [16] and [17], a cache hit occurs, and those objects are returned from local cache. Note that this process is completely transparent to the DASH player whether the media segment is delivered from a unicast workflow or via multicasting those segments previously to the network caches.

4.4 Running the scenario

To start the scenario just a few commands are needed.

To start the VNX scenario, move to the folder where is located the VNX file with the definition of the scenario, *vnet_cache_scenario.xml* (e.g. *<YOUR_ROOT_DIR>/vnet-sandbox/vnx/scenarios/vnet-cache-scenario*), and then execute:

$ sudo vnx -f vnet_cache_scenario.xml -v –create

After executing previous command all the Linux containers that compose the scenario starts. To configure the servers and services, and to start all the services *Ansible* is used. In the directory *ansible* where the VNX file is located, there is a playbook *vnet-sandbox.yml* to configure the scenario. When running the playbook, an output similar to the following is printed out where all the tasks executed can be checked:

$ ansible-playbook vnet-sandbox.yml

```
PLAY [test-lab] ****************************************************

TASK [Gathering Facts] *********************************************
ok: [localhost]

TASK [Add static route] ********************************************
changed: [localhost]

PLAY [caches] ******************************************************

TASK [Gathering Facts] *********************************************
ok: [c1]
ok: [c2]

TASK [Enable Reverse Proxying] *************************************
changed: [c2]
changed: [c1]

TASK [Enable Reverse Proxying - proxy.config.http.server_ports 80] ******
changed: [c1]
changed: [c2]

TASK [Configure Traffic Server to accept PUSH requests] ****************
changed: [c1]
```

```
changed: [c2]

TASK [Caching rule in Traffic Server - mpd files not cached] ************
changed: [c1]
changed: [c2]

TASK [Caching rule in Traffic Server - remove deny actions IPv4] ********
changed: [c1]
changed: [c2]

TASK [Caching rule in Traffic Server - remove deny actions IPv6] ********
changed: [c1]
changed: [c2]

PLAY [c1] ***************************************************************

TASK [Gathering Facts] **************************************************
ok: [c1]

TASK [Add rule to URL mapping config file] ******************************
changed: [c1]

TASK [Restart Apache Traffic Server (Mid-Tier Cache)] *******************
changed: [c1]

TASK [Start Mcast Receiver component] ***********************************
changed: [c1]

PLAY [os1] **************************************************************

TASK [Gathering Facts] **************************************************
ok: [os1]

TASK [Enabled mod_headers] **********************************************
changed: [os1]

TASK [Append Cache-Control header] **************************************
changed: [os1]

TASK [Create symbolic link to video repository] *************************
changed: [os1]

TASK [Restart Origin Server (apache2)] **********************************
changed: [os1]

TASK [Add multicast static route] ***************************************
changed: [os1]

TASK [Start Mcast component] ********************************************
changed: [os1]

PLAY [r1] ***************************************************************
```

```
TASK [Gathering Facts] *********************************************
ok: [r1]

TASK [Start pimd service] ******************************************
changed: [r1]

PLAY RECAP *********************************************************
c1             : ok=10       changed=8     unreachable=0failed=0
c2             : ok=7        changed=6     unreachable=0failed=0
localhost      : ok=2        changed=1     unreachable=0failed=0
os1            : ok=5        changed=6     unreachable=0failed=0
r1             : ok=2        changed=1     unreachable=0failed=0
```

Just to mention that the multicast components are started in the *Origin* and *Mid/Edge-Tier Cache servers* using a task in the playbook. For some test cases these multicast components can be started manually executing the following commands in the */shared/lib*.

In the *Origin server*:

$ java -cp vnet_dist_mcast.jar com.vnet.dist.mcast.McastOriginServer

In the *Cache servers*:

$ java -cp vnet_dist_mcast.jar com.vnet.dist.mcast.McastCacheServer

5 Conclusions and Future Work

5.1 Conclusions

Video streaming consumption is a driving force in the evolution of current networks and is leading the way for the creation of new services and the improvement of video streaming protocols. The evolution of IPTV networks plays an important role in this game. The proposed solution presented in this project adds multicast components in the video delivery network to improve network usage and user experience for the new video services that are coming.

The approach taken in the project development was to firstly analyze the state of the art in current IPTV networks, taken special attention to the network design and protocols used. To continue analyzing open-source tools and projects that could be used to build a practical scenario. As a summary, VNX tool were used to simulate the network, Ansible for configuration management and provision, Apache Web server as the Origin server, Apache Traffic server as the Mid/Edge-Tier Caches servers, GPAC and FFmpeg as video coding tools, and JGroups as communication library. With all the elements up and running in the simulated network, the design and development of the multicast components were done. The content of the multicast groups was defined, and the multicast components developed. All the code, configuration and related files are located in a public repository (GitLab) for reuse and future enhancements.

The Agile methodology was used for the development of the project and YouTrack by JetBrains as the project management tool (it has an academic license for the students of the UPM University). Project planning were divided in six sprints. The first two sprints were dedicated to the definition of the VNX scenario and configuration of OSS components. Sprint 3 and 4 focus on multicast prototyping and then the design and development of the multicast components. The memory of the project has been written in the last two sprints. Reporting of the progress has been done using YouTrack.

The project has met the objectives defined and the beginning of it, i.e. to define and develop a scenario where multicast of media segments to network elements closer the final users improve network usage and user experience. The multicast and unicast protocols combined together improve the general performance of those used independently.

The proposed solution is a running example that could be used to construct new scenarios. The use case created can be used in a lab assignment in a subject related to video streaming and/or overlay networks. The simulated network can be run in

commodity hardware not depending on specific hardware, allowing to be used by anyone with a personal laptop.

5.2 Future Work

This project can be a starting point to build video streaming scenarios, to propose enhancements or new services in the context of network design, protocol definition, and deployment of new elements.

Several lines of work can start from this project. Let's summarize some possible future works. One possible work is to stream live content using MPEG-DASH and multicasting media segments to cache servers closer to users. Latency could be reduced with this approach. And as different representations of a content are closer to the user, watching the same content with different terminals could be also synchronized in time, i.e. no delay between playout in different devices.

Another work could be to build a complete overlay network. This can be performed using the open-source software Apache Traffic Control that provides a complete CDN implementation. From this point the different components can be replaced to add new services or possible enhancements.

Monitoring the network is one overall component that could be added to enhance the analytics element that instructs the multicast component in the Origin server to broadcast contents to certain elements adapting the streaming to network capacity and then reducing possible artifacts to final users. Network monitor could allow to understand traffic and instructs other elements in the network to perform some specific task.

Extend the IPTV network to upcoming 5G network to analyze if this approach could work with users in a mobile network. Edge computing in 5G networks could bring new opportunities in the definition of new services on-the-go.

Bibliography

[1] S. Zeadally, H. Moustafa and F. Siddiqui, "Internet Protocol Television (IPTV): Architecture, Trends, and Challenges," *IEEE Systems Journal,* vol. 5, no. 4, pp. 518-527, 2011.

[2] «Telecomunicaciones y Audiovisual. Informe Económico 2017,» Comisión Nacional de los Mercados y la Competencia (CNMC), 2018.

[3] Information technology - Dynamic adaptive streaming over HTTP (DASH) - Part 1: Media presentation description and segments format, ISO/IEC, 2014.

[4] R. Hat, "Whitepaper Ansible in Depth," 2017. [Online]. Available: https://www.ansible.com/hubfs/pdfs/Ansible-InDepth-WhitePaper.pdf.

[5] K. W. R. James F. Kurose, Computer Networking, Pearson, 2012.

[6] B. Veselinovska, M. Gusev and T. Janevski, "State of the art in IPTV," *MICRO,* 2014.

[7] I. Cable Television Laboratories, "Video IP Multicast. IP Multicast Adaptive Bit Rate Architecture Technical Report," 2014.

[8] V. S. Nath and S. R. K, "Multicast ABR: A Pay TV Operator's Key to Multiscreen Live Streaming," Frost & Sullivan, 2016.

[9] B. Schwarz, "Content Delivery Networks 3.0," *A CTOiC White Paper,* 2010.

[10] G. O'Driscoll, Next Generation IPTV Services and Technologies, John Wiley & Sons, 2008.

[11] C. Timmerer, Special IssueonModernMediaTransport—Dynamic Adaptive StreamingoverHTTP(DASH), Klagenfurt, Austria: Signal Processing: Image Communication, 2012.

[12] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP - Standards and Design Principles," *ACM,* 2011.