FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Virtualização de Estúdios Móveis na Produção de Conteúdos Audiovisuais em Direto

Miguel Ferreira da Cunha Poeira

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Ademar Manuel Teixeira de Aguiar

Supervisor: Alexandre Ulisses F. Almeida e Silva

13 de junho de 2016

Virtualização de Estúdios Móveis na Produção de Conteúdo Audiovisuais em Direto

Miguel Ferreira da Cunha Poeira

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Professor António Miguel Pontes Pimenta Monteiro

Vogal Externo: Professor José Manuel de Castro Torres

Orientador: Professor Ademar Manuel Teixeira de Aguiar

---

13 de julho de 2016

Resumo

A produção de eventos televisivos em direto, pela sua natureza distribuída, requerem a disponibilização dos mais variados recursos em locais geograficamente distintos, elevando bastante os custos de produção. À medida que a performance e o custo-eficácia das redes IP aumentam, as equipas de produção poderão beneficiar destes avanços tecnológicos para produzir e realizar, em tempo real, produções com a qualidade esperada num ambiente profissional.

Da mesma forma, a evolução das tecnologias de virtualização na *cloud* e a investigação realizada para desenvolver soluções que fornecem ambientes resilientes e controlados, permite que esta camada tecnológica possa ser considerada para suportar operações *time-critical*, tal como a produção de conteúdos audiovisuais em direto.

Esta dissertação aborda a produção de conteúdos televisivos em direto, propondo uma arquitetura de alto nível para a virtualização da produção de conteúdos utilizando estúdios móveis.

Utilizando a transmissão de vídeo e áudio descomprimido através do protocolo RTP, pretende-se demonstrar que a produção televisiva de conteúdos audiovisuais em direto pode aproveitar as potencialidades que a *cloud* oferece. De forma a demonstrar essas potencialidades para realizar operações de forma distribuída, o protótipo desenvolvido permite, em tempo real, receber áudio e vídeo, transportá-los descomprimidos numa rede local e gerar versões de baixa resolução para uma interface *web,* a partir da qual o realizador as poderá visualizar.

O trabalho realizado demonstra a possibilidade de estender a arquitetura utilizada a outras operações na cadeia de produção televisiva, como, por exemplo, a comutação entre várias entradas de vídeo e áudio, aproximando-se do tão idealizado "*IP Studio*".

Abstract

Live TV production, due to its distributed nature, requires broadcasters to deploy equipment and human resources to several different places, increasing production's costs. As the performance and cost-effectiveness of IP networks grow, broadcasters can use this to handle real-time production-quality video and audio that is expected in a professional environment.

Similarly, the evolution of virtualization's technologies on the cloud and the efforts developed in order to provide solutions of virtualized, elastic, controllable cloud environments, it enables that this technologic stack may be considered for time-critical applications, such as a live TV production.

This master thesis proposes a high architecture for virtualizing live TV production.

By using an extended version of the RTP protocol to transport audio and video uncompressed feeds through IP networks, it ought to show that live TV production can be virtualized into the cloud, taking advantage of its benefits. While using cloud's benefits to perform time-critical operations in a distributed manner, the developed prototype allows a live remote production team to receive audio and video streams, multicast them locally at an uncompressed quality and publish them as web feeds to an external web interface.

The work developed shows an architecture that can be potentially used to perform other operations that may occur in a live TV workflow, such as video switching, getting closer of the so coveted "IP Studio".

Agradecimentos

Gostaria de agradecer a todos os que tornaram esta dissertação possível, desde a MOG Technologies e os seus colaboradores, ao meu orientador, Professor Ademar Aguiar, por aceitarem este desafio.

Tenho que agradecer, com especial atenção, ao Eng. Pedro Ferreira, ao Eng. Alexandre Ulisses e ao Eng. Daniel Costa pela disponibilidade que demonstraram, pelo espírito crítico e pelo conhecimento transmitido, que permitiu, sem dúvida, enriquecer esta dissertação.

Ao Eng. Victor Fernandes, que contribuiu criticamente para o melhoramento da informação visual presente nesta dissertação.

Um agradecimento especial ao Eng. Pedro Santos, que acompanhou de perto todo o desenvolvimento da dissertação, pela paciência, dedicação, empenho e interesse que demonstrou.

Miguel Poeira

"*There is always three ways in. There's the First Door,*
where 99% of people wait in the line, hoping to get in.
There's the second door, where billionaires and royalty slip through.
But then there is always, always… the Third Door.
It's the entrance where you have to jump out of line,
run down the alley, climb over the dumpster,
bang on the door a hundred times, crack open the window,
and sneak through kitchen. But there's always a way in."

Alex Banayan

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

API	Application Programming Interface
CRUD	Create, Read, Update, Delete
DASH	Dynamic Adaptative Streaming over HTTP
ES	Elementary Stream
HD	High Definition
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JT-NM	Joint Task Force on Networked Media
MCR	Master Control Room
MPD	Media Presentation Description
MPEG-TS	MPEG Transport Stream
MPTS	Multiple Program Transport Stream
NMOS	Networked Media Open Specifications
OB	Outside Broadcasting
PAT	Program Association Table
PCR	Program Control Room
PES	Packetized Elementary Stream
PID	Packet ID
PMT	Program Map Table
PTP	Precision Time Protocol
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
RFC	Request for Comments
RTCP	RTP Control Protocol
RTP	Real-time Transport Protocol
SD	Standard Definition
SDI	Serial Digital Interface
SLA	Service-Level Agreement
SMPTE	Society of Motion Picture and Television Engineers

SPTS	Single Program Transport Stream
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VOD	Video on Demand
XML	eXtensible Markup Language

Capítulo 1

Introdução

A origem dos programas televisivos pode ser abrangida por uma de três grandes categorias: produção remota, em estúdio ou pré-produzido. Aqui existe, logo à partida, a grande diferença de que as duas primeiras compreendem emissões em direto, enquanto a terceira categoria abrange programas previamente gravados. Não obstante, a produção de um programa televisivo, qualquer que seja a sua natureza, requer uma gestão logística e tecnológica de crescente complexidade [1].

1.1 Contexto

A produção de eventos televisivos em direto, pela sua própria natureza, requer requisitos bastante restritos: a entrega de vídeo e áudio com o mínimo de atraso possível, mantendo os requisitos de qualidade e segurança que a indústria televisiva requer para garantir a qualidade de experiência (QoE, em inglês) aos telespectadores.

Figura 1:*OB Van*.

O método atual, que vários autores chamam de tradicional, baseia-se na produção remota com base em camiões de produção móveis, ou como é chamado na gíria, *Outside Broadcasting* (OB) *vans* [2], [3]. Uma OB *van,* como a da Figura 1, pode custar vários milhões de dólares [2], principalmente devido à grande quantidade e diversidade de equipamento que contém. Em muitos casos pode ser necessária a utilização de várias unidades, principalmente pela natureza distribuída dos eventos em direto [4], como por exemplo a cobertura de um ato eleitoral. Esta necessidade, assim como a ligação satélite que permite a conexão entre a OB *van* e o estúdio televisivo, eleva os custos e a complexidade da produção televisiva [5].

Apesar de um estúdio de produção moderno já utilizar extensivamente o protocolo IP, a sua utilização permite principalmente guardar e arquivar materiais multimédia para que estes estejam disponíveis ao longo de toda a cadeia de produção [6]. Da mesma forma, esta infraestrutura permitiu servir os espetadores com serviços de *video on demand* (VOD) como o Netflix e o Hulu [6]. Apesar disso, esta é apenas uma pequena parte do *workflow* na produção televisiva.

Com o desenvolvimento da arquitetura dos serviços *cloud* e o aparecimento de *frameworks* para desenvolver e operar operações *time-critical*, como a produção televisiva de eventos em direto, e o aumento da *performance* das redes, passa a ser possível pensar num IP *Studio*, onde todo o *workflow* destes eventos possa estar virtualizado na *cloud*, aproveitando todos os benefícios que esta oferece, apesar dos riscos inerentes à *Quality of* Service (QoS).

1.2 Enquadramento

Esta dissertação foi realizada em ambiente empresarial, na empresa MOG Technologies, que desenvolve soluções para, principalmente, ambientes de pós-produção. A MOG está no mercado do *broadcasting* há mais de 10 anos a fornecer soluções que permitem a interoperabilidade entre equipamentos utilizados na indústria.

Esta dissertação enquadra-se na área de investigação da empresa, com o intuito de abranger outras áreas da produção televisiva no seu leque de produtos.

1.3 Motivação e Objetivos

Com a virtualização da cobertura televisiva de eventos em direto pretende-se eliminar (ou pelo menos reduzir) a utilização da OB *van* e das ligações por satélites e substituí-las por uma aplicação distribuída na *cloud* suportada pela transmissão de vídeo sobre IP [2], [7]. Assim, as várias *streams* que resultam da cobertura do evento serão enviadas, via IP, para uma aplicação distribuída na *cloud*. Essa aplicação será responsável por fornecer ao realizador uma interface, através de uma aplicação web, que lhe permita comutar entre as várias *streams* de entrada, mais comumamente chamada de *Vision Mixer*.

O trabalho a realizar visa desenvolver um protótipo de uma aplicação distribuída que permita controlar as várias *streams*, assim como a sincronização de cada *stream* (apesar do assincronismo da transmissão de vídeo/áudio por IP). Adicionalmente, deve ser possível visualizar este protótipo através de uma aplicação web. Sendo uma aplicação distribuída na *cloud*, espera-se que exista modularidade e expansibilidade, de forma a ajustar a solução aos mais diferentes e variados cenários. Para tal, será necessário implementar uma arquitetura modular e genérica, que permita a interação entre os vários intervenientes e componentes de um estúdio móvel.

É, também, essencial que, tratando-se de um método que visa substituir o atualmente utilizado, se garanta a fiabilidade e qualidade que estes últimos oferecem.

1.4 Estrutura da Dissertação

Para além da introdução, este documento possui mais 7 capítulos. No capítulo 2 analisam-se os modelos de *cloud*, de que forma a virtualização permite servir os diferentes tipos de modelos, *Hypervisors* e *Containers Engines* e quais as vantagens de ter um produto modular na *cloud*.

No capítulo 3 é estudada a camada de transporte do modelo TCP/IP, compreendendo as diferentes possibilidades de transmissão de dados sobre IP.

O capítulo 4 propõe-se a analisar a produção de Televisão Digital, explicando qual o ciclo de vida dos conteúdos numa cadeia de produção televisiva, de forma a se analisar os vários *workflows* e de que forma é feito o transporte dos conteúdos.

No capítulo 5 revê-se casos de estudo na produção audiovisual em direto sobre IP, assim como se estuda o trabalho existente para permitir a interoperabilidade na transição de SDI para IP. Termina com as conclusões que podemos extrair.

O capítulo 6 explica as restrições e problemas do método atual na produção móvel de eventos em direto, analisando os requisitos que uma possível solução terá que garantir. Assim, propõe uma arquitetura modular para uma aplicação distribuída na *cloud*. Com base na arquitetura proposta, apresenta-se o protótipo desenvolvido e as decisões arquiteturais por detrás do mesmo.

O capítulo 7 apresenta o conjunto de testes de validação e aceitação que foram realizados, de forma a validar o protótipo desenvolvido, assim como se analisa o resultado desses testes e se discute a sua exequibilidade.

A dissertação termina com o capítulo 8, onde é feita uma apreciação geral do trabalho realizado, o cumprimento dos objetivos e se perspetiva trabalho futuro.

Capítulo 2

Computação na *Cloud*

A computação na *cloud* é um serviço *on-demand* que permite desenvolver, distribuir e gerir aplicações, providenciando a cada aplicação um ambiente estável e isolado, sem que esta tenha conhecimento que está a correr sobre *hardware* virtual. A tecnologia por detrás de cada serviço *cloud* pode ser bastante variada, desde *software* proprietário a soluções *open-source*. Apesar disso, os princípios e capacidades de um e outro caso são semelhantes: máquinas virtuais podem ser planeadas, criadas e configuradas em ambientes perfeitamente isolados uns dos outros; o *hardware* que suporta essas máquinas é, em verdade, mais poderoso do que aquilo a que a máquina virtual tem acesso, dado que várias máquinas virtuais podem correr na mesma máquina física [8].

A *cloud* baseia-se num modelo por camadas, onde cada camada corresponde a um maior nível de abstração oferecido à sua camada superior (Figura 2) [8]:

- A camada do servidor são os elementos que são realmente físicos numa *cloud*.
- A camada da infraestrutura abstrai os elementos físicos através da virtualização do *hardware*, utilizando máquinas virtuais. Esta camada fornece ferramentas para criar, gerir, iniciar e parar máquinas virtuais.
- A camada da plataforma utiliza as ferramentas da camada da infraestrutura de forma a simplificar a configuração e criação da infraestrutura necessária (memória, processamento, entre outros) de uma forma automática.
- A camada da aplicação permite implementar e manter uma aplicação na *cloud* de uma forma centralizada, permitindo que esta possa ser administrada remotamente. Utiliza a camada da plataforma de forma a prover-se de elasticidade para se adaptar à sua maior ou menor utilização. Isto é: num período de grande utilização, por exemplo, a camada da aplicação reserva mais recursos, de forma a manter a QoE do utilizador final inalterada.

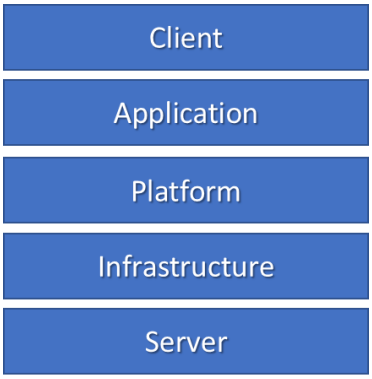

Figura 2: Modelo de camadas da *cloud*.

- A camada do cliente é através da qual o utilizador final interage com a aplicação. Pode ser uma aplicação móvel ou um *website*, por exemplo. As abstrações que as camadas anteriores introduzem, permitem que o cliente se deixe de preocupar com compatibilidades, instalação, manutenção e *upgrades* ao programa, assim como o liberta da necessidade de alocar recursos (espaço em disco, por exemplo) para o programa.

A virtualização significa, portanto, introduzir uma abstração entre duas camadas, a partir da qual as camadas superiores deixam de ter conhecimento de como os recursos estão a ser distribuídos ou partilhados [9].

2.1 Modelos

Dependendo do nível de abstração, os serviços da *cloud* são oferecidos em vários modelos. Se a abstração for colocada ao nível da camada da infraestrutura, está-se perante um serviço *Infrastructure-as-a-Service* (IaaS), permitindo utilizar e alugar recursos consoante as necessidades, sendo apenas cobrados os recursos que são efetivamente utilizados.

Pode-se falar em *Platform-as-a-Service* (PaaS) quando a abstração é colocada na camada da plataforma, abstraindo os servidores e criando um ambiente focado na aplicação.

Software-as-a-Service é um modelo de entrega de *software* através da *cloud*, eliminando a necessidade de o instalar na máquina do utilizador final. Através deste modelo, a aplicação pode ser gerida, mantida e atualizada remotamente, de forma completamente transparente para os utilizadores finais.

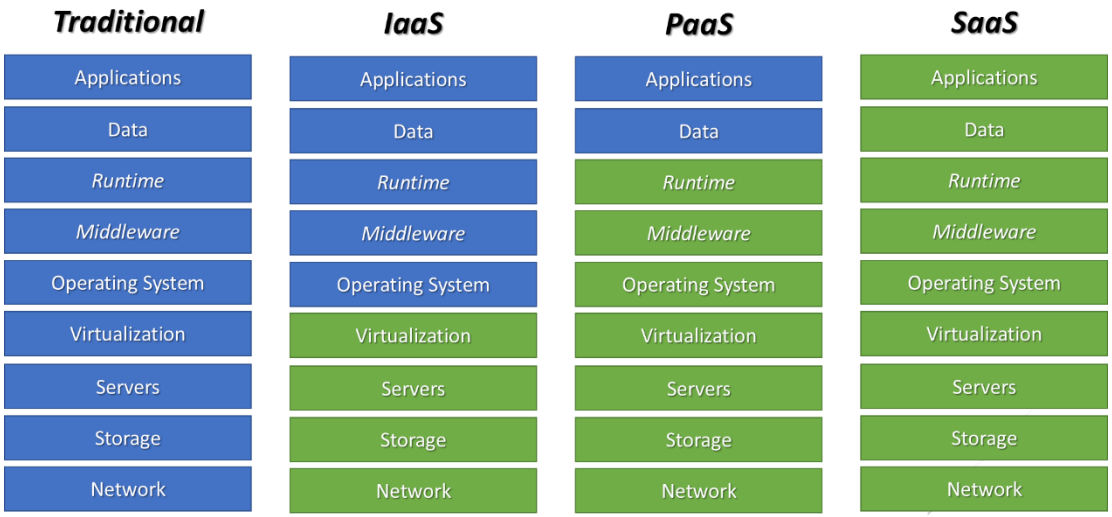

Figura 3: Arquitetura tradicional e na Cloud.

Como se pode ver na Figura 3 (a verde os componentes que são geridos pelo provedor do serviço *cloud*; nos componentes a azul, a gestão cabe ao utilizador), à medida que o nível de virtualização na *cloud* aumenta, o utilizador tem menos controlo das camadas mais específicas (baixo nível). Consegue-se, ainda, facilmente mapear os componentes da Figura 2 com o da Figura 3, dado que a esta última materializa as camadas definidas pela primeira.

2.2 *Hypervisors e Containers*

A virtualização dos recursos implica a partilha dos mesmos. A implementação deste nível de abstração baseia-se na utilização de *software* que respeite duas regras essenciais: isolamento e suporte para múltiplos inquilinos (*multitenancy*, em inglês). Para tal, duas tecnologias são abrangentemente utilizadas: *hypervisors* e *container engines*. Apesar de servirem o mesmo propósito, a grande diferença entre ambas encontra-se na sua arquitetura [10].

Tal como se pode observar na Figura 4, um *Hypervisor* permite o aprovisionamento de aplicações em máquinas virtuais, que podem correr diferentes sistemas operativos. Isto significa que, para além da aplicação e das suas dependências, cada máquina virtual inclui, também, o sistema operativo que é executado. Em contrapartida, um *Container Engine* permite o aprovisionamento de uma aplicação num *container*, que partilha o sistema operativo do *host* com os restantes *containers*. Isso significa que cada *container* contém apenas a aplicação e as suas dependências e, como tal, tem um tamanho muito mais reduzido que uma máquina virtual, tornando-se possível aprovisionar um maior número de *containers* do que de máquinas virtuais num determinado *host*. Da mesma forma, reiniciar um *container* não implica reiniciar o sistema operativo [10].

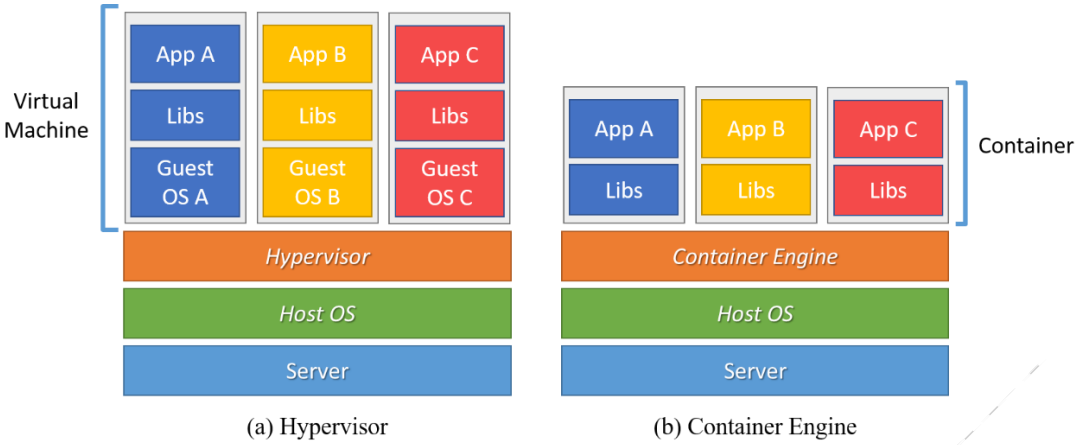

Figura 4: Hypervisor vs. Container Engine.

O Docker[1] é um projeto *open source* que permite construir, aprovisionar e correr aplicações dentro de *containers*. O Docker Engine tira partido de várias funcionalidades do *kernel* do Linux, de forma a prover cada *container* de um ambiente isolado, com um *start time* rápido e que ocupa menos recursos que uma máquina virtual.

A partir de um documento de configuração, o Dockerfile, o Docker é capaz de executar comandos (instalar dependências, expor portas, configurar variáveis de ambiente, por exemplo) incrementalmente, em camadas,

Quando comparado com as máquinas virtuais, *containers* como o Docker beneficiam do isolamento e da alocação de recursos já existente nas máquinas virtuais, mas são mais portáveis e eficientes [11]. Tudo isto torna os *containers* Docker facilmente transferíveis entre *hosts*.

2.3 Workflow Management System

A *cloud* providencia um ambiente virtualizado, controlável, elástico e com serviços *on-demand* de alta qualidade, proporcionando o aparecimento de aplicações distribuídas de grande complexidade [4]. Apesar disso, tanto [12] como [13] consideram que, devido à natureza das aplicações *time-critical*, que requerem a capacidade da sua estrutura se reconfigurar dependendo das circunstâncias em que se encontram, os serviços *cloud* geralmente não fornecem explicitamente QoS para as capacidades da rede. Da mesma forma, para garantir a resiliência de um sistema distribuído, é necessário que existam *workflows* que programaticamente monitorizem métricas específicas, em tempo real, e tomem ações no sentido de manter (ou repor) a QoS e QoE.

Os *Workflow Management Systems* (WMS) permitem modelar e executar processos em ambientes locais ou distribuídos, tendo como objetivo criar uma estrutura reconfigurável durante

[1] http://docker.com/

a execução de uma aplicação, tendo em conta não só os restrições de qualidade impostas pelo *Service-Level Agreement* (SLA), como também a lógica de negócio que é inerente à aplicação [12].

Apesar de existirem vários WMS, como o Apache Taverna[2], o Triana[3] e o Pegasus[4], nenhum deles permite compor cenários *data-driven* onde exista dependência de processos [12].

O *Software Workbench for Interactive, Time Critical and Highly self-adaptative cloud applications* (SWITCH)[5] é um WMS que permite definir, planear e modelar ambientes virtuais, com foco para aplicações *time critical* com grande fluxo de comunicações de dados. O SWITCH reúne métricas da aplicação, compreende a sua semântica e negoceia, em tempo real, condições de SLA com o provedor da *cloud* [4], [12].

O SWITCH compreende 3 sub-sistemas autónomos, mas interdependentes, que trabalham em conjunto para:

- Desenhar e planear o ambiente de execução, proporcionando ferramentas interativas para tal.
- Preparar o sistema de execução da aplicação e o ambiente planeado anteriormente.
- Monitorizar o estado da aplicação em tempo real e, sempre que possível, toma medidas autonomamente de forma a manter o QoS acordado.

2.4 Vantagens da *Cloud*

Ao utilizar os serviços que a *cloud* oferece é possível desenvolver uma solução escalável, eficiente e que está disponível em qualquer computador, independentemente do sistema operativo que este utiliza. Da mesma forma, a natureza distribuída das aplicações na *cloud* permitem introduzir *upgrades*, *patches* e melhorias no serviço de uma forma simples, dado que o utilizador final é abstraído destas mudanças [7], [14].

Algumas aplicações, como o Google Docs[6], aproveitam as tecnologias *cloud* e a centralização da aplicação na rede para permitirem o acesso e edição simultânea na mesma aplicação, permitindo aos utilizadores novas formas de trabalhar e colaborar [2].

A redução de custos que a arquitetura da *cloud* e do modelo de negócio que lhe é inerente é outra das vantagens a enumerar. O princípio de partilha de recursos físicos (através da virtualização) permite dividir custos que antigamente teriam que ser suportados apenas por uma entidade [8].

[2] http://taverna.incubator.apache.org/
[3] http://trianacode.org/
[4] https://pegasus.isi.edu/
[5] http://www.switchproject.eu/
[6] https://www.google.com/docs/about/

Assim, ao se requerer um serviço na *cloud*, é feito um contrato que define os requisitos que o provedor da *cloud* se compromete a fornecer, o SLA, sendo estes pagos com base na utilização, tendo em conta a número e a complexidade dos serviços oferecidos [14].

Da mesma forma, os principais vendedores de serviços *cloud* já suportam *containers*, como o Docker, e ferramentas que permitem a coordenação, orquestração e escalamento desses *containers* [15]. Assim, a *cloud* apresenta-se como uma solução viável, dado que permite criar ambientes perfeitamente customizados às necessidades de cada aplicação, adaptando *on-demand* a infraestrutura às características requeridas para o bom funcionamento do serviço que disponibiliza.

Capítulo 3

Transmissão de dados sobre IP

O modelo TCP/IP é um modelo que define um conjunto de protocolos de comunicação em rede. Este é composto por cinco camadas (Figura 5), onde cada camada é responsável por fornecer serviços aos protocolos que se encontram na camada superior, criando níveis de abstração.

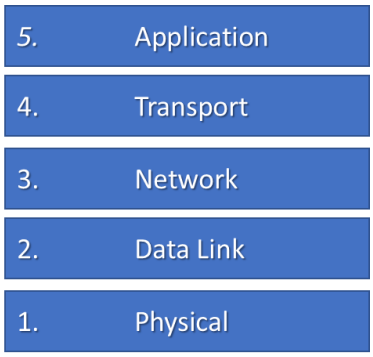

Figura 5: Modelo TCP/IP.

No âmbito desta dissertação importa analisar as duas camadas superiores, ou seja, a camada de transporte e a camada da aplicação. A camada de transporte fornece canais de comunicação para transmissão de dados entre diferentes máquinas, independentemente da logística utilizada para a sua transmissão. Esta camada permite endereçar portas para determinadas aplicações.

Por seu lado, a camada da aplicação fornece serviços que, ao utilizarem a camada de transporte, permitem a processos comunicarem entre si. Para que cada processo seja identificável, este necessita de registar portas de que está à “escuta”.

3.1 A camada de Transporte

O *Transmission Control Protocol* (TCP) é um protocolo orientado à conexão da camada de transporte do modelo TCP/IP. O TCP fornece a garantia de que todos os pacotes entre um emissor e um recetor são entregues de forma ordenada. Ao estabelecer a ligação, o TCP estabelece um pré-acordo *Three-Way Handshake*, assegurando uma conexão entre o emissor e o recetor. Durante a transferência de dados, o recetor vai enviando mensagens de *Acknowledgement*, confirmando a receção de um pacote de dados. Cada pacote de dados é anotado com um número sequencial, permitindo ao recetor reconstruir, se necessário, a ordem correta dos pacotes de dados.

O *User Datagram Protocol* (UDP), por sua vez, é um protocolo não orientado à conexão. Isto significa que o UDP não garante a entrega de todos os pacotes de dados, nem a sua ordenação, já que os pacotes de dados são independentes entre si. Ao contrário do TCP, o UDP transfere cada pacote de dados uma vez. Os pacotes que chegarem corrompidos ao recetor são descartados, sem que o emissor tenha sequer conhecimento.

O UDP é um protocolo mais simples, indicado para aplicações que necessitam de uma transmissão rápida dos dados. A ausência de estruturas de controlo complexas permitem ao UDP eliminar o *overhead* que está inerente ao TCP, garantindo-lhe alta eficiência.

Apesar do protocolo UDP não garantir a entrega dos pacotes de dados, esta tarefa pode ser delegada, se necessário, à aplicação. Assim, o UDP constitui uma alternativa ao TCP, principalmente onde o serviço das camadas inferiores são bastantes confiáveis, como uma rede local.

3.2 Real-time Transport Protocol

O *Real-time Transport Protocol* (RTP) [16] é um protocolo da camada 5 do modelo TCP/IP utilizado para entregar áudio e vídeo sobre redes IP. Enquanto o RTP é utilizado para transmitir *streams* multimédia, o RTP *Control Protocol* (RTCP) é utilizado, em simultâneo, para monitorizar estatísticas da transmissão, QoS e controlo de sessões.

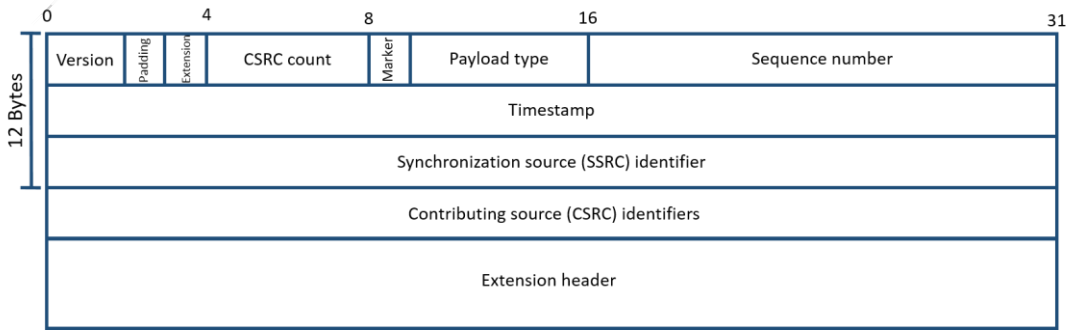

Figura 6: Formato de um pacote RTP.

- *Version* (2 bits): indica a versão do protocolo RTP.
- *Padding* (1 bit): indica a existência de *padding* no final do *payload* do pacote RTP.
- *Extension* (1 bit): se ativo, indica a existência de extensão no cabeçalho do pacote.
- *CSRC count* (4 bits): indica o número de identificadores *Contributing Source* presentes após o cabeçalho fixo.
- *Market* (1 bit): permite a definição, por parte de perfis, de eventos, como, por exemplo, o fim de uma *frame* ou o modo de varrimento da *stream*.
- *Payload Type* (7 bits): indica o tipo de dados presentes no *payload* do pacote, tendo em conta os vários perfis existentes.
- *Sequence Number* (16 bits): número de sequência controlado pelo emissor, permitindo a ordenação sequencial de pacotes.
- *Timestamp* (32 bits): referência temporal dados dados transportados no *payload* do pacote; pacotes diferentes contento informação do mesmo *frame* devem ter o mesmo *timestamp*.
- *Synchronization Source* (32 bits): indica a fonte de contribuição do T*imestamp* e do *Sequence Number*; todos os pacotes de um mesmo SSRC fazem parte da mesma linha temporal.
- *Contributing Source* (32 bits, cada): indica os SSRC de todas as fontes contribuidoras para o conteúdo do *payload* do pacote.

O protocolo RTP fornece informação dos *timestamps* dos pacotes, números de sequência e do formato de codificação dos dados. Para além disso, o cabeçalho dos pacotes RTP contém um campo destinado à extensão do cabeçalho, com novos campos personalizados, tornando o RTP num protocolo flexível e facilmente extensível.

A existência do *timestamp* e do número de sequência nos cabeçalhos do RTP permite a um recetor reconstruir a sequência correta de pacotes, assim como estimar quantos pacotes foram perdidos.

Apesar de ter sido desenhado para ser independente de que protocolo é utilizado no transporte, as implementações de RTP focam-se na utilização do protocolo UDP, em detrimento do TCP, devido ao seu propósito *real-time*, isto é, entrega dos dados da forma mais rápida possível.

3.3 Dynamic Adaptative Streaming over HTTP

O *Dynamic Adaptative Streaming over HTTP* (DASH) [17] é uma técnica de *streaming* de conteúdo multimédia utilizando um servidor de *Hypertext Transfer Protocol* (HTTP), que faz parte da camada 7 do modelo OSI. Apesar do DASH ser agnóstico às camadas por baixo da camada da aplicação, este utiliza o protocolo TCP para transporte dos dados.

Ao partir o conteúdo multimédia em pequenos segmentos, o DASH permite servir o cliente com pequenos segmentos de informação, em diferentes representações, permitindo que essas representações sejam selecionadas de acordo com características como a largura de banda disponível, a resolução do ecrã, o tipo de dispositivo e as preferências do utilizador.

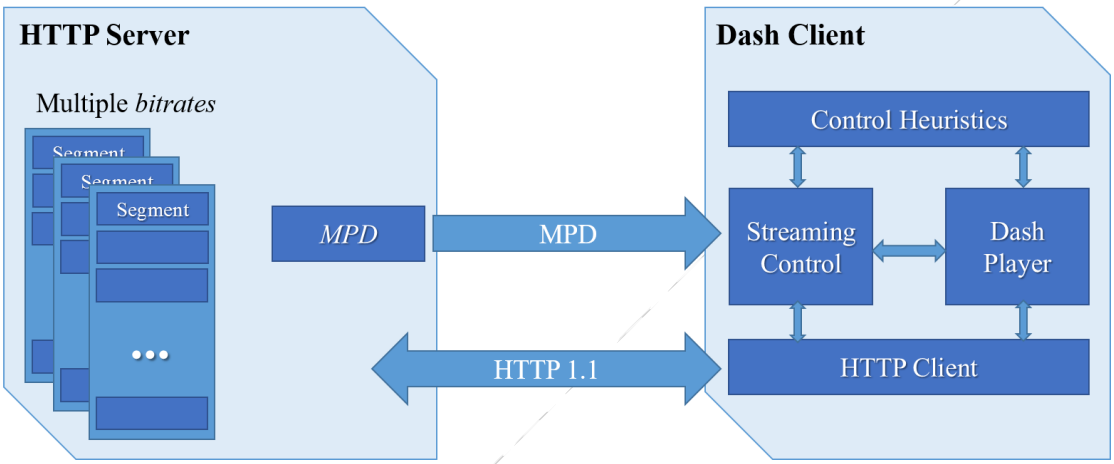

Figura 7: Arquitetura geral do DASH.

Na prática, o servidor disponibiliza o conteúdo em vários *bit rates* diferentes. No início de uma sessão, um cliente DASH requer ao servidor um *Media Presentation Description* (MPD), um manifesto em *eXtensible Markup Language* (XML) que descreve o conteúdo disponível (e os metadados a ele associado). À medida que o cliente DASH vai fazendo *buffering* e reproduzindo o conteúdo, vai também analisando a variação da largura de banda da rede e, dependendo do resultado dessa análise, decide quais os próximos segmentos a descarregar (com maior ou menor *bit rate*) de forma a manter um *buffering* adequado.

Uma latência reduzida é obtida utilizando segmentos de pequena duração e um número reduzido de segmentos em *buffer*. No caso oposto, uma maior estabilidade pode ser conseguida utilizando um *buffer* com mais segmentos. Esta troca entre estabilidade e latência baixa deve ser analisada caso a caso: num cenário em direto (*live*) normalmente pretende-se priorizar o pequeno desfasamento entre o tempo de origem e o tempo de transmissão, enquanto num cenário VOD a estabilidade da transmissão (sem pausas) será o principal objetivo.

Em resumo, o DASH utiliza um algoritmo que adapta o *streaming* ao *bit rate* máximo que as condições de rede permitem, de forma a entregar ao utilizador conteúdo com o QoE esperado.

3.4 Difusão da informação

A difusão da informação numa rede pode ter diversos intervenientes, dependendo do diferente número de recetores a que uma mensagem se destina. Se uma mensagem tiver como destinatário um único nó numa rede, então trata-se de uma comunicação *unicast*. No caso da mensagem se destinar a ser difundida por todos os elementos da rede, então temos um timo de comunicação em *broadcast*, um "anúncio" para a rede.

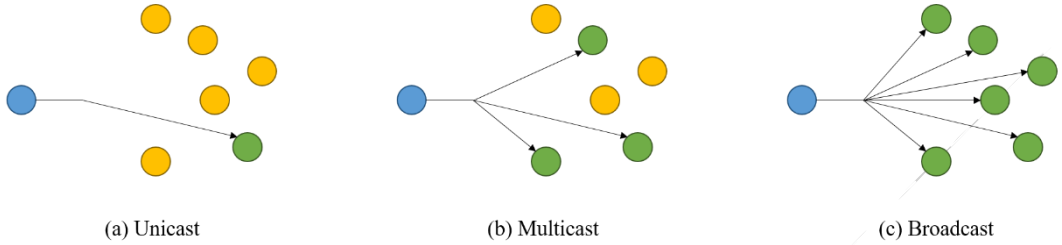

Figura 8: Unicast, multicast e broadcast.

O *multicast*, por sua vez, refere-se à técnica de enviar mensagens para todos os nós na rede que se mostraram interessados em receber mensagens desse tópico. Para tal, todos os nós interessados devem enviar mensagens *join* e *leave* para se juntarem, ou saírem, de um grupo (endereço) *multicast*. Por sua vez, um emissor pode enviar uma mensagem para esse endereço *multicast*, possibilitando a difusão da mensagem a todos os nós interessados sem que o emissor tenha conhecimento prévio de quantos recetores existem na rede. Isto permite que ao longo da execução de um determinado sistema, os recetores entrem em e saiam do grupo, sem que o emissor tenha que alterar o seu modo de funcionamento. Da mesma forma, reduz complexidade do lado do emissor, ao não o obrigar a gerir os recetores existentes.

Capítulo 4

Produção Televisiva Digital

Durante muito tempo os consumidores estavam satisfeitos por utilizar um *hardware* específico para uma única função. Mas a aplicação da tecnologia digital à produção, distribuição e consumo de conteúdos multimédia permitiu que este conteúdo pudesse ser consumido em qualquer lugar, a qualquer hora e em qualquer dispositivo [1].

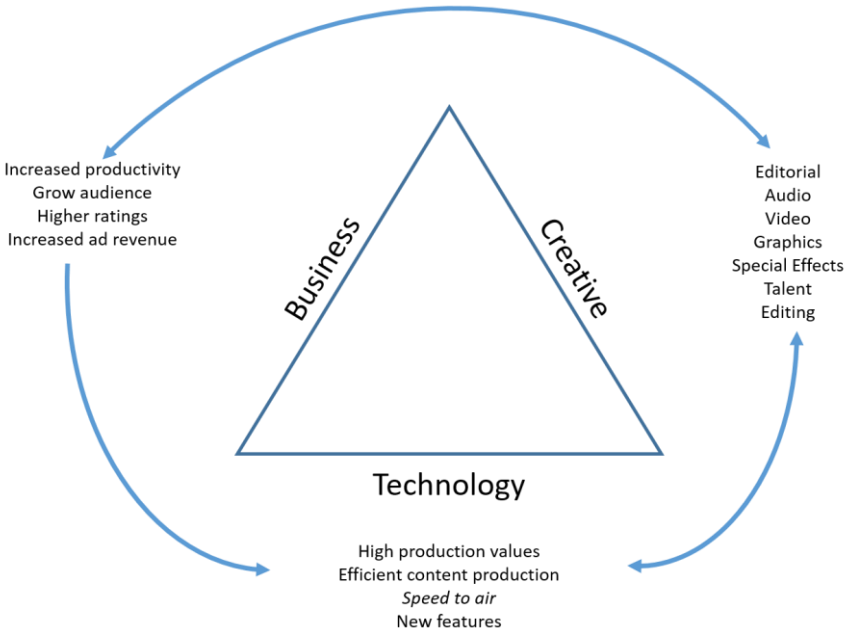

Figura 9: Triângulo da indústria multimédia [1].

A produção de televisão digital num cenário multiplataforma requer a resposta a desafios tecnológicos, criativos e de negócio, o chamado "triângulo da indústria multimédia" (Figura 9).

Esta perspetiva implica construir uma infraestrutura multiplataforma de produção de conteúdos que suporte a distribuição das mais variadas tecnologias para diferentes plataformas, onde, cada uma delas, contém funcionalidades e propósitos distintos [1].

A infraestrutura de um estúdio de produção televisiva é, na verdade, um sistema de sistemas que se devem integrar e cuja interoperabilidade entre os diferentes dispositivos deve ser assegurada [1]. Aliás, a grande diversidade de equipamentos, formatos, arquiteturas e, generalizando, diferentes formas de operar das várias tecnologias presentes num estúdio de produção televisiva, elevam os custos e o *overhead* na produção [5], [8].

Esta dificuldade em introduzir novas tecnologias na cadeia de produção impede os *broadcasters* de aceder a novos mercados de uma forma eficaz e eficiente: o custo de suportar um novo canal de distribuição e preenchê-lo com conteúdos é elevado; como tal, o preço do espaço publicitário aumenta para um valor mais elevado do que aquele que os anunciantes, dos quais o canal depende, estão dispostos a pagar. Desta forma, gera-se uma dependência cíclica na *supply chain* [8].

4.1 Ciclo de Vida dos Conteúdos

O ciclo de vida dos conteúdos multimédia compreende quatro fases essenciais: criação, montagem, distribuição e consumo [1] (Figura 10).

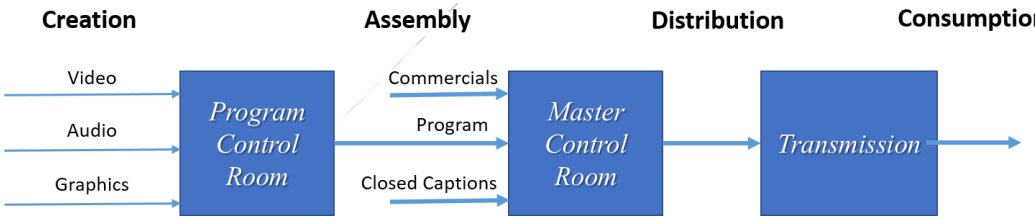

Figura 10: Ciclo de vida genérico para conteúdos televisivos [1].

Inicialmente os conteúdos (vídeo, áudio, gráficos) são produzidos (criação); de seguida, devem ser montados de forma a se obter a visão idealizada para determinado programa (montagem); o sinal deve, então, ser transmitido para os vários canais de consumo (distribuição) e, finalmente, consumido pelos vários dispositivos que o utilizador final possui.

No âmbito desta dissertação, analisemos com mais detalhe as duas primeiras fases.

Seja áudio, vídeo ou elementos gráficos, todos eles contribuem para a criação de uma "história" e de uma experiência imersiva. A criação significa, tal como a própria palavra indica, a conceção criativa de uma ideia. Para tal, um estúdio de produção necessita de estar devidamente equipado (câmaras, microfones, luzes, por exemplo) e configurado (ligações entre equipamentos e aplicações que os controlam e gerem, por exemplo) [1].

Isto significa que o desafio de criar um programa novo, original, que cative a audiência, está não só relacionado com a conceção criativa da ideia, mas também com as capacidades que o estúdio de produção disponibiliza.

Concluindo: os *broadcasters* enfrentam o desafio de produzir conteúdo de alta qualidade com acesso a recursos limitados. Dessa forma, é importante que os *workflows* e a infraestrutura permitam, de forma eficiente, a criação de conteúdo multimédia para as várias plataformas de consumo.

4.2 Workflows e Infraestrutura

Apesar da infraestrutura de um estúdio de produção televisiva ser bastante complexa, as interações de alto nível podem ser descritas como mostra a Figura 11. Os conteúdos chegam ao estúdio de produção de forma remota, ou foram gravados previamente e já estão armazenados. O grafismo é produzido, o áudio e o vídeo são misturados no *Program Control Room* (PCR). No *Master Control Room* (MCR) são adicionadas as legendas, logótipos e os anúncios publicitários. O sinal do programa é, então, modulado e transmitido [1].

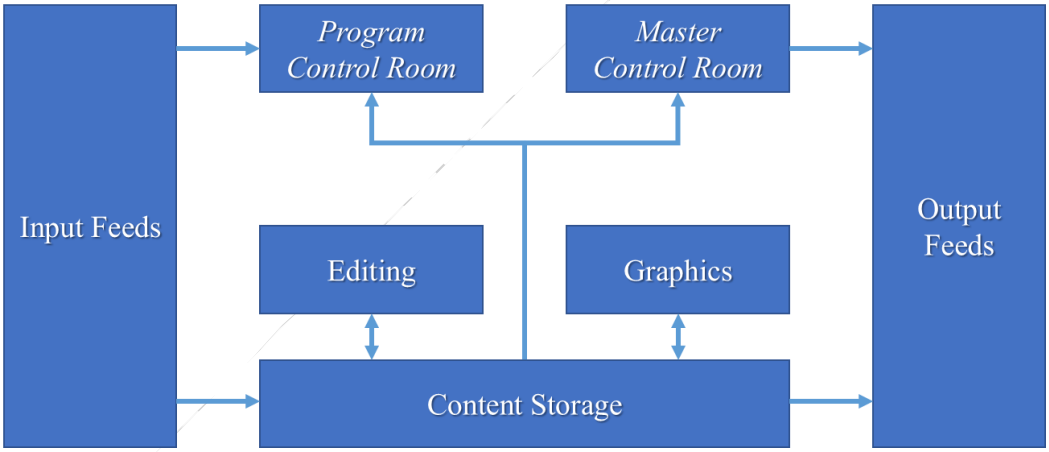

Figura 11: Diagrama funcional da infraestrutura de produção televisiva [1].

Apesar das interações na infraestrutura de um estúdio de produção televisiva parecerem triviais, a necessidade dos *broadcasters* de atingirem um público alargado é um desafio de articulação de conteúdos, equipas e equipamentos. Interessa, portanto, compreender quais os *workflows* que permitem a produção de conteúdos multiplataforma, para os vários ecrãs, formatos e dispositivos existentes.

4.2.1 Workflow Linear

Num *workflow* linear (Figura 12) o conteúdo é criado e formatado de forma independente para cada canal específico. Isto significa não só esforços redobrados para produzir o mesmo conteúdo para diferentes canais, como também duplicação de recursos ao longo da cadeia de produção.

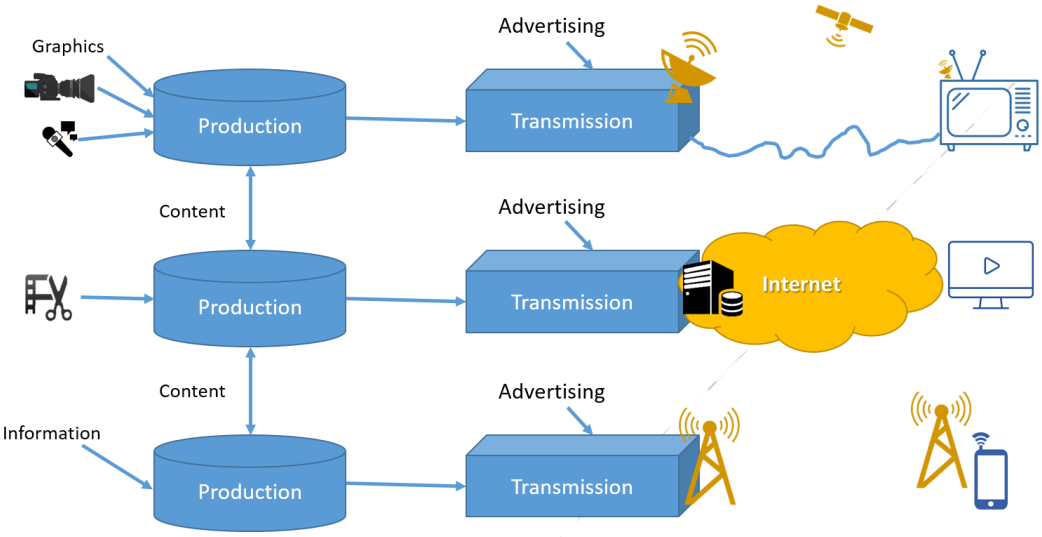

Figura 12: Produção independente e específica a cada plataforma e canal [1].

Um *workflow* linear implica uma produção específica a cada plataforma e canal de distribuição, ou seja, para cada nova plataforma ou canal de distribuição é necessário replicar a infraestrutura existente. Esta redundância deve-se, principalmente, à falta de interoperabilidade do equipamento existente e da pouca flexibilidade da infraestrutura na adoção de novas tecnologias, *workflows* e formatos [5].

4.2.2 Workflow Convergente

Obviamente que um *workflow* linear não é escalável, pois à medida que novas tecnologias (plataformas, canais de distribuição, formatos, *codecs*) vão aparecendo, não só a infraestrutura necessária se torna insustentável, como os recursos despendidos estão a ser utilizados ineficientemente. Assim, torna-se necessário reduzir a duplicação de tarefas, o que, por sua vez, implica compreender quais as tarefas e processos que são comuns aos vários canais de distribuição.

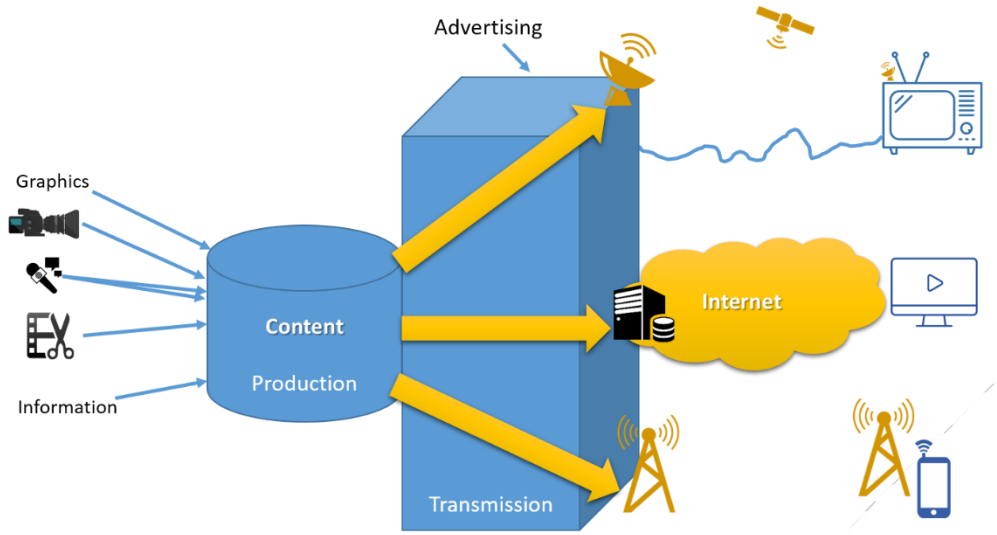

Figura 13: Produção multiplataforma através de um *workflow* convergente [1].

Um *workflow* convergente (Figura 13) assenta no princípio de "*create once, use everywhere*", isto é, o conteúdo é criado uma única vez e utilizado nos diferentes canais de distribuição. Para tal, a infraestrutura deve ser facilmente configurada para se adaptar às diferentes necessidades de cada canal.

A eficiência deste processo de produção multiplataforma provém de esforços de engenharia de *software* em conseguir resolver os problemas de forma inteligente ("*work smarter, not harder*", em inglês). Em verdade, é disso que se trata a produção convergente: construir uma infraestrutura para produzir conteúdos para os vários canais de distribuição sem grande esforço adicional.

Ao eliminar a repetição tarefas redundantes e ao automatizar processos repetitivos, o aumento da eficiência na produção permite reduzir o "*time to air*", ao mesmo tempo que maximiza a utilização de recursos [1].

O *workflow* convergente para produção multiplataforma consiste em três processos fundamentais, tal como a Figura 14 mostra. No processo de produção obtém-se o áudio, vídeo, grafismos, produz-se o conteúdo final a partir desse conteúdo e criam-se os *templates* para os diferentes canais de distribuição. De seguida, todo o conteúdo é gravado numa base de dados, onde é feita a gestão de conteúdos. Na fase de conversão, os conteúdos são formatados de acordo com a plataforma a que se destinam e comprimidos para serem transmitidos.

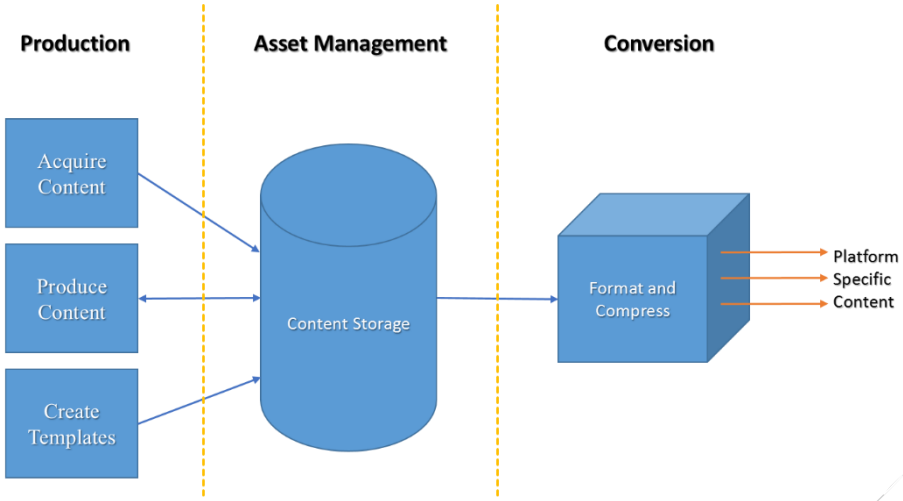

Figura 14: Os três processos de um *workflow* convergente para produção multiplataforma [1].

Apesar das vantagens serem várias, em última instância a produção multiplataforma que um *workflow* convergente proporciona permite que a produção se deixe de focar nas questões tecnológicas e se possa focar no conteúdo e na sua qualidade. Assim, analisando novamente a Figura 9, o avanço tecnológico permite alavancar a dependência cíclica entre as 3 vertentes.

4.3 Transporte de Conteúdos

Na fase de montagem de um programa (a segunda fase no ciclo de vida dos conteúdos, Figura 10), o vídeo, som, grafismo, segmentos editados e animações são combinados num único programa e passados para a distribuição.

A montagem de um programa é um processo complexo: o áudio e o vídeo existem separadamente e não têm nenhuma relação natural de temporização definida entre si. Assim, para além do conteúdo do programa, é necessário transmitir informação acerca de como "reconstruir" o programa antes da visualização (numa televisão, por exemplo) [1]. No transporte de conteúdo, via IP, o MPEG Transport Stream (MPEG-TS) é largamente utilizado [1].

Figura 15: Etapas da conversão de essência para pacotes TS [1].

O MPEG-TS [18] é um *media container* para armazenar e transmitir programas. As *streams* MPEG-TS são compostas por um ou mais programas, que estão descritos no *Program Association*

Table (PAT). No caso de conter apenas um programa, estamos perante uma *Single Program Transport Stream* (SPTS), enquanto se vários programas forem transmitidos na mesma *stream*, diz-se que é uma *Multiple Program Transport Stream* (MPTS) [19].

Um programa é uma associação lógica de *Packetized Elementary Streams* (PES), uma forma de encapsular *Elementary Streams* (ES), resultantes de um *encoder*, por exemplo, em pacotes. Por exemplo, um programa pode ser composto por uma PES de vídeo, outra de áudio e uma terceira contendo as legendas. Tal como o conjunto de todos os programas estão reunidos na PAT, as informações de cada programa existente numa *stream* MPEG-TS encontram-se na *Program Map Table* (PMT).

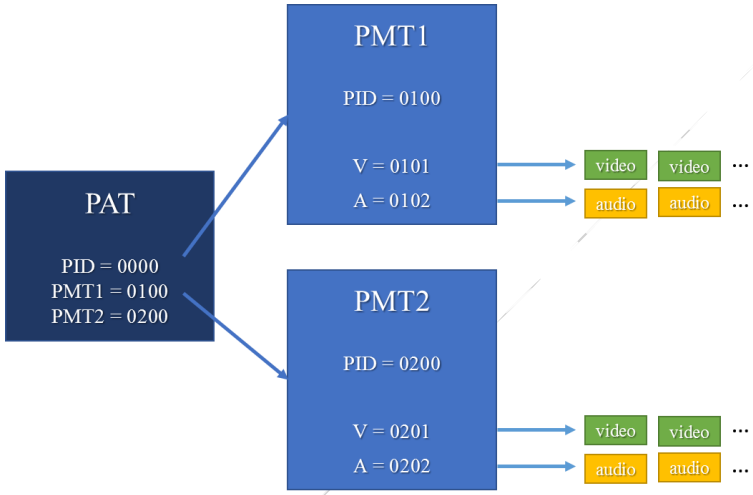

Figura 16: Relação entre a PAT e a PMT [1].

Uma *stream* MPEG-TS é um conjunto de pacotes TS, cada um com um tamanho fixo de 188 bytes (4 para o cabeçalho e 184 para o conteúdo). Cada pacote TS contém, no seu cabeçalho, um *Packet ID* (PID) que permite identificar o conteúdo que transporta. Assim sendo, apesar de um pacote PES poder estar distribuído por vários pacotes TS, um pacote TS com um determinado PID transporta apenas informação de uma única PES.

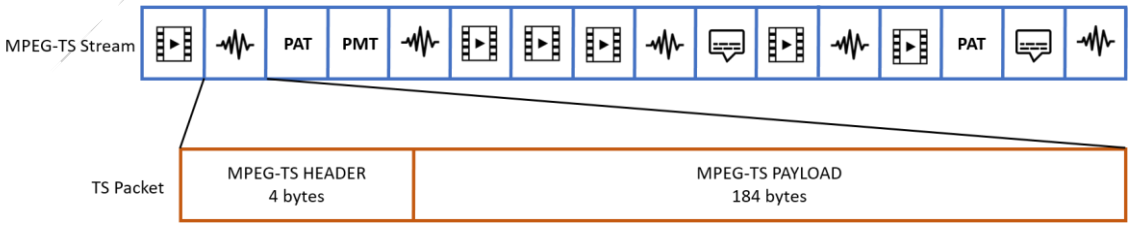

Figura 17: Uma *Stream* MPEG-TS e os seus respetivos pacotes TS.

Como uma *stream* MPEG-TS pode conter várias PES, que, por sua vez, podem pertencer a programas diferentes, o PID existente nos pacotes TS permite, sem a necessidade de muito

processamento, saber quais os pacotes que queremos processar e, da mesma forma, quais os que podemos descartar. Analogamente, uma *stream* MPEG-TS, ao ser multiplexada, combina as várias ES numa só sequência de pacotes TS. Isto significa que não existe uma ordem aparente resultante desse processo (Figura 18). Assim, cada pacote PES contém informação que permite a sincronização, nomeadamente o *Presentation Timestamp* (PTS) e o *Decode Timestamp* (DTS), valores relativos ao *Program Clock Reference* (PCR) e que também é transmitido nos pacotes TS.

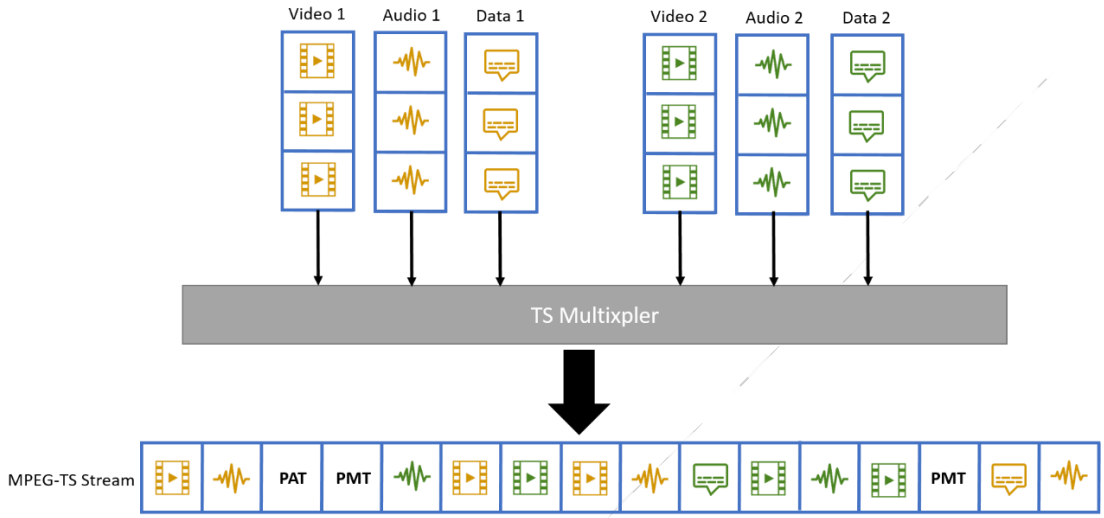

Figura 18: Multiplexer de uma *stream* MPEG-TS.

O transporte de uma *stream* MPEG-TS por IP deve ser encapsulado no conteúdo de um pacote RTP, tal como o *Request for Comments* (*RFC)* 2250 [20] especifica. Cada pacote IP é composto pelo cabeçalho IP, o cabeçalho UDP, o cabeçalho de RTP e um número inteiro de pacotes TS no seu conteúdo (cada um com o tamanho fixo de 188 bytes), tal como mostra a Figura 19.

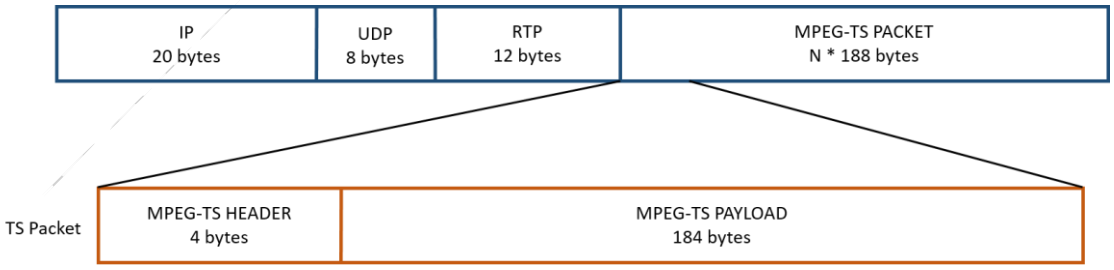

Figura 19: *Stream* MPEG-TS encapsulada em RTP.

4.4 Serial Digital Interface

Serial Digital Interface (SDI) é uma família de interfaces digitais de vídeo, aúdio e metadados normalizada pela *Society of Motion Picture and Television Engineers* (SMPTE) e que se tornou, no início dos anos 90, na interface *standard* para a produção televisiva [7].

Apesar do *standard* original remeter a 1989, vários *standards* adicionais foram desenvolvidos, de forma a introduzir novas resoluções, *frame rates* e vídeo estereoscópico (3D).

Este *standard* está apenas disponível em equipamentos profissionais e a sua natureza síncrona e digital permite transmitir áudio e vídeo sem compressão e sem perdas ou interferências. O SDI é ainda a referência tecnológica no transporte de conteúdos em cenários em direto.

Um sistema de produção em direto baseado em SDI tem uma topologia em estrela e é composto por várias redes, dado que os sinais de vídeo e áudio são transportados em separado. Para além disso, são necessárias redes adicionais que transportam a sincronização do áudio e vídeo, assim como sistemas de gestão, monitorização e controlo de sinais [21].

Por o SDI ser uma interface para a indústria da televisão, cada vez que surge um novo formato, existe a necessidade de adaptar o SDI de acordo com a nova realidade. Isto significa, na prática, que, com a introdução de cada formato, existe a necessidade de reconstruir ou adaptar as infraestruturas de SDI já existentes [22].

Capítulo 5

Produção Audiovisual sobre IP

A virtualização da produção móvel, com recurso à transição do SDI para IP, é objeto de discussão e estudo na indústria de *broadcasting*. Mas o foco da virtualização da produção móvel não passa só por substituir o SDI e as ligações satélites. A ideia "sedutora" do "*live* IP" prende-se com enviar todos os sinais vindos do local do evento (áudio, vídeo, sinais de comunicação, controlo, entre outros) por um único "local" [3].

5.1 Casos de Estudo

A transformação de um estúdio baseado em SDI para tecnologias baseadas na *Ethernet*/IP e na *cloud* é o foco dos casos de estudo que se analisam de seguida.

5.1.1 VISION Cloud

Apesar da VISION Cloud [6] ser uma solução de armazenamento na *cloud*, merece atenção pois criou uma nova camada de abstração no armazenamento e mobilidade de conteúdos multimédia num estúdio televisivo. Ao criar um acesso ao conteúdo multimédia focado no seu conteúdo, a VISION Cloud utilizou a *cloud* e o processamento que lhe está inerente para processar metadados e, assim, relacionar os vários conteúdos multimédia, criando uma semântica entre eles.

Utilizando uma API REST, a VISION Cloud permitia serviços de *Create, Read, Update, Delete* (CRUD) sobre os objetos, assim como definir relações entre objetos, permitindo pesquisas mais complexas que as típicas pesquisas pelo título do objeto.

A VISION Cloud, para além de tirar proveito da arquitetura e escalabilidade da *cloud*, permite, ainda, o acesso ao mesmo conteúdo em simultâneo por diferentes utilizadores e em diferentes codificações.

5.1.2 IP-Studio-Link

O *IP-Studio-Link* [5] é uma placa eletrónica reconfigurável, equipada com várias interfaces de vídeo, aúdio e dados comuns à produção televisiva. Apesar dos autores reconhecerem que a gestão e supervisão de um estúdio televisivo poderá ser realizada totalmente através de *software* e que a *cloud* pode permitir que todos os sinais de áudio e vídeo possam estar disponíveis em qualquer parte do estúdio, o trabalho por eles desenvolvido foca-se apenas na utilização da *Ethernet*/IP para o transporte das *streams*. Outras operações, como o *switching* de vídeo, são realizadas por um *mixer* externo (físico).

Resumindo, o *IP-Studio-Link* apenas converte os sinais de vídeo, áudio e sincronização em pacotes IP e vice-versa, utilizando um *switch* de 1Gb/s em conjunto com o *Precision Time Protocol* (PTP).

5.1.3 IP-Studio

O IP-Studio [23] é um protótipo que permite criar interfaces entre os vários equipamentos presentes num estúdio de produção televisivo, materializando o modelo de dados proposto pela JT-NM [24] e que o *Networked Media Open Specification* (NMOS) especifica.

A ideia geral do IP-Studio é que tudo na cadeia de produção televisiva é importante e, portanto, deve ser registada para poder ser utilizada mais tarde, seja áudio, vídeo, metadados ou outras informações.

O IP-Studio foi testado com sucesso, pela estação televisiva BBC, nos *Glasgow Commonwealth Games 2014* [25], onde foi produzido um cenário multi-câmara *"IP-end-to-end*". A produção do evento foi distribuída pelos vários locais onde o evento se realizou, em Glasgow. O vídeo era entregue desde Glasgow para Londres, onde era acrescentado os comentários e o *mixing* de áudio. Para além disso, também era produzida uma versão DASH que era disponibilizada no *website* da BBC.

Para a produção deste evento *live*, de forma distribuída, dentro do IP-Studio foi utilizado o protocolo RTP para transmitir dados multimédia entre os vários nós, enquanto o MPEG-DASH foi utilizado para obter dados multimédia pré-gravados, devido a ser um serviço em que a latência não é crítica e se privilegia a confiabilidade dos dados. A sincronização dos vários nós foi obtida utilizando o *Precision Time Protocol*. Todos os nós presentes na rede eram facilmente monitorizados utilizando uma *dashboard web* que permitia obter estatísticas como a latência,

jitter, erros, pacotes perdidos, entre outros dados de interesse, de forma a manter a emissão estável.

O IP-Studio conseguiu minimizar a quantidade de equipamento que necessitou de ser instalado no local do evento, dada a natureza distribuída da produção. Consequentemente, a necessidade de recursos humanos foi, obviamente, menor.

5.1.4 Stagebox

O Stagebox [2] é uma solução desenvolvida para ser integrada em *workflows* de produção em direto, assim como em tarefas de pós-produção, como edição e armazenamento de vídeo. O Stagebox utiliza uma placa física onde é feito o *ingest* de vídeo, aliviando o processamento de vídeo não comprimido para *hardware*. O sincronismo entre *streams* é obtido utilizando o PTP.

O Stagebox foi testado em três eventos em direto, reduzindo, com sucesso, o número de formatos que foram precisos suportar e proporcionando uma redução de custos significativa ao reduzir a necessidade de destacar equipamento e recursos humanos para o local onde decorreu o evento.

5.2 SMPTE 2022-6

O SMPTE ST 2022-6 [26] é um *standard* publicado pela SMPTE, pertencente à família de *standards* SMPTE ST 2022 (ainda em desenvolvimento) que permite a utilização da tecnologia IP na indústria de *broadcasting*. O ST 2022-6 define o transporte de SDI sobre IP utilizando o protocolo RTP, pelo que é apelidado de "*SDI over IP*".

Ao encapsular o *payload* do SDI em pacotes IP, o ST 2022-6 permite empacotar o sinal SDI em vários pacotes de 1376 *bytes* cada, transmitir através de uma rede *Ethernet*, receber os pacotes e voltar a reconstruir o sinal SDI. O que, apesar de permitir interoperabilidade com outros equipamentos que utilizem SDI, significa que não existe separação entre as várias *streams* existentes no SDI [27]. Imagine-se, por exemplo, que se pretende modificar o áudio que faz parte de uma *stream* SDI que é transportada com o vídeo correspondente. Nesse caso, será necessário ter que lidar com o vídeo e todo o *overhead* que este trará para o sistema, quando apenas se pretende modificar o áudio.

Ou seja, ao não existir separação entre os vários conteúdos presentes no SDI, não existe flexibilidade no transporte de apenas uma parte do conteúdo.

5.3 RTP *Payload Format for Uncompressed Video*

O RFC 4175 [28] é uma norma que especifica o perfil para o transporte de vídeo não comprimido sobre RTP. O RFC 4175 acrescenta uma extensão de 16 bits denominada de "*Extended Sequence Number*" que estende o já existente número de sequência. A combinação destes dois valores fazem com que seja possível um número com alcance de 32 bits, permitindo enviar e receber pacotes a uma taxa bastante elevada.

Para pacotes de, pelo menos, 1000 bytes, a uma taxa de débito de cerca de 1 Gbps, o número de sequência dos pacotes RTP (sem a extensão) esgotar-se-ia em 0.5 segundos, o que pode ser problemático para ambientes em que a latência é maior que meio segundo.

5.4 Arquitetura de Referência JT-NM

Esta arquitetura de referência, idealizada por um consórcio formado entre a *European Broadcasting Union*, a SMPTE e a *Video Services Forum*, denominado de *Joint Task Force on Networked Media* (JT-NM) *Reference Architecture v1.0* [24], reúne boas-práticas, recomendações e *frameworks* para que possa existir interoperabilidade entre equipamentos de diferentes fabricantes na transição do SDI para o IP.

O JT-NM define um modelo conceptual (incompleto na Figura 20, focando apenas os aspetos necessários e relevantes tendo em conta o âmbito desta dissertação) que permite mapear os *workflows* de forma a existir a desejada interoperabilidade.

No "coração" das operações está a rede, tipicamente *Ethernet*. Os *Nodes* estão ligados à rede e são eles que criam a infraestrutura existente, seja com capacidade de processamento, armazenamento ou interfaces para outros nós.

Os *Devices*, que podem ser virtuais ou físicos, aprovisionados em nós existentes na rede, são agregações funcionais de *Capabilities*, aptidões, para realizar determinadas *Tasks*. Um *Device* pode ser uma câmara de filmar, um adaptador de SDI para IP ou um *mixer* de áudio, por exemplo.

Um *Source* é um conceito abstrato que representa a origem de dados provenientes de um *Device*. Um *Device* pode fornecer vários *Sources*, por exemplo, uma câmara de vídeo fornece um *Source* de áudio e outro de vídeo. Os *Sources* materializam-se em *Flows*, que são sequência de *Grains* de um mesmo *Source*. Um *Source* pode fornecer diferentes *Flows*, por exemplo um *Source* de vídeo pode fornecer um *Flow uncompressed* (sem compressão) e um *Flow* codificado em H.264.

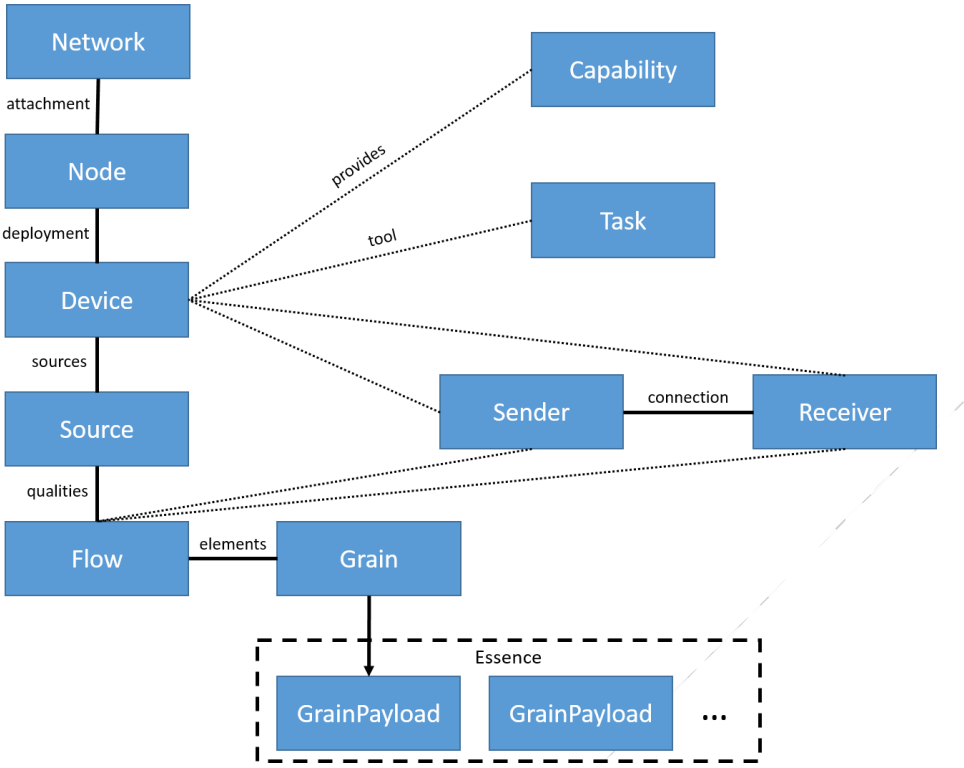

Figura 20: Modelo conceptual simplificado da arquitetura de referência do JT-NM [24].

Um *Grain*, por sua vez, é o elemento atómico que representa um elemento de uma essência, como um *frame* de vídeo, ou *samples* de áudio correspondentes a um *frame*, por exemplo. Cada *Grain* é constituído pelo conteúdo que transporta (vídeo, áudio ou metadados) e por um *timestamp* no qual ele foi criado. O *timestamp* deve ser derivado do relógio presente no *Node*, que está sincronizado com os restantes *Nodes* da rede, com uma precisão ao nível dos nanosegundos, utilizando o *Precision Time Protocol* [29].

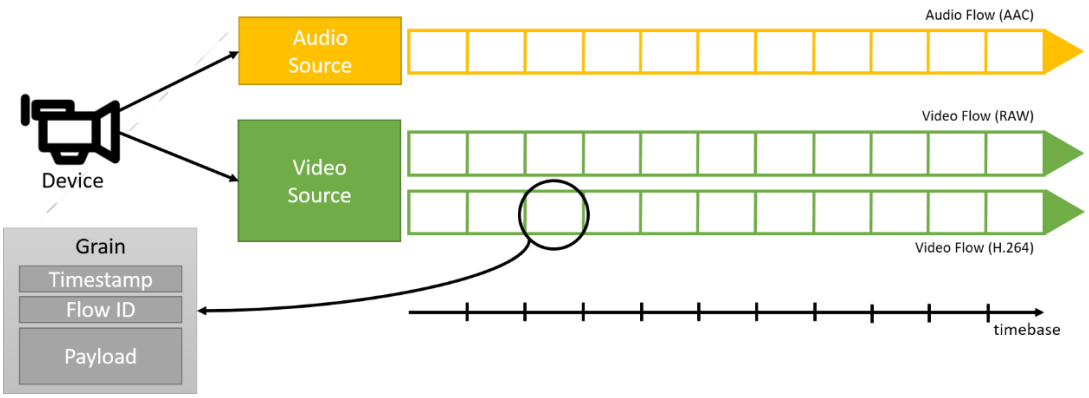

Figura 21: Relação entre *Sources*, *Flows* e *Grains* [23].

A flexibilidade deste modelo conceptual permite situações como a da Figura 21. Com uma base temporal comum, é possível mapear *Grains* de *Flows* do mesmo *Source* de vídeo. Da mesma forma, a sincronização entre o áudio e o vídeo é independente do *Flow* de vídeo que é utilizado.

Para trocarem *Flows* entre si, os *Devices* devem registar-se como disponibilizadores de *output* (*Sender*) e como interessados em determinado *input* (*Receiver*). Um *Sender* pode ser, por exemplo, uma câmara e um *Receiver* um monitor.

Para que esta informação esteja devidamente articulada e possa ser utilizada para a definição de *workflows*, são definidos 3 blocos fundamentais sobre quais este modelo de dados assenta: *Timing*, *Identity* e *Discovery & Registration*.

Cada *Node* na rede deve, após o seu aprovisionamento, registar-se a si e aos *Devices*, *Sources*, *Flows*, *Senders* e *Receivers* que disponibiliza, de forma que outros nós os possam descobrir e obter a informação apropriada sobre cada um (*Discovery & Registration*). Da mesma forma, cada elemento presente na infraestrutura deve ser fácil e unicamente identificável, de forma a poder ser referenciado e utilizado. As relações entre recursos devem ser feitas expressamente, com recurso aos identificadores (*Identity*). Por fim, a utilização de *timestamps*, ao nível dos *Grains*, permite manter a consistência das operações realizadas e, consequentemente, a assegurar que os *Flows* estão corretamente alinhados (*Timing*).

5.5 NMOS

O JT-NM descreve um modelo conceptual de interoperabilidade, que apenas reúne uma arquitetura de referência e uma coleção de boas práticas. Como tal, o Networked Media Open Specifications (NMOS), criado pela Advanced Media Workflow Association, pretende ser uma família de especificações que pretende criar *frameworks* que permitam a interoperabilidade pretendida pelo JT-NM.

O NMOS baseia-se no modelo conceptual de dados proposto pelo JT-NM, de forma a adicionar identidade e relações entre o conteúdo e os equipamentos. Independentemente da tarefa específica que cada *Node* desempenha, a visão lógica de um *Node* segundo o NMOS (Figura 22) permite criar um nível de abstração suficiente para garantir a modularidade e expansibilidade esperadas para se adaptar a diferentes necessidades.

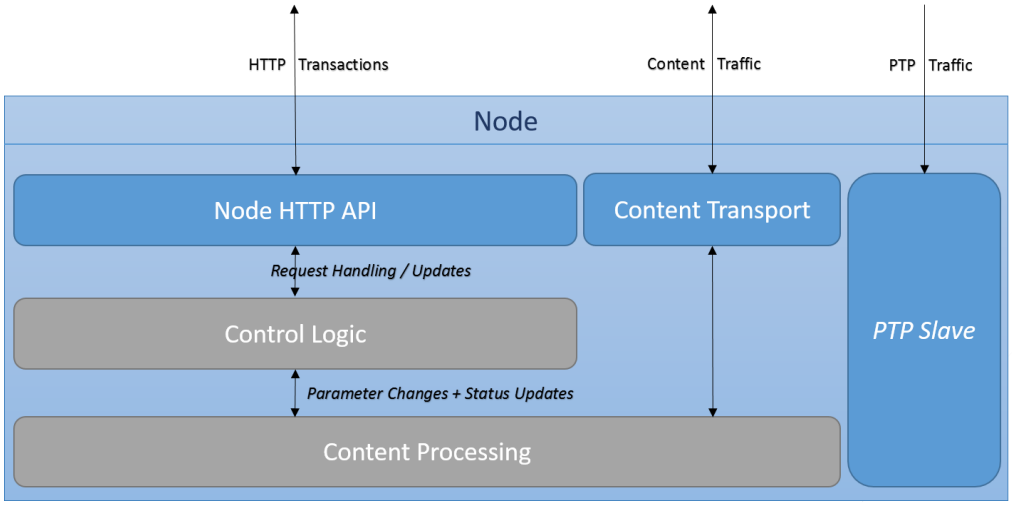

Figura 22: Visão lógica do *Node* proposto pelo NMOS [30].

O NMOS não especifica como cada *Node* deve funcionar internamente (a cinza na Figura 22), apenas especificando quais as interfaces que devem ser expostas. Cada *Node* deve expor transações HTTP realizadas por uma API REST, de forma a permitir controlar o modelo de dados presente em cada *Node*. Isto permite à lógica de controlo do *Node* receber ordens e tomar as ações necessárias. Estas transações estão descritas na especificação "AMWA NMOS Discovery and Registration Specification (IS-04)"[7] proposta pelo NMOS.

5.5.1 RTP NMOS

Para transportar conteúdo entre *Senders* e *Receivers*, o NMOS especifica o mapeamento de identificadores de *Flow* e *Source*, assim como *timestamps*, nos cabeçalhos do RTP. Ao não especificar nenhum formato para o conteúdo do RTP, o NMOS pretende ser agnóstico quanto ao *codec* utilizado. Isto significa que o NMOS incentiva a utilização da transmissão multimédia via RTP sem modificação dos RFCs existentes.

Segundo o NMOS, o primeiro e o último pacote de cada *Grain* devem conter as extensões ao cabeçalho RTP. As extensões são essenciais à coesão do modelo proposto e permitem: a) prover os pacotes RTP com informação que lhes dê significado ao nível aplicacional; b) sincronizar *Flows*.

O *Sync Timestamp* dá uma referência absoluta na qual uma essência de um *Grain* foi captura ou reproduzida. Dois *Grains* de áudio e vídeo coincidentes devem partilhar o mesmo *Sync Timestamp*, que se deve manter inalterado independentemente do processamento que o *Grain* sofre.

[7] https://github.com/AMWA-TV/nmos-discovery-registration

O *Origin Timestamp* fornece uma referência absoluta em que a essência de um *Grain* foi criada (isto é, capturada). Esta relação está intimamente relacionada com a geração de *Flows* provenientes de um mesmo *Source*. No caso de uma captura de direto, por exemplo, o *Origin Timestamp* coincide com o *Sync Timestamp*.

O *Flow Identifier* é uma referência de identidade, pois referencia a que *Flow* o *Grain* pertence. Da mesma forma, o *Source Identifier* identifica o *Source* do qual o *Flow* atual deriva, permitindo compreender qual a sua origem.

5.5.2 Sincronização

Com o mapeamento que o NMOS propõe, a tarefa de sincronizar vários *Flows* torna-se trivial. Como cada *Grain* tem uma referência de tempo comum, é possível sincronizar, quando necessário, os vários *Flows* intervenientes.

Analisemos, por exemplo, a Figura 23. À medida que, quer a essência de vídeo, quer a essência de áudio vão sendo adquiridas, o dispositivo de captura gera *timestamps* que são estarão associados aos *Grains* resultantes. À medida que os *Grains* vão sendo transportados na rede, o dispositivo que os recebe, vai-os colocando num *buffer* ordenado pelo *timestamp*. Daí, o relógio utilizado para a apresentação (*Presentation Clock*) dos *Flows* de vídeo e áudio deve ter um deslocamento do relógio original (*Master Clock*) de forma a compensar os atrasos introduzidos pelo processamento (*decoding* de vídeo, por exemplo) e pela transmissão na rede.

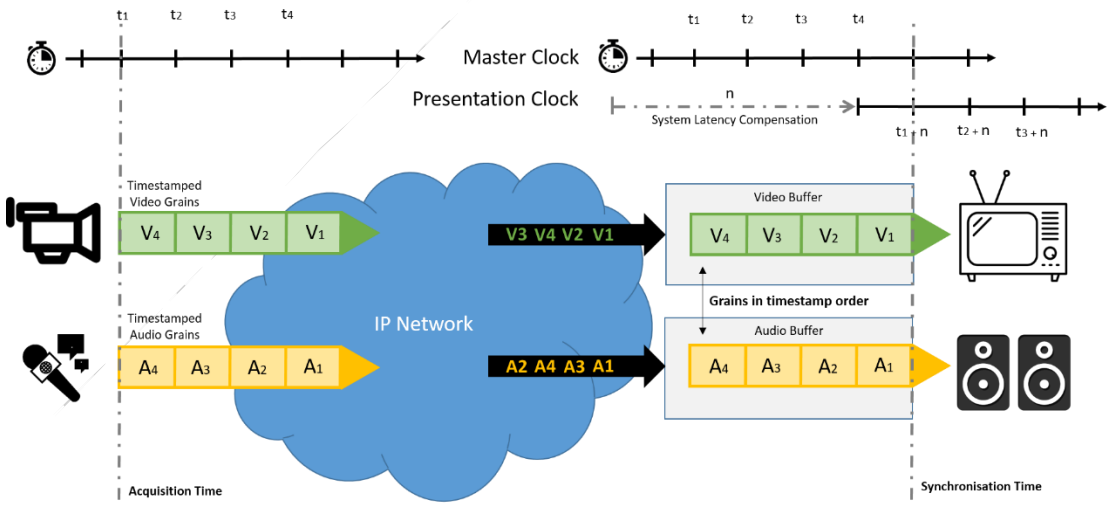

Figura 23: Exemplo de sincronização.

A identidade de cada *Grain* é assegurada pelo par *Flow ID/Synchronization Timestamp*. Isto significa que, por exemplo, na criação de um novo *Flow*, como num processo de *encoding*, não é perdida a relação temporal de *Flows* pertencentes à mesma *Source*.

Na prática, a identidade de cada *Grain* permite sincronizar facilmente *Flows* provenientes de *Sources live*, não só entre si, mas também com *Flows* que já se encontram armazenados e vão ser diferidos.

5.6 Conclusão

Tal como se pôde verificar, apesar de existir um interesse alargado em evoluir a produção televisiva para tecnologias suportadas pelo IP e pela *cloud*, não existe uma solução que permita, simultaneamente, aproveitar as vantagens que ambos oferecerem.

A utilização da *cloud* tem-se focado bastante no armazenamento de conteúdo multimédia, apesar de [14] indicar o *cloud-based transcoding* como umas das potenciais aplicações da *cloud* no domínio da produção televisiva. Apesar disso, nenhum dos casos de estudo analisados utiliza a *cloud* para esses efeitos.

Por outro lado, protótipos como o IP-Studio-Link e o Stagebox mostraram-se funcionais, mas baseados em placas físicas para ações que podiam ser virtualizadas e, portanto, realizadas na *cloud*, num ambiente resiliente, controlável, adaptativo e elástico. Da mesma forma, esses protótipos são opções muito específicas para determinados *worklflows* e ambientes, não proporcionando a modularidade e expansibilidade que seria de esperar para um ambiente tão dinâmico e abrangente como a produção televisiva.

É importante referir que o IP-Studio é uma proposta interessante sob o ponto de vista da abstração dos componentes que compõe a produção, testando uma *framework* que permite a interoperabilidade entre esses mesmos componentes e cuja arquitetura pode ser a base para uma aplicação distribuída servida como SaaS.

O JT-NM e a concretização que o NMOS cria podem ser pontos de partida para o desenvolvimento do tão cobiçado "*full IP Studio*", pelo que será bastante interessante tentar compreender o rumo que ambos irão tomar e se a indústria convergirá nas suas especificações.

Capítulo 6

Virtualização da Produção Móvel

A cobertura de um evento remoto pode ser algo simples como uma simples câmara ligada a uma *OB van*, as várias câmaras distribuídas pelo recinto do evento, ou, até mesmo, um cenário complexo com várias *OB vans*. A possível complexidade que a produção de eventos em direto possibilita, principalmente grandes eventos, resultam não só num desafio tecnológico, como também num desafio logístico [2].

Apesar da tecnologia IP já estar bastante presente em pós-produção *file-based*, em cenários de produção *live* o SDI predomina por completo as soluções disponíveis [1]. Com a tendência na indústria de *broadcasting* em utilizar soluções de *software* baseadas em IP [13], a capacidade de realizar a produção de eventos em direto utilizando um ambiente *cloud* controlado por *software* pode significar reduzir custos e, simultaneamente, flexibilizar a produção.

Interessa, portanto, analisar a metodologia atual e compreender de que forma se pode tirar partido da tecnologia IP em produção *live*.

6.1 Contexto e Método Atual

Na cobertura de eventos televisivos em direto, como por exemplo um jogo de futebol, existem vários intervenientes, nem sempre geograficamente próximos, que trabalham em conjunto: as várias câmaras espalhadas pelo recinto, as entrevistas rápidas junto ao recinto, a produção responsável por adicionar gráficos e animações que tornam a experiência de visualização mais rica e intuitiva e os comentadores e analistas, que acompanham o evento em direito, e que podem estar tanto no recinto como no estúdio.

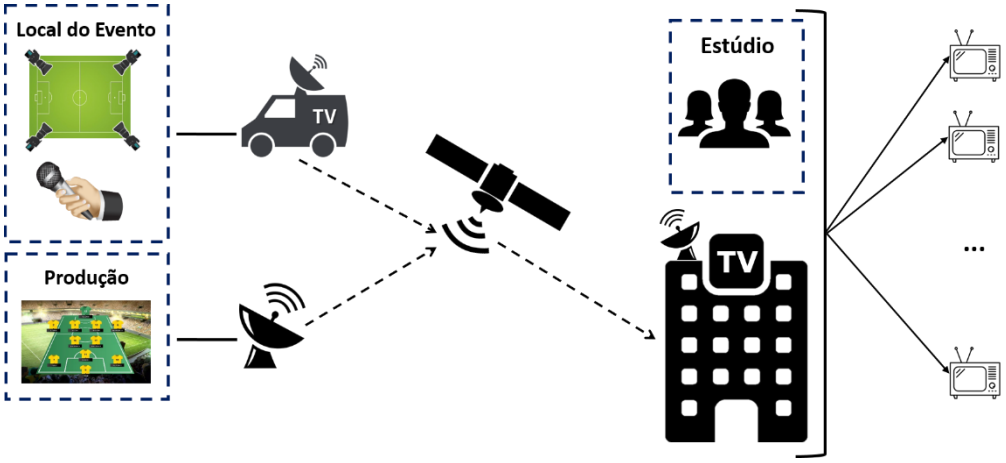

Figura 24: Esquema simplificado da produção móvel utilizando OB van.

O método de cobertura atual mais utilizado baseia-se na utilização de um estúdio de produção móvel, a OB *van*, que se encontra no exterior do recinto e à qual os vários dispositivos (câmaras e microfones, por exemplo) estão ligados. Esse sinal é, de seguida, transmitido da OB *van* para o estúdio por satélite. Este método requer a necessidade elevada de recursos humanos no local do evento, assim como a utilização de equipamentos (OB *van* e a ligação satélite, por exemplo) que elevam bastante os custos de produção.

Da mesma forma, a flexibilidade que a produção tem é muito reduzida. Isto é, os recursos logísticos são requisitados e planeados para um cenário específico e a modificação desse plano (a introdução de mais uma câmara ou um novo formato de entrega de conteúdo, por exemplo) pode ter custos operacionais bastante elevados [3].

6.2 Requisitos

Na definição de uma solução para virtualizar a produção de eventos em direto recorrendo a estúdios de produção móvel, é necessário garantir os seguintes requisitos:

- O ambiente *cloud* deve garantir QoS na largura de banda da rede que: a) faz a contribuição para a *cloud* (*streams input* da *cloud*); b) faz a ligação entre os vários componentes da aplicação distribuída na *cloud*, onde o vídeo é transmitido sem compressão (*uncompressed*).

- Não deve ser utilizado *Forward Error Correction* (FEC). Como tal, quer o *error rate*, quer o *loss rate* devem ser 0. Este requisito implica a ausência de ruído ou congestionamento da rede, o que pode ser bastante restritivo. De qualquer forma, para manter a QoE, não é possível relaxar este requisito.

- A rede deve permitir a comunicação por *multicast*, com o protocolo *Inter*net *Group Management Protocol* (IGMP) ativo.

- Para manter a QoE, a aplicação deve ser *real-time*. Isto significa que os componentes têm que ser capazes de escalar verticalmente, de forma a não acrescentar atrasos. Isto implica, por exemplo, que o *transcoding* na *cloud* seja, pelo menos, *real-time*, isto é, o atraso no processamento não pode ultrapassar 1 *frame* de diferença.

- O atrasado introduzido pela aplicação é constante e deve ser definido no momento do aprovisionamento da mesma. É da responsabilidade do SLA concordado entre o utilizador final e o provedor do serviço manter as condições de rede ideais para que o atraso possa ser o menor possível, de forma a manter a transmissão o mais perto possível do *real-time*.

6.3 Solução Proposta

Para a produção de eventos televisivos em direto, de uma forma distribuída, é proposta uma aplicação distribuída na *cloud* suportada pela transmissão de vídeo através de IP e que permita ao realizador, através de uma *Web App*, realizar ações como trocar de câmara, ou seja, escolher de entre um número de *streams* de entrada, qual o *output* a obter (Figura 25).

Ao utilizar as potencialidades de escalabilidade da *cloud* é possível lidar com, por exemplo, o *transcoding* de vídeo na *cloud* ou para se adaptar o número de *streams* que se ligam em simultâneo à aplicação. A realização de operações *software-based* como o *transcoding* ou o *switching* na *cloud*, antes realizadas por *hardware*, permite virtualizar a produção não só para um ambiente distribuído, como também para um ambiente flexível e adaptável.

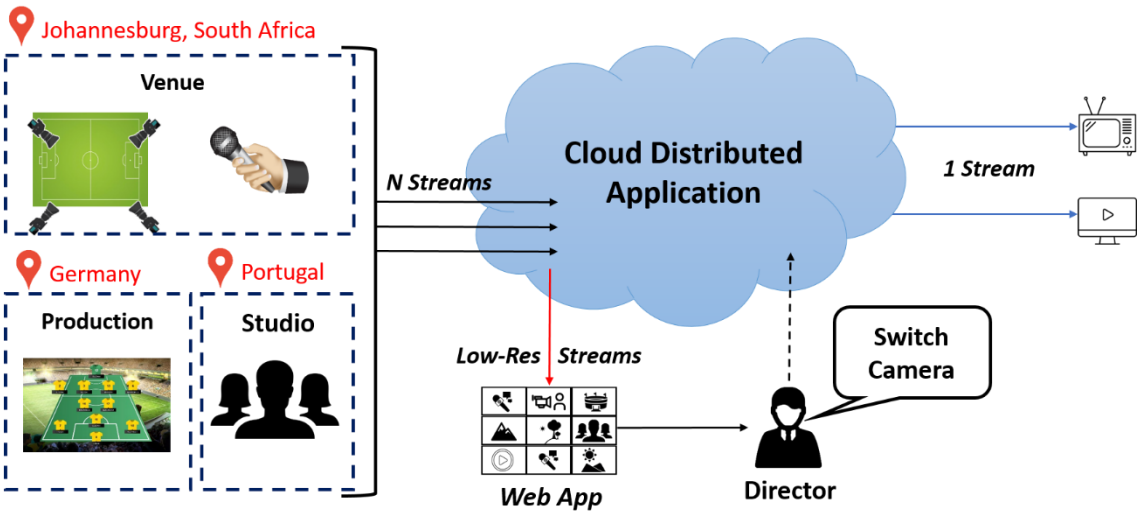

Figura 25: Vista *black-box* da aplicação.

A aplicação distribuída deve ser capaz de receber *streams* de áudio e vídeo e servir versões *proxy* (de baixa resolução) para a *Web App*, ao mesmo tempo que lida com a sincronização das *streams* e a coerência que está inerente ao *output*. Ao ser servida ao utilizador final como SaaS, permite atualizações constantes de forma transparente ao utilizador, ao mesmo tempo que tira partido da flexibilidade da *cloud*.

6.3.1 Arquitetura

A aplicação proposta é composta originalmente por 5 blocos (Figura 26), cada um com uma função distinta e bem específica. Cada bloco corresponde a um nó (*Node*) na rede e, apesar de terem funções distintas, o nível de abstração que oferecem à rede e a comunicação entre nós é idêntico entre todos eles e baseado nos *Nodes* propostos pelo NMOS. Para que o QoS para uma aplicação *time-critcal,* como a que se apresenta, seja assegurado, poderá ser necessário que os 5 nós estejam na mesma rede privada.

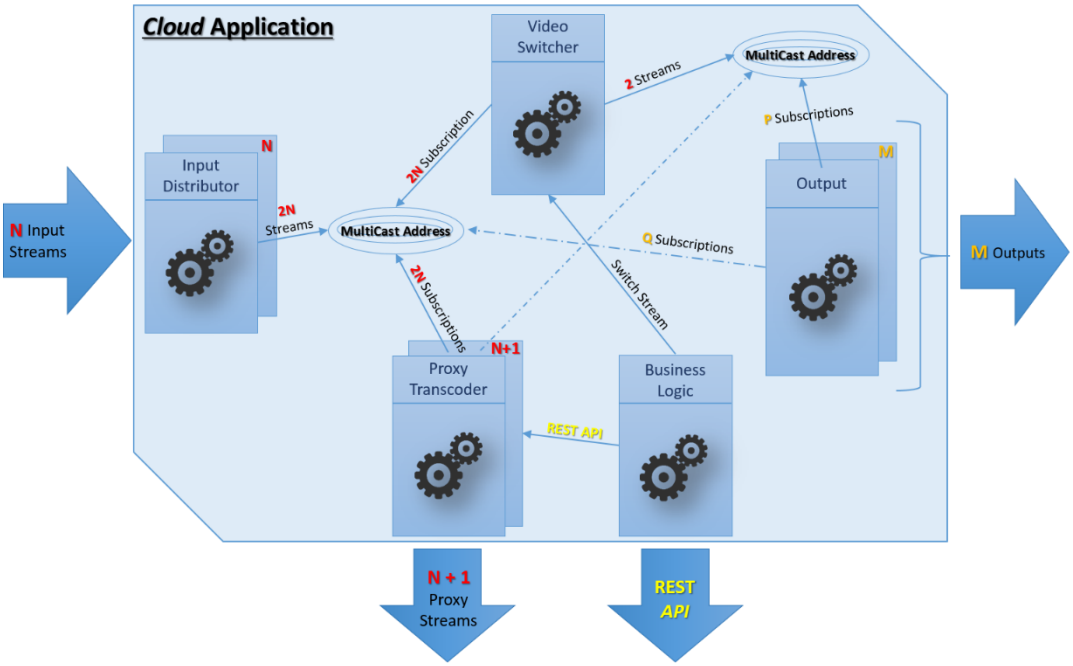

Figura 26: Diferentes nós da aplicação distribuída.

De forma a manter a qualidade que o SDI já proporciona num ambiente de produção atual, os *Flows* dentro da aplicação *cloud* devem ser transportados sem compressão (*uncompressed*).

Cada nó *Input Distributor* é responsável por receber um *stream* de entrada, descomprimi-la e fornecer, por *multicast,* os F*lows* resultantes. Neste caso, os principais nós interessados são o *Video Switcher* e o *Proxy Transcoder*.

Cada *Proxy Transcoder* é responsável fazer *transcoding* ao par de *Flows* que subscreveu (o par vídeo e áudio correspondente), gerar uma versão *proxy* e disponibilizá-la para o exterior, por exemplo para uma *Web App*.

O *Business Logic* será o ponto de conexão com o exterior. Deverá disponibilizar chamadas REST para obter informações da aplicação e comandar as operações a realizar. O *Business Logic* deve servir de interface para o modelo de negócio, prescrevendo como os diferentes módulos devem interagir, ou seja, implementa a *Query API* proposta pelo NMOS.

O *Video Switcher* deve subscrever os endereços *multicast* que os *Input Distributors* estão a fornecer, guardar em memória os dados que vai recebendo e servir, por *multicast*, o par de Flows (vídeo e áudio correspondente) que o *Business Logic* determinar. É necessário guardados dados em memória devido ao *delay* introduzido pelas várias transmissões em rede e pelo processo de *transcoding* realizado pelo *Proxy Transcoder*. Apesar disso, [21] sugere outra técnica de *switching* baseado no IGMP, mas que, por si só, não garante *frame accurate switching*.

Cada nó *Output* recebe, por *multicast*, os dois *Flows* (par vídeo/áudio) provenientes do *Video Switcher* e entrega-os para o exterior, numa única *stream*, com uma codificação específica e para uma plataforma específica. Isto significa que podem existir vários *Outputs*, incluindo, por exemplo, um *Output* que entregue uma *stream* com as mesmas características do *Input*, de forma a proporcionar cenários em cascata.

6.3.2 Fluxo dos Conteúdos Audiovisuais

Atentando à Figura 26, para cada stream de *input* é aprovisionado um *Input Distributor*, existindo, portanto, uma relação 1:1. Isto significa que, sendo N o número de *input streams* a entrarem na *cloud,* N *input streams* implicam N *Input Distributors*. Devido à desmultiplexação que o *Input Distributor* realiza, obtém-se 2N *Flows*, dado que cada *input stream* origina 1 *Flow* de vídeo e outro *Flow* do áudio correspondente.

A relação entre um *Input Distributor* e um *Proxy Transcoder* também é de 1:1. Ou seja, para 2N *Flows* provenientes do conjunto dos *Input Distributors*, existem 2N subscrições feitas por N *Proxy Transcoders*, cada um subscrevendo o par de *Flows* de vídeo e áudio correspondentes. Para além disso, existe 1 *Proxy Transcoder* adicional, responsável por servir o resultado do *Video Switcher*, pelo que a proporção final de *Proxy Transcoders* é N+1.

Seja M o número de nós de *Output*, que é independente de N, considere-se que existem P subscrições ao resultado do *switching* realizado pelo *Video Switcher*. Dado que um *Output* pode, por exemplo, ser uma versão só áudio de um programa, não existe a necessidade de subscrever o *Flow* de vídeo correspondente, pois apenas acrescenta *overhead* desnecessário. Assim, o número de subscrições P não é necessariamente M. Da mesma forma, o *Output* pode subscrever Q *Flows*

diretamente do *Input Distributor*, para efeitos de completar uma cadeia de *transcoding*, por exemplo.

Assim sendo, atente-se na Tabela 1, onde estão reunidos, para cada conjunto de nós, a quantidade que são aprovisionados, e a relação entre os *Flows* de entrada e os *Flows* que são originados.

Tabela 1: Escalamento dos nós tendo em conta o fluxo de dados da Figura 26.

Conjunto dos Nós	*Aprovisionamento*	*Flows de Entrada*	*Flows de Saída*
Input Distributors	N	N	2N
Proxy Transcoders	N + 1	2N + 2	N + 1
Video Switcher	1	2N	2
Business Logic	1	-	-
Outputs	M	P + Q	M

Entenda-se que as variáveis independentes são N (número de *streams* de *input* na *cloud*) e M (número de *streams* de *output* da *cloud*) e dizem respeito à lógica de negócio. P está limitado, no máximo, a 2M subscrições. Q, por sua vez, está limitado a, no máximo, 2N subscrições.

6.3.3 Expansibilidade e Modularidade

Cada nó na aplicação é interdependente dos restantes nós. A modularidade da aplicação permite que possam ser desenvolvidos novos tipos de nós, como um *Persistent Storage* (Figura 27), que grave e armazene o resultado das operações realizadas pelo *Video Switcher*, por exemplo.

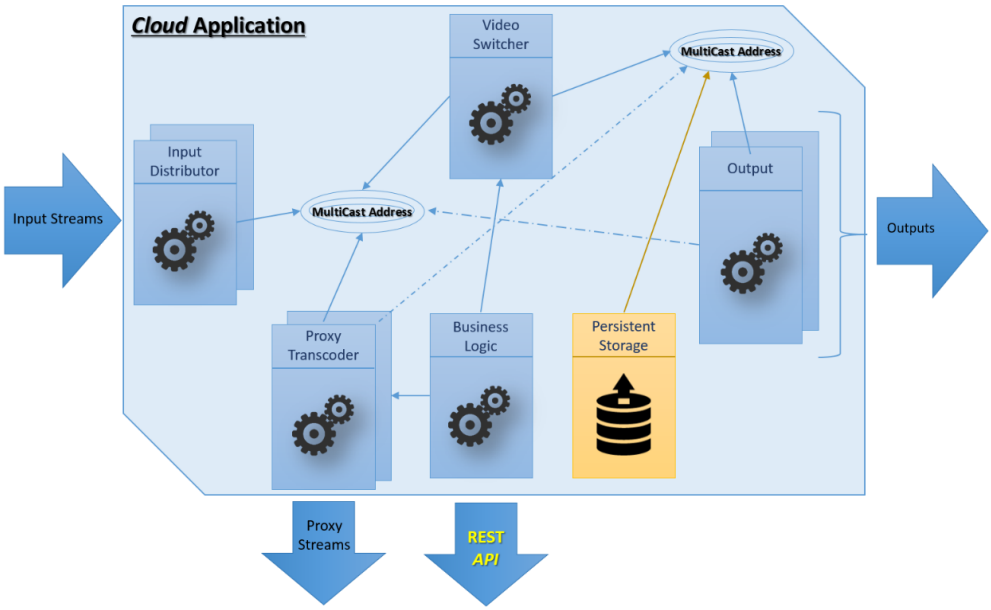

Figura 27: Exemplo de expansibilidade da arquitetura.

As possibilidades de expansibilidade podem permitir a resposta a diferentes necessidades e a criação de *workflows on-demand*. A escolha de entregar Flows por *multicast* permite a entrega *"one-to-many*", proporcionando o aprovisionamento *on-demand* de nós, sem a necessidade de reconfigurar a infraestrutura já existente.

A arquitetura proposta tem em conta a especificação proposta por [21], ao definir interfaces ao nível dos dispositivos e de como eles funcionam e comunicam e interfaces de lógica de negócio, que permitem mapear o que é possível realizar com os serviços disponibilizados.

6.4 Protótipo Desenvolvido

No desenvolvimento do protótipo pretendeu-se que o trabalho realizado permitisse validar a arquitetura definida, para que esta possa, potencialmente, ser utilizada noutras operações do *workflow* da produção televisiva.

O protótipo desenvolvido é capaz de receber um número fixo de *inputs* de áudio e vídeo, descomprimi-los e transportá-los numa rede local através do protocolo RTP, que foi estendido para cumprir a especificação proposta pelo NMOS. Os *inputs* são servidos numa versão de baixa resolução para a *web*, onde, através de uma interface gráfica simples, é possível visualizar as várias *streams*.

O trabalho desenvolvido assentou sobre a definição e conceção da arquitetura e o desenvolvimento de pequenos módulos, com funções e propósitos bem definidos, para que possam ser reutilizados em níveis de abstração superiores. Assim, no âmbito desta dissertação, foram desenvolvidos o *Input Distributor*, o *Proxy Transcoder* e o *Output*, para além de uma *Interface Web* que permite visualizar uma versão de baixa resolução das *streams*.

6.4.1 Restrições e Limitações Técnicas na Prototipagem

Como o foco do trabalho foi a validação da arquitetura definida, pretendendo-se avaliar a sua exequibilidade e viabilidade, foram definidas algumas restrições e limitações técnicas que permitiram agilizar o processo de desenvolvimento:

- São aceites, no máximo, 4 *streams* MPEG-TS de *input*, entregues por RTP, cada uma com 2 PES: uma de vídeo H.264 1080i50 4:2:0 8 bits e outra de áudio AES3.
- O protótipo desenvolvido deve funcionar em ambiente Linux.
- Não existe mecanismo de recuperação de falhas. Se um nó falhar, deve ser reiniciado.

- O *output* será MPEG-TS, entregue por RTP, com apenas o PES de vídeo, nas mesmas condições das *streams* de *input*.

6.4.2 Implementação

Da aplicação proposta na Figura 26, o protótipo desenvolvido focou-se na validação da arquitetura proposta, através do desenvolvimento do *Input Distributor*, do *Proxy Transcoder* e do *Output* (Figura 28). O formato de entrada e de saída das *streams* na aplicação será o mesmo, MPEG-TS transportado por RTP, possibilitando cenários mais complexos, como por exemplo ter várias aplicações encadeadas em cascata. Por sua vez, as versões de baixa resolução serão servidas pelo *Proxy Transcoder* em MPEG-DASH.

A escolha dos *Nodes* a desenvolver permite testar a arquitetura e a sua validade, nomeadamente:

- Testar o *input* na *cloud*, encapsulado em MPEG-TS, uma tecnologia largamente utilizada na indústria de *broadcasting*.
- Testar a transmissão de vídeo *uncompressed* e a respetiva sincronização.
- Testar as operações de *demux* e *decode* e de *encode* e *mux* em ambientes virtualizados.
- Validar as especificações propostas pelo NMOS.
- Estudar métricas como capacidade de processamento, utilização da memória entre outras.

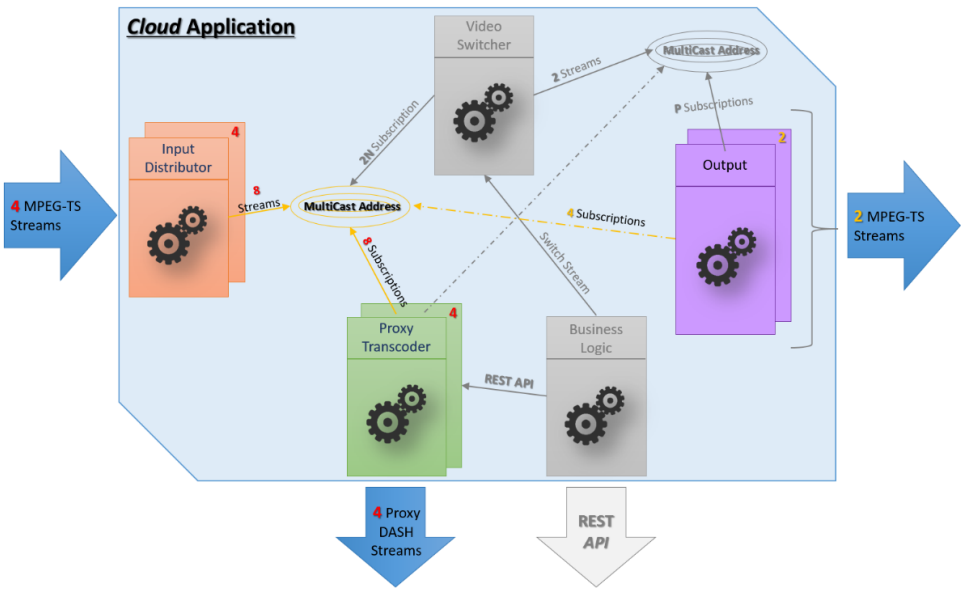

Figura 28: Contextualização do protótipo desenvolvido.

Para desenvolver os *Nodes* de forma modular e isolada, foi escolhida a tecnologia Docker, devido às vantagens que esta possibilita e que foram discutidas no capítulo 2.2. Assim, cada nó é um *container* Docker com base no Ubuntu Server 14.04 LTS [8]. Esta decisão permite aproximar o ambiente de desenvolvimento ao possível ambiente de produção, tendo em conta que vários serviços *cloud* já possibilitam, atualmente, o aprovisionamento direto de *containers* Docker nas suas infraestruturas.

Para cada *stream* MPEG-TS de *input* é aprovisionado um *container* Docker com o *Input Distributor* e outro *container* com o *Proxy Transcoder* correspondente. O fluxo de dados (Figura 29) mostra, alto nível, o ciclo de vida do conteúdo: é recebido por RTP MPEG-TS no *Input Distributor*, enviado por RTP para um endereço *multicast* que o *Proxy Transcoder* pode subscrever para gerar uma versão *proxy*, enquanto o *Output* pode subscrever para voltar a devolver o *Flow* como MPEG-TS transportado por RTP.

Para a transmissão de vídeo sem compressão ,o NMOS incita à utilização do RFC 4175, já analisado no capítulo 5.3. Adicionalmente, estão também presentes os cabeçalhos específicos do NMOS.

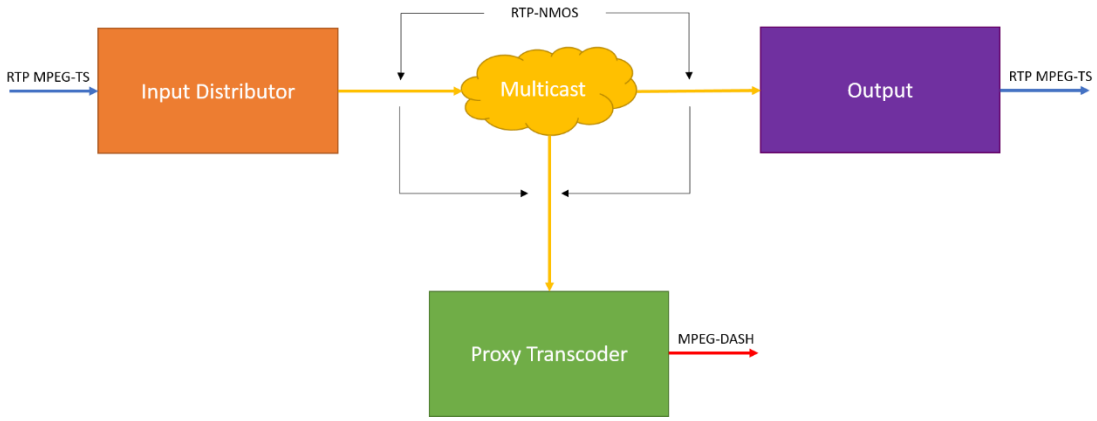

Figura 29: Fluxo de dados de uma *stream* na aplicação.

A escolha de um endereço *multicast* para cada *Flow*, ao contrário do que vária literatura sugere (por exemplo, [2], [3], [21]) permite, mais facilmente, não só possíveis melhoramentos na rede, como também facilita a filtragem dos pacotes.

Ao analisarmos mais detalhadamente a Figura 29 e percebendo as várias transformações que o conteúdo vai sofrendo, conseguimos compreender a necessidade de vários módulos comuns ao longo dos diferentes *Nodes*.

[8] http://releases.ubuntu.com/14.04/

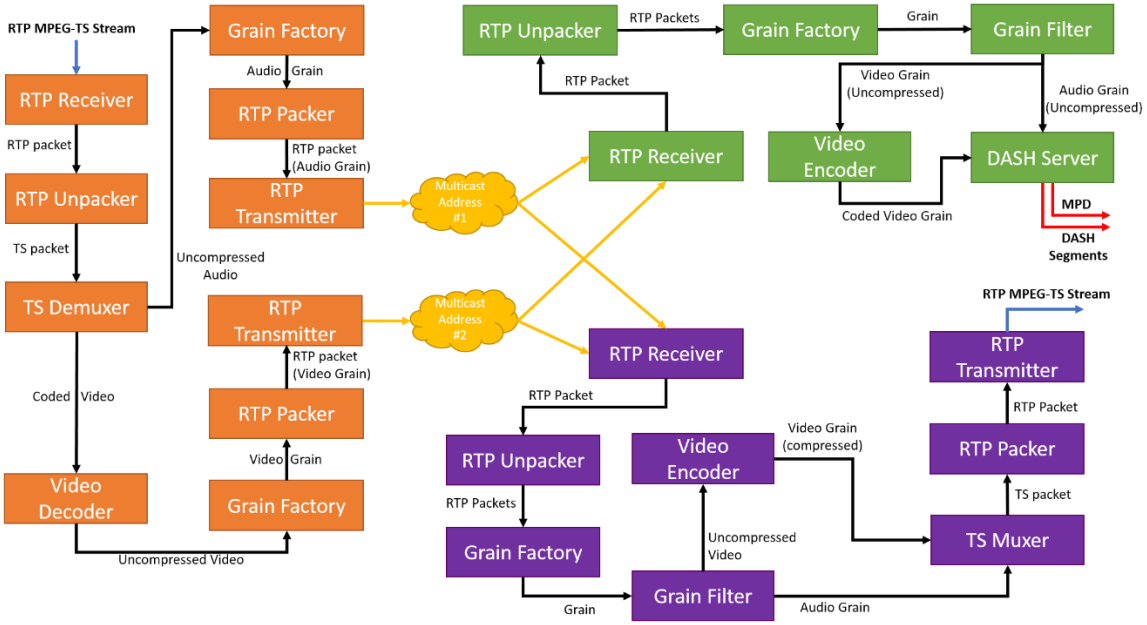

Figura 30: Visão aprofundada do fluxo de dados.

No *Input Distributor* a *stream* (compatível com a norma RFC 2250 [31]) é recebida pelo *RTP Receiver* (utilizando a biblioteca JRTP[9]) e os pacotes RTP são desempacotados (novamente utilizando a biblioteca JRTP) pelo *RTP Unpacker*. Os metadados presentes nos cabeçalhos dos pacotes RTP permitem reordená-los (se necessário), utilizando o número de sequência. De notar que o *RTP Receiver* utiliza um fila de prioridade (ordenada pelo número de sequência do RTP) para ordenar pacotes que cheguem fora de ordem ou repetidos.

O *demuxing* dos pacotes TS presentes no *payload* dos pacotes RTP é feito utilizando a biblioteca FFMPEG[10], obtendo-se, também, metadados relativos à informação que as várias tabelas do MPEG-TS possuem. A partir daí, as essências são tratadas separadamente. Quer o vídeo, quer o áudio, ambos não comprimidos, são transformados em *Grains*, resultantes da associação com os metadados disponíveis.

Para que os *Grains* possam ser transmitidos entre os vários *Nodes*, é necessário voltar a empacotá-los em pacotes RTP (*RTP Packer*). Os pacotes que daí originam possuem os cabeçalhos propostos pelo NMOS.

No *Node Proxy Transcoder*, um módulo *RTP Receiver* é instanciado para receber o par de *Flows* (áudio e vídeo) que subscreveu. De seguida, acontece o processo inverso de desempacotar um pacote RTP e transformá-lo num *Grain*. O vídeo, por sua vez, volta a ser comprimido de forma a ser gerada uma versão de baixa resolução para a *web*.

[9] http://research.edm.uhasselt.be/~jori/page/index.php
[10] https://www.ffmpeg.org/

Na geração do MPEG-DASH, o MPD é gerado de forma a obter *chunks* de curta duração (2 segundos), de forma a minimizar o atraso em relação ao "*real-time*".

O *Output* realiza as operações inversas do *Input Distributor*: recebe os *Flows* que subscreveu por RTP, recuperando os metadados transportados e, portanto, reconstruindo os respetivos *Grains*. O vídeo volta a ser comprimido, multiplexado em MPEG-TS e enviado para o exterior da *cloud* utilizando RTP.

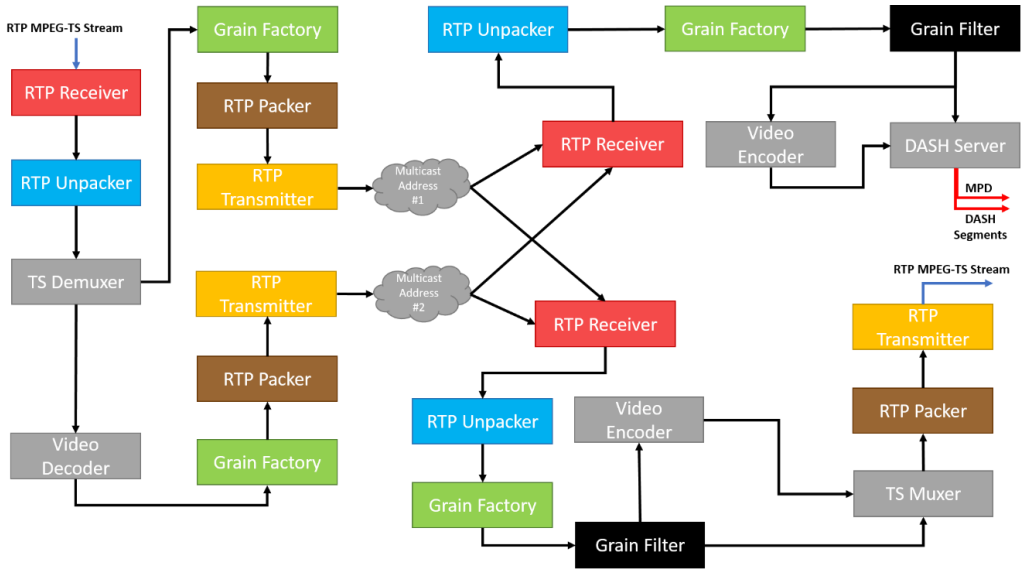

Figura 31: Repetição de tarefas ao longo do fluxo de dados.

Compreende-se, portanto, que existem várias operações que vão sedo repetidas ao longo do ciclo de vida da aplicação, tal como a Figura 31 mostra: cores iguais assinalam o mesmo módulo, enquanto os módulos a cinza são módulos únicos. Estas operações estão maioritariamente relacionadas com interfaces de enviar e receber conteúdo por RTP, pelo que não só podem ser reaproveitadas para os *Nodes* já definidos, como também para possíveis novos *Nodes* que expandirão as atuais funcionalidades.

6.4.3 Interface Web

Foi, também, desenvolvida uma interface *web* simples, de forma a poder observar as *streams proxy*. Para tal, foi utilizado o leitor Dash.JS[11]. Este leitor tem a restrição de apenas reproduzir vídeo com uma amostragem 4:2:0 e um espaço de cor em 8 bits, pelo que o vídeo foi convertido no *Proxy Transcoder* segundo estas restrições.

[11] https://github.com/Dash-Industry-Forum/dash.js

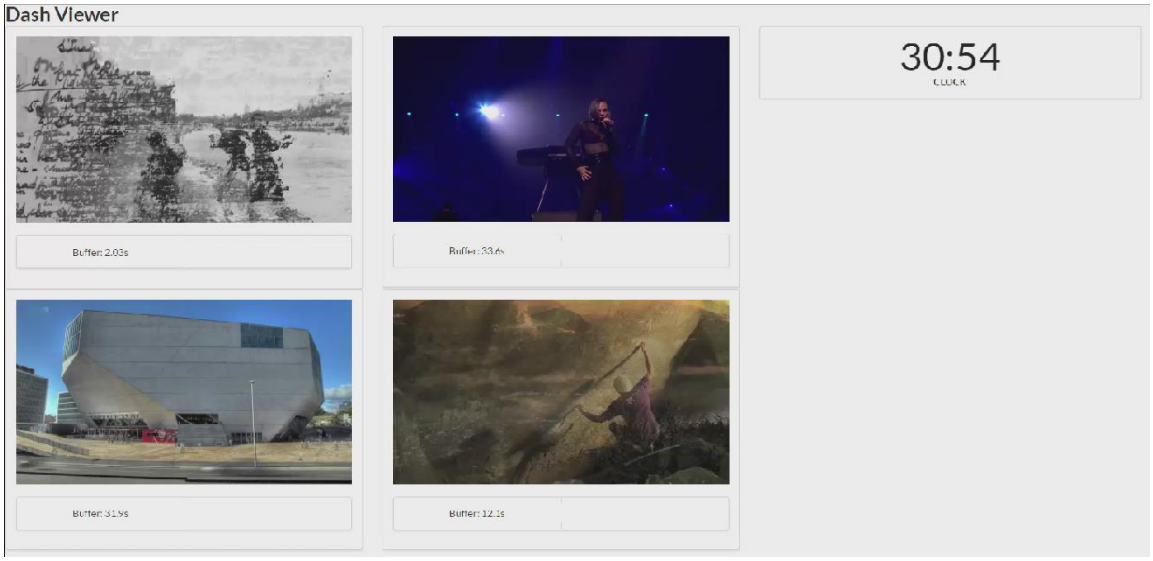

Figura 32: Interface *Web*.

A interface desenvolvida (Figura 32) apresenta as *streams proxy*, ao mesmo tempo que fornece estatísticas do *buffering* realizado pelo leitor. Para além disso, do lado direito podemos observar o relógio, que marca os minutos e segundos, permitindo verificar possíveis atrasos entre as várias *streams*.

Capítulo 7

Resultados e Discussão

Neste capítulo pretende-se estudar métricas relativas aos testes utilizados para garantir o QoS do protótipo desenvolvido, de forma a validar a arquitetura e consequente implementação. Apresenta-se o ambiente de teste, a metodologia utilizada, os resultados e a discussão dos mesmos, avaliado a escalabilidade da solução.

7.1 Ambiente de Teste e Metodologia

Na realização dos testes, foi utilizado um cenário de testes como mostra a Figura 33. No "PC", foi utilizada uma interface de linha de comandos do FFMPEG para transmitir as *streams* MPEG-TS de teste por RTP.

No "Host" foi instalado o Docker Engine, de forma a ser possível correr *containers* Docker. Este ambiente pretende ser um ambiente de teste e desenvolvimento. Ao correrem sobre o *engine* do Docker, os *containers* não têm conhecimento se estão todos a correr na mesma máquina física, aproximando este ambiente de um ambiente virtualizado sobre o qual a aplicação deve correr.

Na Tabela 2 estão descritas as principais características de ambas as máquinas.

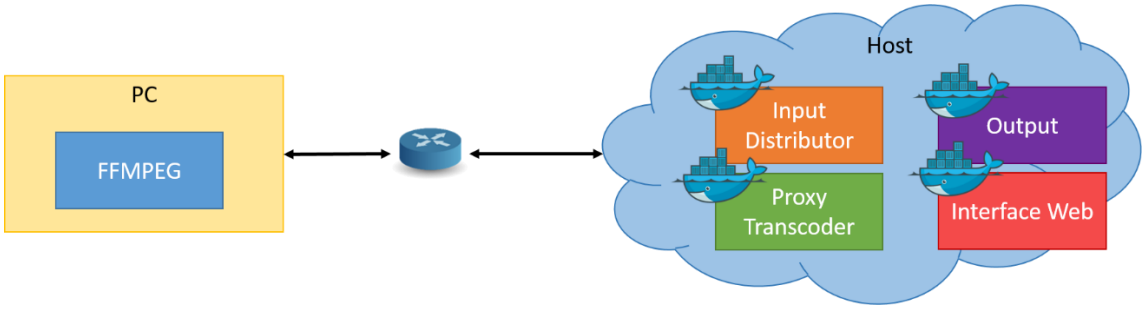

Figura 33: Cenário de testes.

Tabela 2: Características das máquinas de teste.

	PC	*Host*
CPU	Intel Core I7-4770 @ 3.4GHz	2x Intel Xeon CPU E5640 @ 2.67GHz
Memória RAM	8 GB	32 GB DDR3 1333 MHz
Placa de Rede	Intel Ethernet I217-LM	2x NetXtreme II BCM5716 Gigabit Ethernet
Sistema Operativo	Windows 10 Enterprise	Ubuntu Server 14.04 LTS

O *Maximum Transmission Unit* (MTU) da rede é de 1500 bytes, com IGMP e *multicast* ativo.

Foram realizados testes com três vídeos diferentes, cujas principais características estão descritas na Tabela 3.

A utilização de vídeos onde as imagens têm de altura 1088 pixéis deve-se à maior performance por parte da biblioteca *libx264* que o FFMPEG utiliza para o *encoding* e *decoding* de vídeo H.264.

Tabela 3: Características dos vídeos de teste.

	Vídeo #1	*Vídeo #2*	*Vídeo #3*
Tamanho em Disco	206 MB	2.41 GB	34.6 MB
Duração	4 min 27 seg	53 min 56 seg	31 seg
Dimensões	1920x1088	1920x1088	1920x1088
Espaço de Cor	4:2:0	4:2:0	4:2:0
Profundidade de Cor	8 bits	8 bits	8 bits
Modo de Varrimento	progressivo	progressivo	progressivo

7.2 Métricas e Resultados

Para analisar a performance do protótipo desenvolvido, foram monitorizadas as seguintes métricas:

- Largura de banda utilizada.
- Utilização do CPU.
- Utilização da memória RAM.

À entrada da *cloud*, em cada *Input Distributor*, a carga medida encontra-se na Tabela 4. Apesar destes valores serem relativamente baixos, é de realçar que o valor de entrada pode ter valores bastante variáveis.

Tabela 4: Carga medida à entrada de um *Input Distributor*.

	Vídeo #1	*Vídeo #2*	*Vídeo #3*
Média (Mbit/s)	7.70	7.92	7.64
Máxima (Mbit/s)	12.25	18.14	17.84

Analisemos, para o vídeo #3, por exemplo, as métricas estipuladas, de forma a compreender a exequibilidade da arquitetura proposta e do protótipo desenvolvido. A Tabela 5 apresenta os pacotes enviados e recebidos para cada um dos módulos.

7.2.1 Largura de Banda Utilizada

Tabela 5: Tráfego para o Vídeo #3.

	Input Distributor	*Proxy Transcoder*	*Output*
Recebidos	29 650	1 923 978	1 923 978
Enviados	1 923 978	-	29 650

Ao analisar a Tabela 5, compreende-se que não existiu perda de pacotes da transmissão entre os componentes, isto é, todos os pacotes enviados pelo *Input Distributor* foram recebidos pelo *Proxy Transcoder* e pelo *Output*.

De notar que apesar do número de pacotes recebidos pelo *Input Distributor* e enviados pelo *Output* serem o mesmo, não teria forçosamente de acontecer, pois os pacotes podem ter tamanhos diferentes.

Sabendo que o vídeo #3 tem 31 segundos de duração e que o foram transferidos 1 923 978 pacotes, obtemos 62 064 pacotes por segundo.

Como cada pacote tem 1300 bytes de *payload* mais 40 bytes de *header* (RTP/UDP/IPv4), um pacote tem 1340 bytes de tamanho. Dessa forma, cada *Flow* de vídeo implica cerca de, em média, 83 165 760 bytes por segundo, ou seja, aproximadamente 665 Mbits/s.

7.2.2 Utilização do CPU

Quanto à utilização do CPU por parte dos *containers* Docker, que está espelhada na Figura 34, é necessário notar que os três nós analisados realizam operações computacionais bastante exigentes. Daí que cada nó necessite de 2 *cores*, apesar de, em média, não os usar por completo.

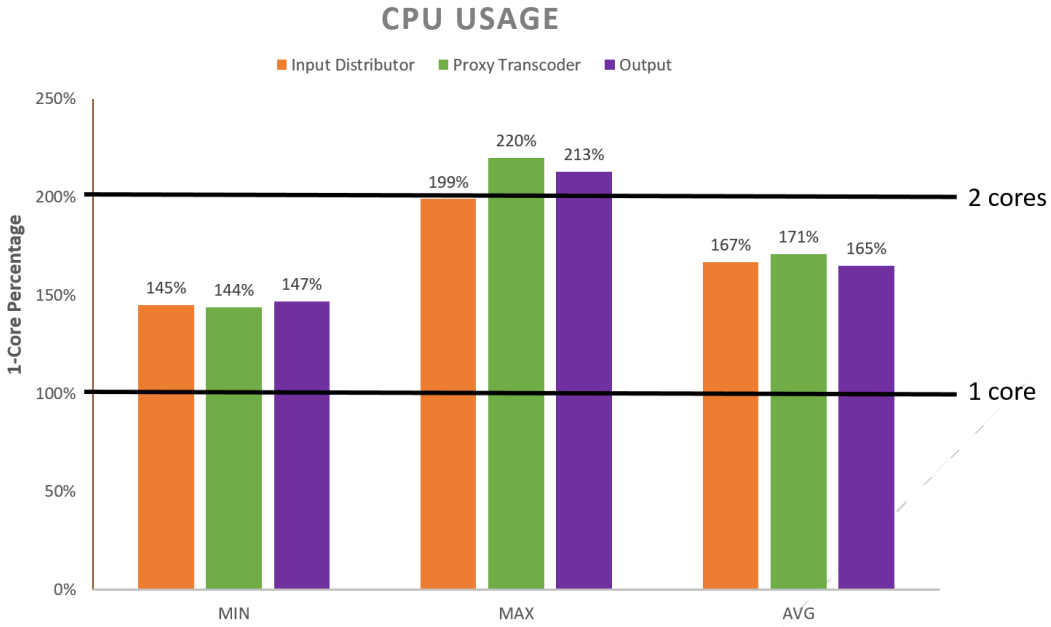

Figura 34: Utilização do CPU por cada *container* Docker.

De notar, ainda, que tanto no *Proxy Transcoder* como no *Output*, o valor máximo que a utilização atingiu ultrapassou a utilização de 2 *cores*, pelo que a quantidade de processamento disponibilizada a cada nó é crítica.

7.2.3 Utilização da memória RAM

A utilização da memória RAM pode ser observada na Figura 35. Ao analisarmos os dados obtidos, podemos concluir não só que os *containers* Docker são bastante leves em termos de recursos utilizados (tal como tinha sido estudado no capítulo 2.2), como também sugere a baixa necessidade de memória RAM para os processos que estão a ser realizados nos nós.

De notar que a utilização da memória RAM está diretamente relacionada com a capacidade de processamento do nó. Ou seja, se a capacidade de processamento de um nó for limitada, tendo em conta as medições do capítulo 7.2.2, a utilização da memória RAM aumentará, dado que será necessário manter um *buffering* de dados maior.

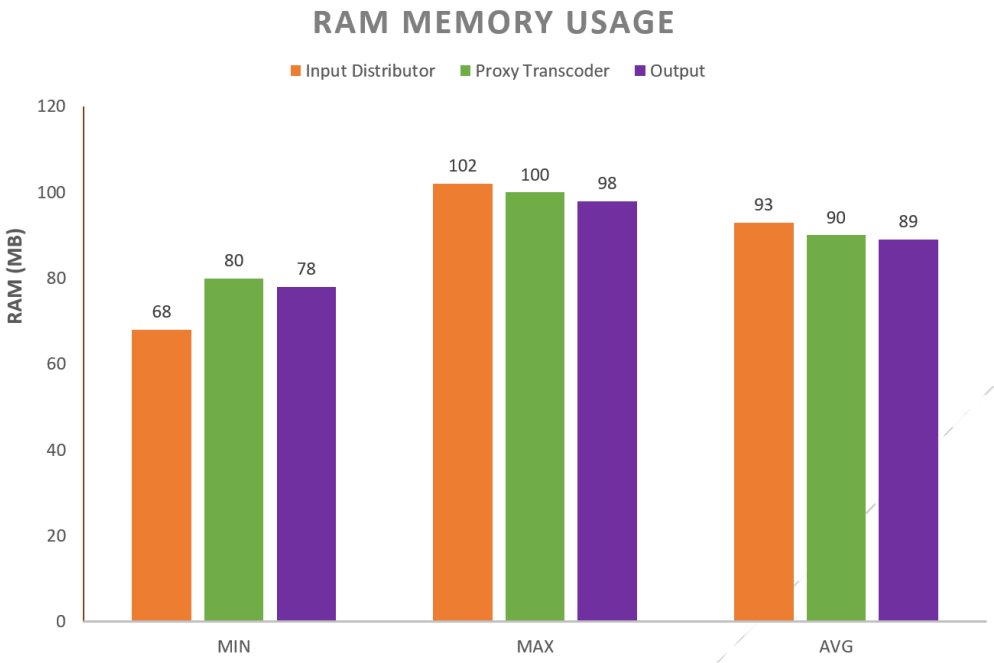

Figura 35: Utilização da memória RAM por cada *container* Docker.

7.3 Escalabilidade

Tendo em conta os resultados observados, denota-se que este cenário pode ser escalado, apesar da utilização do CPU ser uma componente crítica e que poderá precisar de melhoramentos de forma a reduzir a quantidade de recursos necessários.

Atente-se na Figura 36, que analisa, em detalhe as necessidades de CPU, onde são considerados vários cenários de *input* (eixo horizontal), considerando apenas 1 *Output Node*. Se para uma *stream* MPEG-TS de *input* é necessário o processamento equivalente a 5 *cores* do CPU para a totalidade do sistema, isto significa que o dimensionamento da carga é crítico para garantir a escalabilidade do sistema.

De forma a avaliar se é possível relaxar estas necessidades de CPU, é importante analisar qual o *buffering* existente no leitor de DASH, presente na Figura 37. Em média, o leitor contém 1.85 segundos de *buffering*, o que significa que pode ser legítimo sacrificar velocidade de processamento em prol de baixar os requisitos necessários para a aplicação. Ainda assim, está-se a diminuir a confiabilidade

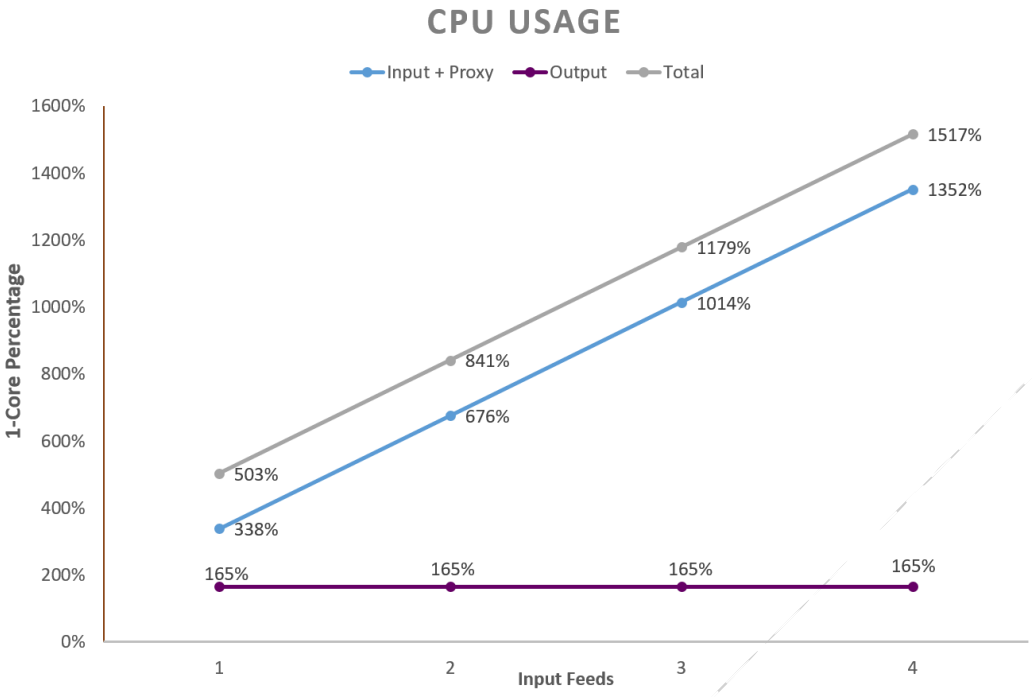

Figura 36: Previsão da utilização de CPU.

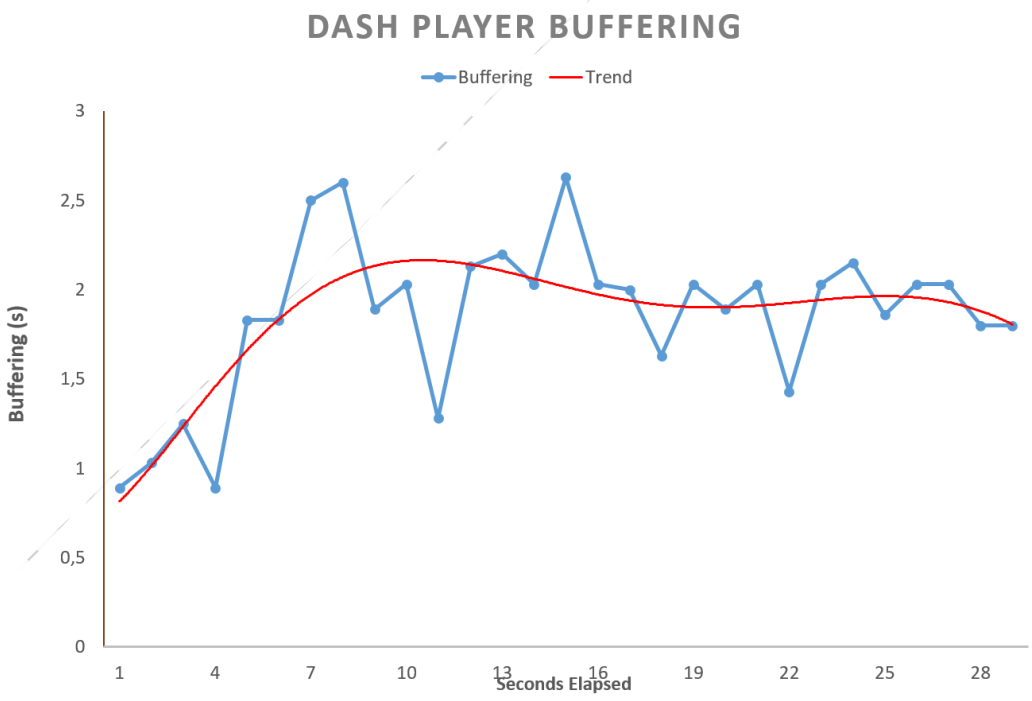

Figura 37: *Buffering* do leitor DASH.

Capítulo 8

Conclusões e Trabalho Futuro

A transição da tecnologia SDI para o IP não significa apenas a alteração de protocolos, formatos e materiais (*routers*, cablagem, entre outros). A adoção do "*IP Studio*" compreende uma nova viragem no paradigma da produção televisiva. A possibilidade de criar novos *workflows* mais flexíveis, com infraestruturas escaláveis e *on-demand* abre portas não só a uma diminuição dos recursos, como também a uma melhor gestão e capacidade de resposta à mudança.

A utilização da *cloud* para serviços que são habitualmente realizados por *hardware* físico especializado, como o *switching* ou o *transcoding* de vídeo, podem ser virtualizados para serviços *on-demand pay-per-use*. Mas as vantagens da *cloud* não se ficam apenas pelo seu modelo de negócio. A possibilidade de facilmente montar uma infraestrutura, em qualquer parte do mundo, sem a necessidade de despender dias a configurar toda a infraestrutura é, por si só, suficiente para ser atrativa.

É certo que o sonho de transportar um estúdio para a *cloud* está longe de ser realizado, mas a virtualização da produção móvel é um passo na direção de se compreender se as tecnologias utilizadas vão de encontro ao objetivo pretendido e idealizado.

Nesta dissertação propôs-se uma arquitetura de alto nível, com base no que o JT-NM e o NMOS propõem, para construir uma aplicação distribuída na *cloud* para a produção de eventos recorrendo a estúdios de produção móvel. Verificou-se não só a exequibilidade da arquitetura (através do protótipo desenvolvido), como também as várias possibilidades de expansão possíveis, tornando-a no início de uma solução possível de ser trabalhada e melhorada futuramente.

Assim, esta dissertação contribui para a aproximação de várias especificações que a indústria televisiva está a levar a cabo. Da mesma forma, transporta tecnologia atual testada com sucesso noutras indústrias, como a *cloud*, *SaaS* e modularização em *containers*, para a indústria televisiva.

8.1 Trabalho Realizado e Satisfação dos Objetivos

Os objetivos propostos inicialmente foram atingidos com sucesso. O trabalho realizado permitiu compreender de que forma se pode virtualizar a produção de eventos remotos em direto, recorrendo a tecnologias testadas por outras indústrias.

A definição de uma arquitetura permite que, através de comunicação por *multicast,* o trabalho realizado possa ser continuado no futuro, de forma a cobrir outras operações na produção televisiva, possibilitando, por exemplo, desenvolver novos *Nodes*.

O protótipo desenvolvido permitiu testar a arquitetura proposta, demonstrando, com sucesso, a possibilidade de transpor a infraestrutura atual para a *cloud*. Para além disso, permitiu compreender as necessidades do meio virtualizado.

8.2 Trabalho Futuro

Ao indicar uma arquitetura de alto nível, esta dissertação inicia uma série de trabalhos, estudos e investigações que podem ter uma grande contribuição para a indústria televisiva.

Desde já, a família de especificações NMOS, à data da realização desta dissertação, ainda não é um trabalho completo. Por si só, sendo um trabalho em aberto, existe a possibilidade de contribuir e participar na construção do NMOS.

A criação de uma camada de *middleware* que permite aos nós descobrirem-se e registarem-se, assim como disponibilizarem conteúdos a outros nós, poderá ser um dos próximos passos no desenvolver da aplicação. Isto poderá permitir, a longo prazo, conseguir aprovisionar um infraestrutura "*plug 'n play*".

Apesar do *Proxy Transcoder* escalar verticalmente, um trabalho bastante interessante seria estudar de que maneira seria possível este escalar horizontalmente, ou seja, conseguir distribuir a carga de *transcoding* por vários nós (por exemplo, um para as linhas verticais, outro para as linhas horizontais, ou um para os *frames* pares e outro para os ímpares).

Da mesma forma, a velocidade das operações de *encoding* e *decoding* no *Input Distributor, Proxy Transcoder* e *Output* são as operações mais críticas no sistema, dado que é necessário manter a natureza *real-time* da aplicação. Assim, é de grande interesse estudar, por um lado, otimizações possíveis a esses processos, e por outro, diferenças de *performance* para diferentes *codecs*.

Por outro lado, os leitores de MPEG-DASH atuais são bastante limitados ao nível dos formatos que suportam. Para além disso, não suportam a sincronização entre vários leitores, num mesmo contexto. Seria bastante interessante realizar trabalhos no sentido de não só alargar os

formatos por eles suportados, como permitir a sincronização entre vários leitores (ideal para um cenário em direto).

Quanto à transmissão de conteúdos em redes IP, seria bastante interessante estudar uma solução para compreender o *buffering* necessário tendo em conta as condições de rede existentes, de forma a equilibrar duas variáveis essenciais: a QoE e o investimento feito por parte das estações televisivas.

Referências

[1] P. Cianci, “Technology and Workflows for Multiple Channel Content Distribution: Infrastructure implementation strategies for converged production,” 2009.

[2] J. Jachetta, “IP to the Camera – Completing the Broadcast Chain,” in *IP Streams, Control, and Production*, 2015, pp. 1.53–29.

[3] A. Krug, “IT-TV-Live An Integrated Concept for IP-based Distributed Broadcast Production with ‘SDI Quality,’” in *IP Streams, Control, and Production*, 2015, pp. 1.57–16.

[4] Z. Zhao, P. Martin, J. Wang, and A. Taal, “Developing and Operating Time Critical Applications in Clouds: The State of the Art and the SWITCH Approach,” *Procedia Comput. …*, 2015.

[5] L. Montalvo, G. Mace, C. Chapel, S. Defrance, T. Tapie, and J. Le Roux, “Implementation of a TV studio based on Ethernet and the IP protocol stack,” in *2009 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting*, 2009, pp. 1–22.

[6] S. V. Gogouvitis, M. C. Jaeger, H. Kolodner, D. Kyriazis, F. Longo, M. Lorenz, A. Messina, M. Montagnuolo, E. Salant, and F. Tusa, “Vision Cloud: A Cloud Storage Solution Supporting Modern Media Production,” *SMPTE Motion Imaging J.*, vol. 122, no. 7, pp. 30–37, Oct. 2013.

[7] A. Kovalick, “The Fundamentals of the All-IT Media Facility,” in *SMPTE 2013 Annual Technical Conference & Exhibition*, 2013, pp. 1–14.

[8] A. S. Davies, “The Silver Lining—Utilizing Cloud Computing in Broadcast Applications,” *SMPTE Motion Imaging J.*, vol. 120, no. 2, pp. 30–36, Mar. 2011.

[9] E. Openshaw and G. Sakata, “Playout Automation in a Virtual Environment,” in *SMPTE 2013 Annual Technical Conference & Exhibition*, 2013, pp. 1–19.

[10] D. Bernstein, “Containers and Cloud: From LXC to Docker to Kubernetes,” *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, Sep. 2014.

[11] D. Liu and L. Zhao, “The research and implementation of cloud computing platform based on docker,” in *2014 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing(ICCWAMTIP)*, 2014, pp. 475–478.

[12] K. Evans, J. Trnkoczy, G. Suciu, V. Suciu, P. Martin, J. Wang, Z. Zhao, A. Jones, A. Preece, F. Quevedo, D. Rogers, I. Spasić, I. Taylor, V. Stankovski, and S. Taherizadeh, “Dynamically reconfigurable workflows for time-critical applications,” in *Proceedings of the 10th Workshop on Workflows in Support of Large-Scale Science - WORKS ’15*, 2015, pp. 1–10.

[13] C. C. Koh, “Next-Generation Techniques to Protect and Secure Realtime IP Media Transport,” *SMPTE Motion Imaging J.*, vol. 122, no. 5, pp. 32–38, Jul. 2013.

[14] J. Footen and M. Ananthanarayanan, “Service-Oriented Architecture and Cloud Computing in the Media Industry,” *SMPTE Motion Imaging J.*, vol. 121, no. 2, pp. 22–30, Mar. 2012.

[15] M. Abdelbaky, J. Diaz-Montes, M. Parashar, M. Unuvar, and M. Steinder, “2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC),” *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*. pp. 368–371, 2015.

[16] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RFC 3550: RTP: A Transport Protocol for Real-Time Applications,” *IETF*, 2003. .

[17] T. Stockhammer, “Dynamic adaptive streaming over HTTP--: standards and design principles,” *Proc. Second Annu. ACM Conf. …*, 2011.

[18] Mpeg-2, “Generic coding of moving pictures and associated audio information -- Part 1: Systems,” *ISO/IEC 13818-1, Ed. 2*, vol. 2000, 2000.

[19] T. S. ETSI, “Transport of MPEG-2 TS based DVB services over IP based networks,” 2007.

[20] G. Fernando, D. Hoffman, V. Goyal, and M. R. Civanlar, “RTP Payload Format for MPEG1/MPEG2 Video.”

[21] T. Kojima, J. J. Stone, J.-R. Chen, and P. N. Gardiner, “A Practical Approach to IP Live Production,” *SMPTE Motion Imaging J.*, vol. 124, no. 2, pp. 29–40, Mar. 2015.

[22] Michael Goldman, “What’s the Best IP Video Path Forward?” [Online]. Available: https://www.smpte.org/publications/past-issues/January-2015. [Accessed: 12-Mar-2016].

[23] P. J. Brightwell, J. D. Rosser, R. N. J. Wadge, and P. N. Tudor, “The IP Studio,” *SMPTE Motion Imaging J.*, vol. 123, no. 2, pp. 31–36, Mar. 2014.

[24] European Broadcasting Union, Society of Motion Picture and Television Engineers, and Video Services Forum, “Joint Task Force on Networked Media - Reference Architecture v1.0.” 2015.

[25] A. Rawcliffe, “Covering the Glasgow 2014 Commonwealth Games using IP Studio,” *BBC R&D White Pap. WHP289*, 2015.

[26] SMPTE, “ST 2022-6: Transport of High Bit Rate Media Signals over IP Networks (HBRMT),” 2012.

[27] M. Laabs, “SDI Over IP—Seamless Signal Switching in SMPTE 2022-6 and a Novel Multicast Routing Concept,” *EBU Tech. Rev. Q*, vol. 4, p. 2012, 2012.

[28] L. Gharai and C. Perkins, “RTP Payload Format for Uncompressed Video.”

[29] T. Committee, *IEEE Std 1588-2008, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, vol. 2008, no. July. 2008.

[30] A. M. W. Association, “Networked Media Open Specification,” 2016. [Online]. Available: https://github.com/AMWA-TV/nmos.

[31] D. Hoffman, G. Fernando, V. Goyal, and M. Civanlar, “RTP payload format for MPEG1/MPEG2 video,” 1997.