On Decomposition and Deployment of Virtualized Media Services

Gourav Prateek Sharma, Didier Colle, Wouter Tavernier, and Mario Pickavet

***Index Terms*—media, broadcast, virtualization, IP, decompostion, SMPTE, 4K**

I. Abstract

For decades, broadcasters have heavily relied on specialized hardware appliances for media transport and processing. However, new architectures, where media transport is realized using IP networking and general-purpose compute hardware is used to run Virtual Media Functions (VMFs), are increasingly being adopted in the broadcast industry. To truly exploit the benefits of these architectures, efficient resource allocation algorithms are needed. Hence, we have proposed an algorithm to optimize a media service's VMF Forwarding Graph (VMF-FG) prior to deployment. To deploy media services, two VMF Placement and Chaining (VMF-PC) algorithms– Next-Fit Placement and Chaining (NFPC) and k-cut Placement and Chaining (k-cutPC) are proposed. The presented evaluation results compare the performance of the two VMF-PC algorithms along with highlighting the improvement in resource allocation as a result of VMF-FG decomposition.

II. Introduction

Today's TV broadcasters today are facing multiple challenges. On the one hand, due to the severe competition in the broadcast industry, the revenue generated through the users has not increased substantially and on the other hand, the user demand for high-quality formats has risen [1]. To match the increase in demand, broadcasters are forced to regularly upgrade their infrastructure that consists of specialized media transport and processing hardware; this leads to a substantial increase in the total expenditures. Moreover, these appliances offer no to a little flexibility, e.g., configuring a the hardware appliance to process high-quality media. As a result, broadcasters are seeking economical and flexible architectures to produce high-quality broadcast content. Internet protocol (IP) has been a de facto standard to interconnect devices within the Internet and also for many other computer networks. Broadcast studios are also witnessing the adoption of IP technology, albeit for a fraction of the total workloads. In addition to that, media processing based on Commercial Off-the-Shelf (COTS) compute platforms is expected to become common in the foreseeable future [2]. The BBC and Grassvalley have already demonstrated the feasibility of general-purpose compute for broadcast applications [3], [4]. To summarize, studio architectures are being explored where media transport is based on IP networking and a general-purpose compute platform is utilized to realize media processing because of two main advantages– (i) cost reduction due to the replacement of costly hardware appliances with general-purpose platforms and (ii) increased flexibility in terms of media service deployment, upgrade and management [1].

Prior to the TV broadcast industry, the telecom industry witnessed a similar transition from proprietary hardware appliances to general-purpose platforms running packet processing functionality in the form of Virtual Network Functions (VNFs). This new architecture that realizes network services using virtualized infrastructures to run VNFs is referred to as Network Function Virtualization (NFV) [5]. Similar to NFV, Media Function Virtualization (MFV) aims to leverage IT virtualization technologies to realize media processing functionality. Implementing a complex media service in a virtual environment involves processing of several uncompressed media streams through a network of software-based media processing function, which are referred here as Virtual Media Functions (VMFs). Deployment of network services in virtualized environments has been studied extensively [5], but that is not true for media service deployments in virtual environments. Further, due to the real-time nature of media production, several unique challenges are faced by broadcasters when transitioning towards MFV. Due to these peculiarities of MFV vis-a-vis NFV, media transport and processing using general-purpose platforms need to be studied to truly exploit the opportunities offered by MFV.

Deployment of a media service in an MFV environment entails the mapping of the media service's VMF Forwarding Graph (VMF-FG) to the underlying infrastructure. The mapping involves an assignment of Virtual Media Functions (VMFs) to the server nodes, along

All authors are from IDLab, Department of Information Technology Ghent University - IMEC (e-mail: gouravprateek.sharma@ugent.be, didier.colle@ugent.be, wouter.tavernier@ugent.be and mario.pickavet@ugent.be) .

with linking VMFs to allow the flow of media traffic between them. As VMF-FG mapping influences the amount of infrastructure required to host media services, it should be performed efficiently. The media traffic between VMFs can be decomposed in multiple sub-streams, each of which represents a different region of the frame [6]. Further, due to the possibility of software-based processing, the decomposed sub-streams can be independently processed using numerous VMFs. These two opportunities unique to the MFV environment give an opportunity to optimize media service deployment. Specifically, the media service's VMF-FG can be decomposed to obtain an optimized VMF-FG that requires less amount of resources when deployed. In our previous work, we demonstrated the benefits of VMF decomposition for static deployment of media services [7]. This paper extends that work by proposing a generalized VMF-FG decomposition algorithm and two algorithms aimed at media service deployment.

In concrete terms, the contributions of this paper are as follows:

1) Formally defining the VMF-FG decomposition problem and proposing a generalized procedure to solve the problem
2) The design of two algorithms for VMF-FG deployment (i) Next-fit VMF Placement and Chaining (VMF-PC) and (ii) k-cut VMF-PC
3) Evaluation of the performance of the proposed VMF-PC algorithms with varying decomposition of VMF-FGs

The rest of the paper is structured as follows. In Section III, we deal with the technical background and related works. The system model, the problem of VMF-FG decomposition and the procedure to solve this problem are presented in Section IV. Section IV also describes the two VMF-PC algorithms– NFPC and k-cutPC. The evaluation of the two VMF-PC algorithms and VMF-FG decomposition is dealt in Section V. Finally, Section VI draws the main conclusions along with the potential future research.

III. Background and Related Works

The popular usage of IP in the TV broadcast industry is in media distribution due to the flexibility it offers over the traditional broadcast methods; IPTV services are deployed over a managed network. Recently, the concept of Over-the-Top (OTT) media has emerged to provide additional services such as Video on Demand (VoD) and interactive TV over the internet; these services were not earlier possible with Cable and Direct-to-home (DTH). By using techniques like Adaptive Bit Rate (ABR) the quality of media is adjusted in accordance to the available bandwidth [8]. Furthermore, there has been an interest to move media workflows to the cloud managed by service providers (e.g., AWS, GCP). In [9], on-premise and cloud-based media broadcast scnearios are compared in terms of protocols and used technologies. The authors have also proposed various hybrid architectures with different amounts of offloading to the cloud. A proof-of-concept for SDN/NFV enabled video transcoding has been proposed in [10]. Agility offered with this solution is a key factor to dynamically adapt media quality with changing network conditions.

The objective of this work is to optimize only the media production workflows in an on-premise facility. Therefore, we are not concerned here with the optimization of workflows in media distribution networks.

A. Media Transport

For decades, broadcasters have employed proprietary baseband technologies for the purpose of media production. Serial Digital Interface (SDI) is one such technology popularly utilized for transporting media streams across broadcast studios [11]. SDI connections are serial data circuits carried over dedicated coaxial cables with BNC connectors. Different SDI standards exist that are used to transport uncompressed media streams of different formats. For example, HD-SDI (SMPTE 292M) interfaces can be used to transport 720p or 1080i video whereas 1080p60 streams are transported using 3G-SDI (SMPTE 424M). The media streams in an SDI-based network are circuit-switched using an SDI switch containing a switching matrix that interconnects the matrix's input to the specific matrix's outputs. The switching matrix operates at the speed equal to the sum of the line speeds of all its ports, resulting in a nonblocking switching operation at all times. In addition, SDI-based media transport in studios have proven to be robust, deterministic and reliable.

Lately, broadcasters are increasingly replacing SDI networking in their studios with IP-based solutions. Although IP has been widely successful in other domains (e.g. telecom) owing to its flexibility, its utilization in broadcast studios has been limited. Mostly, the file-based workflows depend on IP networks to transport the media between different studio devices; for instance, between editing workstations, file servers, and archiving systems. Outside the studios, media contribution and distribution are widely done via IP networks. However, applications such as live media production still rely on SDI-based transport. This is could be attributed to the deterministic performance and robustness of SDI vis-a-vis IP. However, with time the speed of Ethernet switches has increased many folds, up to the point that media transport could now easily be achieved using IP networking. IP also supports

multiplexing, i.e., multiple media streams of different formats can be carried on the same link subject to the link's bandwidth. This allows a gradual up-gradation of the studio infrastructure for high-quality media formats such that the same format-agnostic IP networks can be used until enough bandwidth is available. This contrasts with conventional studios where specialized hardware like SDI routers need to be replaced with new hardware compatible with new media formats. For instance, consider a standard full-HD (FHD) and 4K or ultra-HD (UHD) video having a resolution of 1920x1080 and 3840x2160, respectively, with 4:2:2 sampling, 10 bits per sample, and a frame rate of 30fps. The uncompressed FHD and UHD video streams in this format require 1.244 Gbps and 4.976 Gbps, respectively, on a (physical) link. Thus, theoretically, up to 8 FHD or 2 UHD or, 4 FHD + 1 UHD, streams can be simultaneously carried on a 10G link. Ethernet switches with tens of 10G ports are commercially available so that multiple FHD streams can be switched by these switches simultaneously. Multiplexing along with bi-directionality of IP allows a significant reduction in the amount of cabling required when compared to SDI; thus resulting in cost reduction along with an ease of management. Furthermore, higher-resolution video formats with bitrates touching tens of Gbps, e.g., 4K, can also be transported on an IP network by upgrading the network with COTS 25G or 40G port devices that are expected to become significantly cheaper in the coming years; whereas the upgrade cycles for proprietary SDI switches are very long and expensive.

A broadcast studio facility needs to interconnect multiple media devices. Interconnection based on SDI switching has low-latency is non-blocking, lossless, and supports point to multipoint [12]. These properties must also be supported when transitioning to the IP-routed infrastructure. A multilayer switching architecture, as commonly used in datacenters, can be thought as a single switch interconnecting all the studio devices. Fig. 1 illustrates an all-IP studio architecture, where the switching core is a fat-tree topology connecting media sources (e.g., cameras, microphones), general-purpose compute nodes (e.g., Intel Xeon servers) and media sinks (e.g., monitors, speakers). Although the size of the on-premise switching network is much smaller than that of large datacenters owned by Google or Facebook, but the same fat-tree topology can be used to fulfill the broadcast studio requirements. Moreover, the datacenter topology is an ideal way to interconnect multiple servers that are used to host virtualized media processing functionality, as explained later. These topologies allow an easy upgrade to new media formats (e.g., FHD, UHD) and scaling the number of interconnections (e.g., new cameras) due to the format-agnostic nature of the IP. Grass Valley has built an IP-routed switching network capable of switching 6 Tbps live media traffic at a BBC facility [13]. In summary, by building the studio network along the lines of datacenter networks (e.g. leaf-spine), the speed of the switching fabric can be scaled massively such that multiple uncompressed 4K video streams could also be transported across the studio network built entirely upon IP [12].

Due to the growing popularity of IP for media transport,

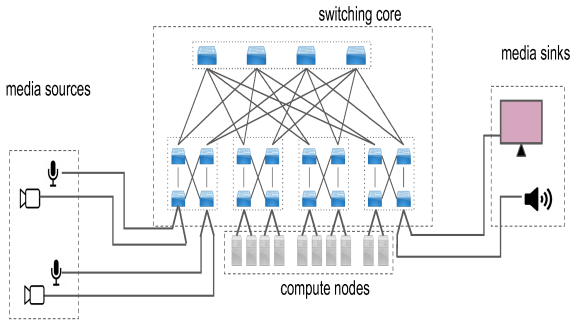

Fig. 1. Illustration of studio architecture based on a datacenter topology.

the Society of Motion Picture and Television Engineers (SMPTE) has released a suite of standards ST 2110 that describe how to transport uncompressed media streams over an IP network [14]–[17]. The ST 2110 suite of standards allows transport and processing of media essences, i.e., video, audio, and ancillary data, as independent streams. By allowing separate elementary essence streams, media production is significantly simplified in contrast to the tightly bundled streams in SDI or ST 2022 [18]. Table I lists different standards of the suite ST 2110 along with a short description.

For the independent transport of media essences,

TABLE I
SMPTE 2110 SUITE OF STANDARDS AND SHORT DESCRIPTION

Standard	Description
ST 2110-10/-20/-30 [14]–[16]	addressing system concerns and uncompressed video and PCM audio streams
ST 2110-21 [19]	specifying traffic shaping and delivery timing of the uncompressed video
RP 2110-23 [6]	specifies methodologies for splitting high bandwidth single video essence streams into several lower bandwidth streams
ST 2110-31 [20]	specifies the real-time, RTP-based transport of AES3 signals over IP networks, referenced to a network reference clock
ST 2110-40 [17]	transport of ancillary data packets

SMPTE ST 2110-10/-20/-30 specifies use of the Real-time Transport Protocol (RTP) which itself is based on the User Data Protocol (UDP). At the essence sender, the data (e.g., video, audio, or ANC) is broken into multiple segments that go into the RTP payload, which is then attached to the RTP header containing the required flags and fields. The RTP packets are recursively encapsulated while transcending the lower layers of the networking stack. The packets on the network are independently transported to the receiver where they are reassembled by referring to the RTP sequence number contained in the header. The payload is then passed to the application/device where the full frame is generated for further processing or display.

1) Media Decomposition: Uncompressed high-resolution video streams, e.g., 4K or 8K, are difficult to transport and process given their high bandwidth and therefore high compute requirements. By decomposing a high-bandwidth stream into multiple low-bandwidth streams, this issue can be alleviated. Given this challenge, the SMPTE released a recommended practices document RP 2110-23 that describes three mechanisms through which high-bandwidth streams can be split [6]. The first scheme is called Phased decomposition where a high frame rate stream is decomposed into multiple low frame rate streams, also known as "phases". This decomposition scheme is particularly useful when working within an environment consisting of high-speed cameras with high refresh rates, e.g., 120 Hz. Sample interleave decomposition is the second scheme where M-way splitting of a high-bandwidth stream results in M sub-streams each carrying frames of resolution $1/M$ that of the original stream. Lastly, the third stream decomposition method– Square Division (SD) is explained next.

M-way SD decomposition of a media stream results in splitting of each frame of the original stream into M quadrants each carried by a different sub-stream. Fig. 2 illustrates 4-way decomposition of a 4K video stream. The original 4K (3840x2160p30) stream with the bitrate 4.976 Gbps when 4-way decomposed results in four 2K (1920x1080p30) sub-streams each with bitrate 1.244 Gbps. Each 2K stream can be further decomposed again using 4-way SD such that a total of sixteen 960x540p30 sub-streams are obtained. The transport and processing of multiple low-bandwidth streams (2K or 960x540p30) is clearly easier when compared to a single high-bandwidth (4K) stream. This paper only considers the SD decomposition method as it can be exploited for VMF-FG decomposition, which has been explained in detail in section IV.

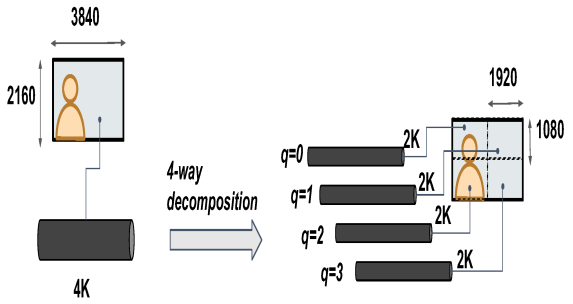

Fig. 2. An example of 4-way SD decomposition of 4K video stream.

B. Virtualized Media Processing

Similar to the use of proprietary equipment for media transport, media processing in broadcast studios has been dominated by proprietary hardware appliances. This is due to the high performance offered by specialized hardware platforms [2]. COTS platforms consisting of general-purpose compute nodes (e.g., Intel Xeon servers), however, have lately become significantly faster (Moore's law) so that they can be used to run Media Functions (MFs). For instance, consider an MF that mixes two video streams with a special effect (e.g. wipe or dissolve transitions). Today, broadcasters use a specialized hardware appliance called vision-mixer to achieve this functionality. This functionality, however, can also be realized in a software running on a COTS server [21]. The production quality vision-mixer has been implemented using open source tools like OBS studio and KX studio running on a general-purpose platform. The resulting vision mixer provides a real-time delay of 1.4s, which is acceptable for multi-camera production. Furthermore, the processing delays in software-based media production facilities can be reduced by exploiting several optimizations, e.g., kernel offload mechanisms such as DPDK and netmap [22], hardware-acceleration of media processing using FPGAs and GPUs [23], [24]. As the focus of this paper is the media service deployment problem so performance optimization mechanisms shall not be discussed in this paper.

Exploiting general-purpose compute platforms instead of dedicated hardware to host services is not limited only to the TV broadcast industry. Telecom operators are adopting Network Function Virtualization: an architecture where packet processing functionality, which is usually realized by specialized hardware appliances called middleboxes, is being realized using Virtual Network Functions (VNFs) running on the general-purpose compute platforms. Analogous to the VNFs in NFV, media processing can be realized using software-based implementations of MFs running in a virtualized envi-

ronment; we refer to these software MF implementations as Virtual Media Functions (VMFs). Tab. II lists some example VMFs along with a short description for each of them.

TABLE II
EXAMPLE VMFS WITH SHORT DESCRIPTION.

Name	Notation	Description
Chroma key	$chrm\text{-}key$	Replaces the background (narrow range of colour) of a video stream with another video stream, e.g., weather presenter background.
Picture-in-picture	pip	Inserts a small resolution video stream into a large resolution video at a given coordinate.
Video quality assessment	vqa	Assesses the amount of distortions introduced to the media by a VMF.
Brightness Adjustment	$brt\text{-}adj$	Adjusts the pixel values of a video stream according to the correction signal produced by a VQA VMF.

The BBC has built prototypes for live IP production and also carried out a live multi-site all-IP UHD production trial at the Glasgow Commonwealth Games in 2014 [25]. Partnering with Isotama, they have also demonstrated the live mixing of video streams using a software-based video processing pipeline that can be controlled through a browser application [3]. Additionally, Grassvalley's has released Agile Media Processing Platform (AMPP) which is a microservice-based solution that leverages elastic compute of the COTS platforms to run a variety of media processing workflows [4]. To summarize, media production using general-purpose compute infrastructure is quite feasible and broadcaster are expected to adopt them with time.

Taking inspiration from NFV, we define MFV as an architecture where media services are implemented using Virtual Media Function (VMFs) running on general-purpose compute platforms. Fig. 3 shows a simplified view of the MFV architecture. The lowest layer in the architecture is the MFV Infrastructure (MFVi) layer that contains all resources, i.e., both physical and virtual, required to run VMFs along with the virtualization layer that is responsible for providing the required isolation between running VMFs. Depending on the type of virtualization technology used, the virtualization layer can be a Hypervisor (type 1 or 2) if Virtual Machines are to be deployed or it can be a Docker engine if containers are to be used. Above the MFVi layer, lies the VMF layer that consists of VMFs, in the form of VMs or containers, where actual media processing occurs. The VMFs can be chained together to realize a complex media service (top layer) as explained in the next section.

The Control and Optimization (CO) layer in Fig. 3 is analogous to the Management and Orchestration (MANO) layer in the NFV architecture.

The role of the CO layer in MFV is multi-fold. First, it manages resources (physical and virtual) through the use of some infrastructure management tools such as Openstack.

Second, it is responsible for managing the state of one or more VMFs by performing tasks like update, query, scaling, healing, and termination of the VMFs. The operation of various VMFs needs to be altered at times according to the requirements of the director. For example, the director may need to switch through a number of camera feeds throughout an event. This can be done using a switcher VMF that takes as an input all camera feeds and switches to a particular stream according to the control signal sent by the director using a controller to the switcher VMF. The CO layer is responsible for the distribution of control signals to the deployed VMFs.

Third, the CO layer performs service orchestration by coordinating various resources across the MFVi layer. To this end, the CO layer takes various inputs such as VMF Placement and Chaining (VMF-PC) algorithms, the media service representation (e.g., VMF-FG), profiles containing VMFs' resource demands, the QoS requirements of the media services, etc, to do resource allocation by running the selected VMF-PC algorithm that outputs the required VMF-FG-to-MFVi mapping, that is used to reserve physical resources in the MFVi layer. By carefully designing and selecting the appropriate VMF-PC algorithm, the QoS requirement of the media service can be met while efficiently utilizing the resources.

The decomposition of high-bandwidth streams in an

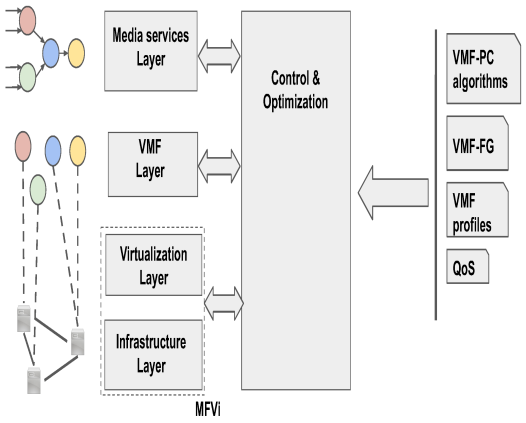

Fig. 3. An overview of the MFV architecture.

MFV environment provides an opportunity to further decompose the VMFs of a VMF-FG; thus it can result in improving resource utilization, as shall be discussed in the next section.

IV. System Model

In this section, we formalize the MFV network model, describe the VMF-FG decomposition and deployment problem, and present two algorithms to solve them.

The traditional media production environments are inflexible in terms of transport and processing of media streams due to their dependence on specialized hardware. Therefore, IP-networking for media transport combined with virtualization of media processing workflows seems to be well suited for future broadcast studios. The media services in these environments can be denoted using a directed graph referred to as VMF-FG as explained later in this section.

MFV is more than just the softwarization of MFs and then its deployment on the COTS platform. Virtualization of MFs opens several opportunities that were not earlier possible with the physical implementations of MFs. Before the deployment of a media service, VMF-FG can be optimized such that fewer resources are consumed compared to un-optimized VMF-FG post-deployment. The optimization can be done by decomposing the VMFs, distributing the switching functionality, and using the state of downstream VMFs to enable/disable upstream VMFs, etc.

Before discussing the VMF-FG decomposition and deployment problems, we first formalize the MFV model.

The notations used for various parameters, variables and procedures along with a short description are listed in Tab. III. We model the MFVi physical network as a connected directed graph $G^I = (N, E)$. Here, N and E denote the set of physical nodes and links, respectively, in the MFVi network. A subset of nodes $N_c \subset N$ are COTS server nodes that have the required resources to run VMFs. The rest of the nodes $\forall n \notin N_c$ are just forwarding nodes, i.e., switches or routers. As each node $n \in N_c$ has a limited amount of resources (e.g. CPU, RAM, disk, etc), we just use cpu_n to represent the resource capacity of node n, here it is the total number of CPU cores. Each physical link $e = (n_i, n_j) \in E$ has an associated physical bandwidth denoted by bw_{n_i,n_j}.

A media service s is realized by allowing the media traffic to flow through a specific arrangement of VMFs. The arrangement of VMFs defining the service is represented by a directed acyclic graph (DAG) $\mathcal{G} = (\mathcal{F}, \mathcal{L})$, here referred to as VMF Forwarding Graph (VMF-FG). A node $f \in \mathcal{F}$ of the VMF-FG represents an endpoint (media sources or sinks), or a VMF, whereas an edge $l = (f_i, f_j) \in \mathcal{L}$ is the virtual link connecting the VMFs f_i and f_j. An endpoint is either from the set of media sources $\mathcal{F}_{src}$ that includes cameras, playout servers or from the set of media sinks $\mathcal{F}_{snk}$ that includes multiviewer screens, file-servers (archives, OTT, etc) and the broadcast transmitter (DVB-T Tx). Each virtual link $l \in \mathcal{L}$ for a video processing service is annotated by the parameters such as the frame resolution (e.g., 2K, 4K), the color space (e.g., RGB, YCbCr), the sub-sampling scheme (e.g., 4:2:2, 4:2:0), and the refresh rate (e.g., 24fps or 30fps), where the frame resolution is P_hxP_v, P_h is the number of pixels in horizontal direction and P_v is the number of pixels in the vertical direction. Similarly, for an audio stream, the annotation contains parameters such as the sampling frequency (e.g., 48kHz), the number of audio channels (e.g., 1, 2, 8), etc.

Fig. 4 shows $\mathcal{G}$, the VMF-FG representation of an example media service. In this VMF-FG, $\mathcal{F} = \{src0, src1, src2, chrm\text{-}key, scl, vqa, pip, dst0, dst1\}$ and $\mathcal{L} = \{(src0, chrm\text{-}key), (src1, chrm\text{-}key), (src2, scl), (chrm\text{-}key, pip), (scl, pip), (chrm\text{-}key, vqa), (vqa, dst0),$

TABLE III
Notations used in the system model for parameters, variables and procedures.

Notation	Description
$G^I = (N, E)$	Directed graph representation of the MFVi network, where N and E are the set of the physical nodes and links, respectively.
N_c	Set of all server nodes, i.e., nodes with compute resources.
cpu_n	CPU resources (in number of cores) present on a server node $n \in N_c$.
bw_{n_i,n_j}	Available bandwidth available on the physical link $(n_i, n_j) \in E$.
$\mathcal{G} = (\mathcal{F}, \mathcal{L})$	VMF-FG representation of a media service, where $\mathcal{F}$ and $\mathcal{L}$ are the set of VMF and virtual links, respectively.
$\mathcal{F}_{src}$	Set of all media sources in $\mathcal{G}$.
$\mathcal{F}_{snk}$	Set of all media sinks in $\mathcal{G}$.
$P_h(l)$, $P_v(l)$	Number of pixels on a frame corresponding to the virtual link $l \in \mathcal{L}$.
$Size(l)$	Total number of pixels on a frame corresponding to the virtual link $l \in \mathcal{L}$.
$fps(l)$	Refresh rate of the stream on the virtual link $l \in \mathcal{L}$.
P_f	Set of all port in the VMF f.
$Upstr(f^p)$	Upstream VMF on the port $p \in P_f$ of the VMF f.
M	Decomposition parameter by which each virtual link is decomposed, where $M = 4^m, m \geq 0$.
$\mathcal{G}_{sw}$	A subgraph in the VMF-FG $\mathcal{G}$ consisting of only switching VMFs.
$\mathcal{G}_{sw}^{set}$	The set of all subgraphs $\mathcal{G}_{sw}$ in the VMF-FG $\mathcal{G}$.
$tx_{u,v}$	u-to-v Transmit condition.
$Tx_{u,v}^{snk}$	u-to-v Cumulative transmit condition when starting the traversal from the sink VMF snk.
α	Variable containing VMF-to-node assignment mapping.
γ	Variable containing Virtual link -to- physical link mapping.
$\mathcal{F}_f^{nbrs}$	The set of upstream VMF neighbours of $f \in \mathcal{F}$.
$used\_res$	Variable containing currently reserved resources.
cap	Resource capacity of physical nodes and links in G^I.

$(pip, dst1)\}$. Here, the set of video sources is $\mathcal{F}_{src} = \{src0, src1, src2\}$ and the set of video sinks is $\mathcal{F}_{snk} = \{dst0, dst1\}$. The parameters associated with the streams corresponding to all virtual links take values as follows: YCbCr for the color space, '4:2:2' for the sub-sampling scheme, and 30fps as the refresh rate. The link $(scl1, pip)$ has a resolution of 646x364 and all other links have the resolution of 1920x1080, except for $(vqa, dst0)$. The VMF $chrm\text{-}key$ applies the chroma-keying operation on the its inputs that are connected to the two video sources $src0$ and $src1$, to produce the output where the background (e.g., green pixels) of the first stream ($src0$) is replaced by the background stream produced by the second source ($src1$). VMF vqa assesses the quality of $chrm\text{-}key$'s output and stores the results in $dst1$. The output of the $chrm - key$ VMF may contain some artifacts due to, e.g., low CPU allocation or delayed / lost packet in one of its input stream. The output of $chrm\text{-}key$ is also mutlicasted to pip that embeds a low-resolution video stream received on its second input. Using the scaler VMF $scl1$, the second input is generated by scaling down the video stream produced by $src2$. The output of pip VMF is then terminated at the sink $dst1$. Similarly, a VMF-FG can be used to represent any other complex media service. Next, we define the VMF-FG

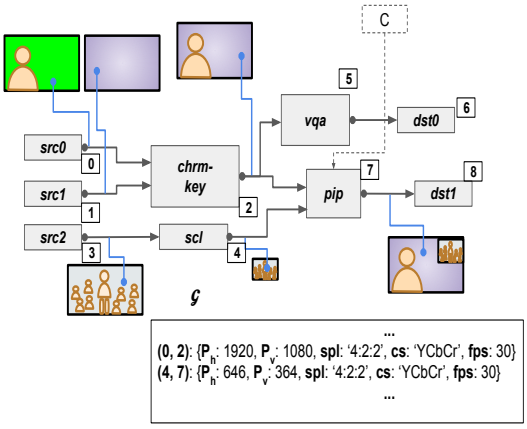

Fig. 4. The VMF-FG $\mathcal{G}$ representation of an example media service.

decomposition problem and describe an algorithm to produce the M-way decomposition of a given VMF-FG.

A. VMF-FG Decomposition

Given a VMF-FG $\mathcal{G} = (\mathcal{F}, \mathcal{L})$, the M-way decomposition of $\mathcal{G}$ is another VMF-FG $\mathcal{G}'$ that is functionally equivalent to $\mathcal{G}$ but all of its virtual links are M-way decomposed. Here, we assume that the decomposition scheme used is SD decomposition. Decomposing the virtual links of a VMF-FG further allows the decomposition of the VMFs in of the given VMF-FG. With VMFs and virtual links being decomposed, several optimization can be applied to the VMF-FG without altering the overall functionality. By solving the VMF-FG decomposition problem, we imply that a functionally equivalent but optimized VMF-FG is generated. The optimized VMF-FG is generally less resource demanding than the original VMF-FG and thus can be deployed more efficiently.

Next, we describe an algorithm we have proposed to solve the VMF-FG decomposition problem.

The VMF-FG decomposition algorithm takes an undecomposed VMF-FG, $\mathcal{G} = (\mathcal{F}, \mathcal{L})$ along with the decomposition parameter M and returns the M-way decomposed VMF-FG. The decomposition parameter $M = 4^m$, where $m \geq 0$, indicates the level of decomposition for each virtual link in the VMF-FG. For example, $M = 1$ ($m = 0$) implies no decomposition, whereas $M = 4$ ($m = 1$) implies a 4-way decomposition as shown in Fig. 2. For the sake of simplicity, we assume that all VMFs are Multiple Input Single Output (MISO) type, similar to the one shown in Fig. 5. A MISO VMF can have multiple input ports such that each of its input ports is connected to the output port of only one upstream VMF. The single output port of the VMF f is simply denoted by the VMF itself, i.e., f, whereas the p^{th} input port of the VMF is denoted by f^p. In Fig. 5, the VMF f connected to the port p of the VMF f is denoted by $Upstr(f^p)$ and the frame size on the virtual link $l = (Upstr(f^p), f)$ is simply denoted as $Size(Upstr(f^p))$, e.g. if $Upstr(f^p) = u_p$, $Size(u_p, f^p) = Size(Upstr(f^p), f^p) = Size(f^p)$. The single output port of the VMF can be multicasted to multiple destinations, e.g., the raw camera footage being processed by some VMF can also be archived at the same time for later use. In Fig. 5, the output of f is multicasted to D destination VMFs denoted by $d_0, d_1, d_2, ..., d_{D-1}$ from the single output port 'f'.

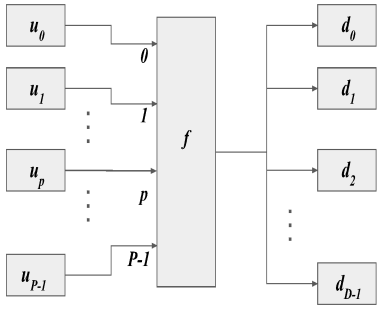

Fig. 5. An illustration of a MISO VMF f.

A VMF with multiple output ports can be easily

decomposed into multiple VMFs with single output ports. For example, consider the Multiple Input Multiple Output (MIMO) VMF f, shown in Fig. 6 (a), where its P inputs ports are connected to the upstream VMFs $u_0, u_1, u_2, ..., u_{P-1}$ and similarly its P' output ports are connected to the downstream VMFs $d_0, d_1, d_2, ...d_{P'-1}$ on separate unicast links. The VMF f internally implements a function g, operating on a input vector $I = [u_0, u_1, u_2, ..., u_{P-1}]$, whose output is then shared to produce the outputs $h_0(g(I)), h_1(g(I)), h_2(g(I))...h_{P'-1}(g(I))$. The MIMO VMF f can hence be decomposed into $n+1$ MISO VMFs as shown in Fig. 6 (b). First, the VMF g is applied over I to produce a single output g, which is then multicasted to the inputs of P' VMFs $h_0, h_1, h_2, ..., h_{P'-1}$, respectively. In this example, each output port of f is connected to a single downstream VMF on a unicast link. However, MIMO VMFs having multicast links at their output ports can be decomposed in a similar manner. This way, any MIMO VMF in a VMF-FG can be transformed into a set of MISO VMFs before proceeding to the first step of the VMF-FG decomposition algorithm.

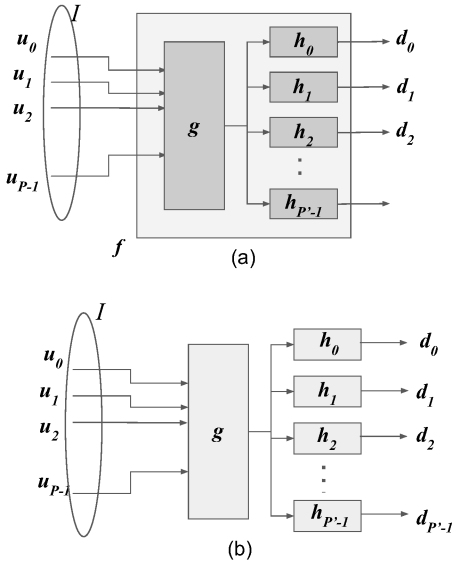

Fig. 6. (a) Internals of the MIMO VMF f and (b) its decomposition into MISO VMFs g, h_0, h_1, ..., $h_{P'-1}$

The VMF-FG decomposition algorithm consists of four steps as shown in Fig. 7.

Virtual Link Decomposition: The first step of the procedure is the decomposition of all virtual links of the VMF-FG by M. Each link $\forall(f_i, f_j) \in \mathcal{L}, f_i = Upstr(f_j^p)$, is replaced by M (f_i, f_j) links, where $f_i = Upstr(f_j^{p'}), p' = [pM, (p-1)M-1]$. As we have assumed the SD decomposition in our procedure, the

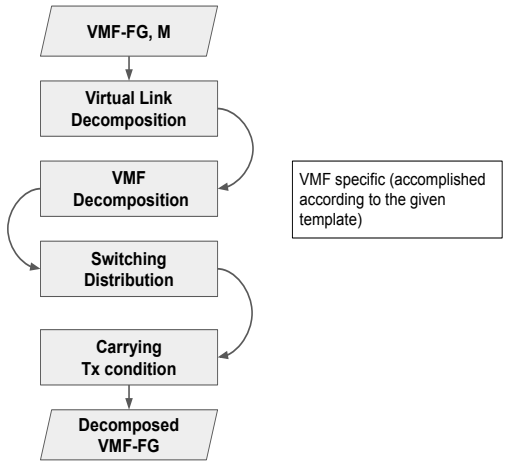

Fig. 7. Flowchart showing the VMF-FG decomposition procedure.

frame size of the virtual link (f_i, f_j) after decomposition becomes $Size'(f_j^{p'}) = Size(f_j^p)/M = (P_h P_v)/M$, here P_h and P_v are number of pixel in horizontal and vertical directions, respectively, in the frame corresponding to (f_i, f_j) before decomposition. For example, consider the part of the VMF-FG showing two neighbouring VMFs f_i and f_j as shown in Fig. 8 (a) whereas Fig. 8 (b) shows link decomposition of (f_i, f_j) into M new links.

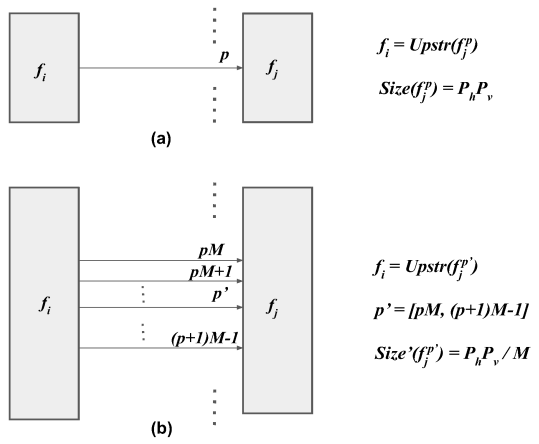

Fig. 8. Illustration for the virtual link decomposition step. (a) The virtual link $(f_i, f_j) \in \mathcal{L}$ and (b) its decomposition by M.

The forwarding graph returned after the first step is denoted as $\mathcal{G}^1$.

VMF Decomposition: After the first step, the number of input and output ports of each VMF in the resulting forwarding graph $\mathcal{G}^1$ are grown M times. Next, $\mathcal{G}^1$ is

transformed into a forwarding graph only containing VMFs with single outputs. In this step, each MIMO VMF is replaced by a forwarding graph consisting of MISO (switch and non-switch) VMFs connected through multicast links. The forwarding graph nodes, i.e., MISO switch and non-switch VMFs, are then appropriately connected to the control signals.

The VMF decomposition procedure is specific to each VMF. This can be provided as a template/code by the developer of the VMF that contains the steps to generate the required forwarding graph and distribute the control signals to the VMFs.

Next, we show how a $chrm\text{-}key$ and picture-in-picture (pip) VMF can be decomposed.

The VMF decomposition step for $chrm\text{-}key$ is quite straightforward as shown in Fig. 9 (c). Prior to the VMF decomposition, the link decomposition step is performed, as shown in Fig. 9 (b). Each link in Fig. 9 (a), i.e, $(s_0, chrm\text{-}key)$, $(s_1, chrm\text{-}key)$ and $(chrm\text{-}key, d_0)$ is decomposed into $M = 4$ links corresponding to the four quadrant of a frame. As each output pixel of $chrm\text{-}key$ VMF is computed using the corresponding input pixels only, the chroma-keying operation can be performed in each quadrant independent of each other. Therefore, $chrm\text{-}key$ can be decomposed into $chrm\text{-}key0$, $chrm\text{-}key1$,$chrm\text{-}key2$ and $chrm\text{-}key3$, each responsible for producing output for quadrants $q = 0$, $q = 1$, $q = 2$ and $q = 3$, respectively.

Next, we discuss a more complex example of VMF decomposition.

Consider the pip VMF shown in Fig. 10 (a). The pip VMF inserts the frames of the video stream on port 1 into the frames of the video stream on port 0 at a given coordinate in the upper-right quadrant. The controller connected to the pip VMF is used to control the operation of the pip VMF, i.e., it signals if the second stream is to be inserted in the first stream or not. In other words, if say $ctrl == 0$, the pip VMF is disabled and the output of the pip VMF is the same as that of s_0 and if $ctrl == 1$ the pip operation is enabled on the output. Fig. 10 (b) shows the pip VMF and its neighbours after performing the virtual link decomposition step by $M = 4$. Before decomposing the pip VMF, its upstream VMFs s_0 and s_1 are decomposed into $s_{0q}, \forall q \in [0, M-1]$ and $s_{1q}, \forall q \in [0, M-1]$, respectively, each corresponding to the four quadrants, as shown in Fig. 10 (c). As in this example, picture insertion occurs in the upper-right ($q = 1$) quadrant; thus processing will be required for this quadrant only. This implies that after pip VMF decomposition, the VMF pip_1 ($q = 1$), which is responsible for performing pip operation in the upper-right quadrant will be instantiated. Moreover, as no picture is inserted in the other three quadrants, no processing is required for these quadrants, i.e., the pip VMFs – pip_0, pip_2 and pip_3, corresponding to the upper-left ($q = 0$), lower-left ($q = 2$) and lower-right ($q = 3$) quadrants, respectively, are not instantiated (Fig. 10 (c)). A two-port switch VMF is also instantiated that takes the output of s_{01} at port 0 and pip_1 at port 1. Based on the $ctrl$ value, the sw VMF switches between its inputs, i.e., if $ctrl == 0$, the sw VMF outputs the s_{10}'s output whereas if $ctrl == 1$, the sw VMF outputs the pip_1's output. For other three quadrants, the respective upstream ($s_{0q}, \quad \forall q = \{0, 2, 3\}$) and downstream VMFs ($d_{0q}, \quad \forall q = \{0, 2, 3\}$) are connected.

Other VMF types such as split-screen, switcher (with transition effects, e.g., wipe), etc, can be also decomposed similarly.

The forwarding graph returned after the second step is denoted as $\mathcal{G}^2$

Switch Functionality Distribution: Following the decomposition of all VMFs, the resulting forwarding graph $\mathcal{G}^2$ consists of MISO switch and non-switch VMFs linked by multicast virtual links. Next, we describe how the functionality of switch VMFs can be distributed to the VMFs that are upstream and downstream to the switch VMF.

The switch VMF sw shown in Fig. 11 has two inputs connected to the upstream VMFs u_0 and u_1 and its output is connected to the downstream VMF d. The functionality of the switch VMF sw can be distributed to the upstream VMFs by allowing triggering the correct upstream VMFs to transmit to the correct downstream VMF(s) based on the value of the control signal $ctrl$. The control signal is wired to u_0 and u_1 to allow the distribution of sw functionality. For $ctrl == 0$, only u_0 transmits to d and u_1 transmits to d if $ctrl == 1$.

The forwarding graph $\mathcal{G}^2$ obtained after the VMF decomposition step can contain groups of switch VMFs chained together with multicast links between two or more non-switch VMFs as shown in Fig. 12. These subgraphs with switch VMFs (blue cloud) and multicast links can be simplified as explained next.

First, the subgraphs $\mathcal{G}_{sw}^{set}$ consisting of switch VMFs only are isolated by removing all non-switch VMFs from $\mathcal{G}^2$ and then performing the graph traversal algorithm to find all the connected components consisting of switch VMFs. The switching functionality of each isolated subgraph is then distributed to its upstream VMFs. Assume the subgraph $\mathcal{G}_{sw} = (\mathcal{F}_{sw}, \mathcal{L}_{sw})$ is one such sub-graph ($\mathcal{G}_{sw} \in \mathcal{G}_{sw}^{set}$) obtained from $\mathcal{G}$ that consists of only switch VMFs and the control signals to all switch VMFs in $\mathcal{G}_{sw}$ are represented using a single value $ctrl$. The task is to distribute the switching functionality of $\mathcal{G}_{sw}$ to its upstream (non-switch) VMFs. The set of

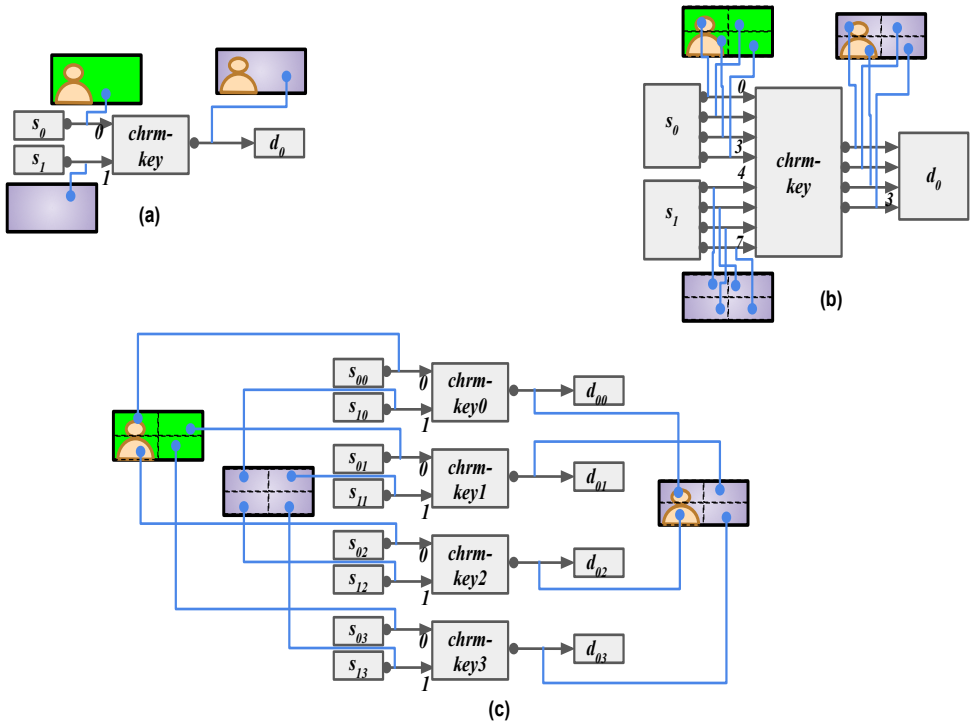

Fig. 9. 4-way decomposition of the *chrm-key* VMF. (a) Illustration of operation of the *chrm-key* VMF. The *chrm-key* VMF after (b) the virtual link decomposition step and (c) the VMF decomposition step.

non-switch VMFs whose output is connected to the VMFs in $\mathcal{G}_{sw}$ is denoted by $\mathcal{U}_{sw} = \{Upstr(f^p) | \forall f \in \mathcal{F}_{sw}, \forall p \in P_f; Upstr(f^p) \not\subset \mathcal{F}_{sw}\}$. For each VMF $u \in \mathcal{U}_{sw}$, first, the set of non-switch VMFs $\mathcal{D}^{u,ctrl_0}$ downstream to $\mathcal{G}_{sw}$ whose input is connected to u is determined along with the corresponding control signal, say $ctrl = ctrl_0$, that results in the connection. Next, a multicast link is created for $ctrl = ctrl_0$ and the VMFs in $\mathcal{D}^{u,ctrl_0}$ are added to this multicast link. Fig. 12 (a) shows a part of VMF-FG with a switching subgraph $\mathcal{G}_{sw}$ that needs to be distributed. Assume the VMF u serves the output ports of the VMFs $v_0^{ctrl_0}, v_1^{ctrl_0}, v_2^{ctrl_0}, ...v_{n-1}^{ctrl_0}$ of $\mathcal{G}_{sw}$ when the control signal value is, say, $ctrl_0$. At the output of u, we create an output port (multicast link) corresponding to $ctrl_0$ and also add the inputs of the downstream VMFs $\mathcal{D}^{u,ctrl_0} = \{d_0^{ctrl_0}, d_1^{ctrl_0}, d_2^{ctrl_0}, ...d_{n-1}^{ctrl_0}\}$ to this multicast link as shown in Fig. 12 (b). Similarly, corresponding to other control signal values, additional mutlicast links (on separate output ports) are created for u and downstream VMFs are added to those mutlicast links. Then, the process is repeated for the rest of the upstream VMFs $u \in \mathcal{F}_{sw}$. Lastly, the switching subgraph $\mathcal{G}_{sw}$ is removed from $\mathcal{G}^2$.

Afterwards, the above steps are repeated for the rest of the switching subgraphs in ($\mathcal{G}_{sw}^{set}$).

The upstream VMFs of switching subgraphs in $\mathcal{G}^2$ have now multiple outputs. However, at any instance in time, one control combination is active and thus traffic flows only on a single multicast link. Also, the downstream VMFs ports listen to many multicast links out of which only one has an input on it at any instance of time.

The forwarding graph returned after the third step is denoted as $\mathcal{G}^3$

Transmit condition Propagation: In $\mathcal{G}^3$, if a control signal value results in the VMF f having no destination VMFs to transmit then the media processing done by f is not utilized, thus the control signals for f can be used to switch-off computation of f and its upstream

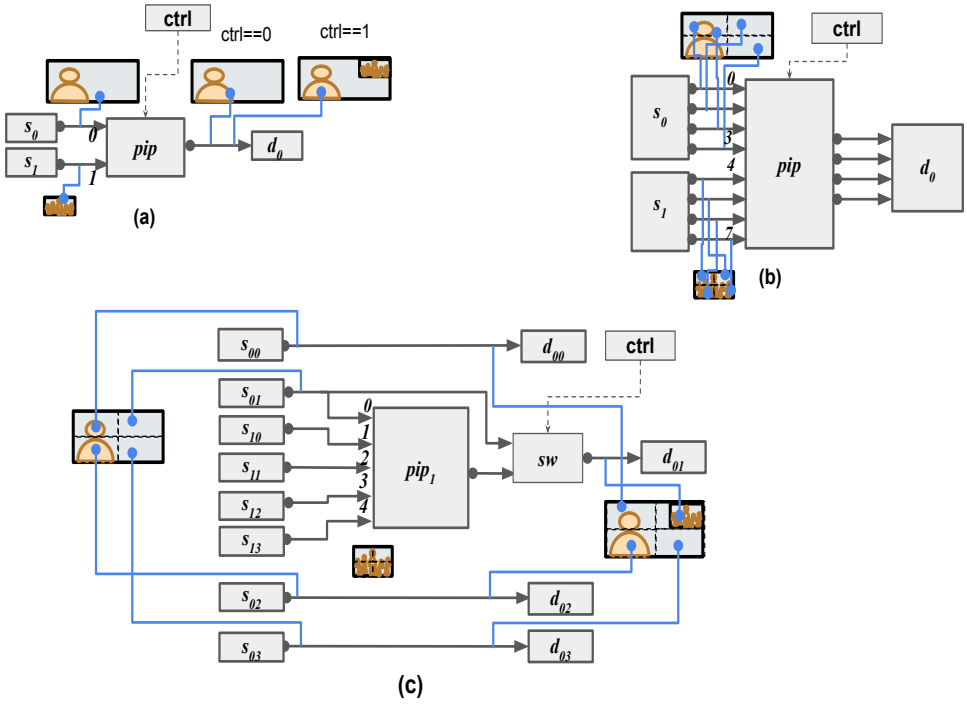

Fig. 10. 4-way decomposition of the *pip* VMF. (a) Illustration of operation of the *pip* VMF. The *pip* VMF after (b) the virtual link decomposition step and (c) the VMF decomposition step.

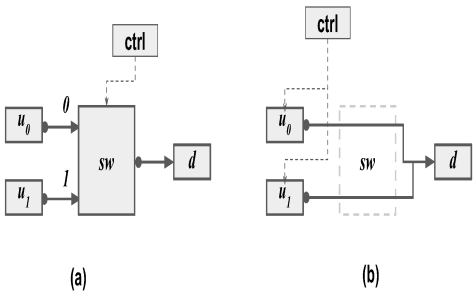

Fig. 11. Illustration showing the distribution of switching functionality of a VMF to its upstream VMFs.

VMFs that only feed f.

Starting from a sink node $snk \in \mathcal{F}_{snk}$, we travel in the reverse direction using the DFS algorithm [26] and only backtrack when we have encountered a source node $f_{src} \in \mathcal{F}_{src}$. During the traversal, transmit condition of the upstream VMF are updated according to its current transmit conditions and the transmit condition of its downstream VMF. Consider the three consecutive VMFs u, v and w during a path traversal starting with snk through $\mathcal{G}^3$, as shown in Fig. 13. Assume, during this traversal the cumulative transmit condition of v (to w) is $Tx^{snk}_{v,w}$ and the transmit condition from 'u to v' is $tx_{u,v}$. Then, the cumulative transmit condition for the VMF u to v for this traversal is $Tx^{snk}_{u,v} = Tx^{snk}_{v,w}$ AND $tx_{u,v}$, where AND is the binary AND operation.

Similarly, the VMF-FG is traversed one-by-one starting from the remaining sink VMFs meanwhile updating the transmit condition of the VMFs. The same VMF can be encountered during various traversals from different sinks; the transmit condition in such cases is OR between VMF's transmit conditions in all traversals.

The forwarding graph returned after the fourth step is the required decomposed VMF-FG $\mathcal{G}'$.

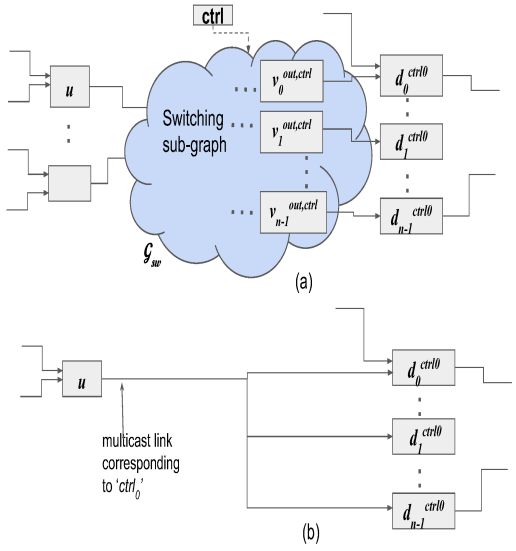

Fig. 12. Distribution of switching functionality of a switching sub-graph to its upstream VMFs. (a) VMF-FG region showing a switching subgraph and (b) its distribution to the input VMF u for a given control value $ctrl_0$.

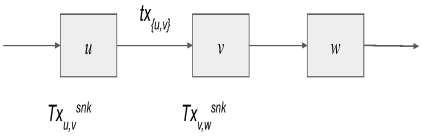

Fig. 13. Propagation of transmit conditions from the downstream v VMF to the upstream VMF u.

B. VMF-FG Deployment

In this section, we describe two different VMF-PC algorithms that can be used to deploy a VMF-FG $\mathcal{G}$, decomposed or not, on a given MFVi network G^I. The first algorithm is next-fit based VMF-PC algorithm we refer to as NFPC algorithm and the second algorithm is based on min k-cut algorithm we refer to as $kcut$-PC.

1) Next-fit Approach: The pseudo-code for the next-fit based VMF placement and chaining algorithm is shown in Alg. 2. The inputs to the algorithm are i) the directed graph representing the VMF-FG $\mathcal{G} = (\mathcal{F}, \mathcal{L})$ of the media service, ii) the directed graph representing the MFVi $G^I = (N, E)$ and iii) the resource capacity cap of the physical nodes and links in G^I, i.e., cpu_n and bw_{n_i,n_j}. The algorithm outputs the variables α and γ that denotes VMF-to-node mapping and denotes the virtual link -to- physical path mapping, respectively. Before starting, variable initialization is done: qu_{vmf} representing a queue, $used\_res$ representing the physical resources currently used in the procedure, α, and γ are all initialized with ϕ.

Algorithm 1: Procedure for Next-Fit search

```
1  Procedure NextFit(f, α, used_res,
     G = (F, L), G^I = (N, E), cap):
       /* start with the last used
          node in N_c                          */
2      for n in N_c do
3          links_f ← {(f, f_i) | ∀(f, f_i) ∈ L} ;
            /* virtual links associated
            with f */
4          pths ← chainVMFs(f, α, used_res,
            links_f, G^I, cap, n) ;  /* chains f
            with neighb. VMFs */
           /* check resources on n for
              α                                */
5          if enghRes(f, used_res, G^I, cap, n)
            then
6              return n, pths;
7          end
8      end
9      return None, None;
10 end
```

Each sink node snk is considered as a starting node for the BFS traversal through the VMF-FG (l. 3-17) [26]. Until the queue qu_{vmf} is empty, the following procedure is repeated.

qu_{vmf} is first de-queued and assigned to f. Using the `NextFit` procedure shown in Alg. 1, placement and chaining of f is attempted. The `NextFit` procedure starts searching through N_c server nodes starting with the last used node and returns with the node having enough resources to place f and chain it with the previously placed VMFs. If the PC of f is not successful, the algorithm is stopped (l. 6). Otherwise, α is updated with the node mapping of VMF f, γ is updated with the physical path mapping of the virtual links linking f with its down-streams VMFs, $used\_res$ is updated according to the resource demands of f (l. 6). Next, up-stream neighbours f' of f are assigned to $\mathcal{F}_f^{nbrs}$. Then, a *for* loop is used to loop over $\mathcal{F}_f^{nbrs}$. If a VMF $f' \in \mathcal{F}_f^{nbrs}$ is already placed, it is chained with f (l. 10); otherwise, it is en-queued to qu_{vmf}. If the PC of all the VMFs is successful, the resource capacity in MFVi (G^I) is updated by referring to $used\_res$ variable that contains the current resource usage.

Assuming, the maximum number of VMFs in a VMF-FG is constant, i.e., $|F| \leq F_{max}$, it can be concluded from Alg. 1 and Alg. 2 that the asymptotic time complexity of the NFPC heuristic is linear in the number of requests and total server nodes, i.e., $\mathcal{O}(|R| * |N_c|)$.

2) k-cut Approach: The pseudo-code for k-cutPC algorithm is shown in Alg. 5. The output of the

Algorithm 2: Algorithm for next-fit based VMF-PC

```
   /* VMF-FG and MFVi network                    */
   Input  : G = (F, L), G^I = (N, E), cap
   /* VMF placement and chaining
      mapping                                    */
   Output : α, γ
   Initialize: qu_vmf, used_res, α, γ
1  for snk in F_snk do
2      En-queue snk to qu_vmf;
3      while qu_vmf ≠ φ do
4          De-queue f from qu_vmf;
5          NextFit(f, α, used_res, G, G^I, cap)
            ;     /* Place and chain f */
6          If successful update α, γ, and used_res
            else stop procedure;
7          Assign upstream neighbours of f to
            F_f^nbrs;
8          for f' in F_f^nbrs do
9              if f' is placed then
10                 Chain f and f';
11                 If successful update γ and
                    used_res else stop procedure;
12             end
13             else if f' ∉ qu_vmf then
14                 En-queue f' to qu_vmf
15             end
16         end
17     end
18 end
   /* All VMFs have been PC'ed                   */
19 Update cap using used_res;
```

algorithm is same as that of the NFPC algorithm, whereas the input additionally includes n_{bst} shown in Alg. 2. Here, n_{bst} is the number of best k-cuts that are checked for VMF-PC. Before starting, $used\_res$, α, and γ are initialized with ϕ.

k-cutPC algorithm attempts to first partition each connected component of VMF-FG $\mathcal{G}$ and then deploy them using the NFPC algorithm shown in Alg. 2. The algorithm starts with the minimum possible k value $k_{min} = \sum_{f \in \mathcal{C}} cpu_f / \max_{n \in N}\{cpu_n\}$, n_{bst} best k_{min}-cuts of $\mathcal{G}$ are returned using the procedure `NbstKcut`.

The pseudo-code for the procedure `NbstKcut` is shown in Alg. 4. It uses the random contraction algorithm `RandContr` to get a k-cut of an un-directed graph G [27]. The quality of the generated k-cuts, in terms of the sum of the weights of the crossing edges, can be boosted by repeating `RandContr` N_{runs} times (l. 4-13) and n_{bst} best k-cuts are generated by running `RandContr` N_{runs} times. By passing $n_{bst} = 10$, the ten best cuts are returned that are sequentially checked for PC in Alg. 5.

One-by-one PC is attempted on each k-cut $(\mathcal{F}_{cut}, \mathcal{L}_{cut}) \in \mathcal{F}_{cuts}, \mathcal{L}_{cuts}$ in the order of increasing weight until all the VMF clusters of the selected cut $((\mathcal{F}_{cut}, \mathcal{L}_{cut}))$ are placed and chained successfully (l. 6-12). If a component $\mathcal{C}$ could not be deployed for any cut $(\mathcal{F}_{cut}, \mathcal{L}_{cut}) \in \mathcal{F}_{cuts}, \mathcal{L}_{cuts}$, the procedure (l. 5-13) is repeated for an incremented value of k until the maximum value of $k = k_{max} =\mid \mathcal{C} \mid$. If a component $\mathcal{C}$ could not be deployed for $k \in [k_{min}, k_{max}]$, the deployment has failed and the algorithm stops (l. 15). In case, all the connected components $\mathcal{C} \in \mathcal{C}_{cpts}$ of $\mathcal{G}$ are placed, resource capacities in G^I are updated by referring to $used\_res$.

The k-cutPC algorithm trades off speed for the amount of used network bandwidth. As the asymptotic running time of `RandContr` is $\mathcal{O}(|L|)$ and it is called $N_{runs} = |F|^{2k-2} * \log|F|$ times from `NbstKcut`, the time complexity of `NbstKcut` is $\mathcal{O}(|F|^{2k-2} * \log|F| * |L|)$. Assuming $|F| \leq F_{max}$, the overall time complexity of the k-*cut*PC is also $\mathcal{O}(|R| * |N_c|)$, albeit the scaling constant $|F|^{2k-2} * \log|F| * |L|$ results in a lower speed of k-cutPC as compared to NFPC.

Algorithm 3: Procedure for randomized graph contraction [27]

```
1 Procedure RandContr(G = (F, L), k):
2     while | F |> k do
3         Pick a link l = (f_i, f_j) randomly from L
           with probability ∝ bw(l);
4         Merge f_i and f_j into a single node;
5         remove self-loops;
6     end
7     wt ← Sum of weight of all edges crossing
       last k nodes;
8     return G, wt
9 end
```

V. Evaluation

In this section, we evaluate the impact of VMF-FG decomposition on media service deployment and evaluate the performance of the proposed VMF-PC algorithm. First, the given VMF-FG is decomposed using the proposed VMF-FG decomposition procedure for a given M and then the resulting VMF-FG is deployed using either of the two VMF-PC algorithms– and their performance is evaluated. The evaluation is done in terms of four metrics (i) acceptance ratio, (ii) resource reservation, (iii) resource utilization, and (iv) end-to-end hops latency. The acceptance ratio gives a

Algorithm 4: Procedure for generating n_{bst} best k cuts

Procedure NbstKcut($G = (F, L)$, k, n_{bst}):
  $G_{cltrs}, W_{cltrs} \leftarrow \phi, \phi$; /* Set of generated cuts and corresponding weights */
  $N_{runs} \leftarrow \mid F \mid^{(2k-2)} * \ln \mid F \mid$;
  while $N_{runs} > 0$ **do**
    G_{cut}, $wt_{cut} \leftarrow$ RandContr(G, k) /* get a random k-cut */
    if $\mid G_{cltrs} \mid \leq n_{bst}$ **then**
      $G_{cltrs} \leftarrow (G_{cltrs} \cup G_{cut})$ /* gather n_{bst} best cuts */
    end
    /* replace worst (max. weight) cut with G_{cut} */
    else if $wt < \max W_{cltrs}$ **then**
      $G_{rep} \leftarrow G \in G_{cltrs}$ with $wt > wt_{cut}$ /* k-cut with min. weight but > wt_{cut} */
      Replace G_{rep} in G_{cltrs} with G_{cut};
    end
    $N_{runs} \leftarrow N_{runs} - 1$;
  end
  return G_{cltrs};
end

Algorithm 5: Algorithm for the k-cut based VMF-PC.

/* VMF-FG, MFVi network and # of best k-cuts */
Input : $\mathcal{G} = (\mathcal{F}, \mathcal{L})$, $G^I = (N, E)$, cap, n_{bst}
/* VMF placement and chaining mapping */
Output : α, γ
Initialize: $used\_res$, α, γ
$\mathcal{C}_{cpts} \leftarrow$ ConccCompts($\mathcal{G}$) ; /* all connected components of $\mathcal{G}$ */
for $\mathcal{C}$ **in** C_{cpts} **do**
  Calculate k_{min}, k_{max} values;
  for k **in** $[k_{min}, k_{max}]$ **do**
    ($\mathcal{F}_{cuts}$, $\mathcal{L}_{cuts}$) = NbstKcut($\mathcal{C}$, k, n_{bst}) /* get n_{bst} best k-cuts */
    for *($\mathcal{F}_{cut}, \mathcal{L}_{cut}$)* **in** *($\mathcal{F}_{cuts}$, $\mathcal{L}_{cuts}$)* **do**
      PC of cluster ($\mathcal{F}_{cut}, \mathcal{L}_{cut}$) using Alg. 2;
      if *PC successful* **then**
        Chain cluster ($\mathcal{F}_{cut}, \mathcal{L}_{cut}$) with other VMFs;
        If cluster chaining successful, update α, γ, $used\_res$ and **break**;
      end
    end
    break loop **if** $\mathcal{C}$ is PC'd;
  end
  If $\mathcal{C}$ still not PC'd, stop the algorithm;
end
/* All connected components have been PC'ed */
Update cap using $used\_res$;

measure of how likely a VMF-FG can be deployed on a given MFVi. The resource reservation indicates the amount of compute and network resources that are expected to be used by the PC algorithm to deploy the given set of VMF-FGs. Post-deployment, the control values for VMFs vary, resulting in varying real-time usage of resources, which is indicated by the resource utilization metric. End-to-end latency of the deployed VMF-FG is an important parameter in live media production. This metric can be measured in terms of the number of hops (server nodes) from a source VMF to a sink VMF in the deployed VMF-FG. As the number of hops increases, the end-to-end is expected to increase and vice versa.

Next, we explain the simulation settings used to perform the evaluations.

A. Simulation settings

The MFVi physical network considered for the evaluation is fat-tree data center topology. The topology consists of κ pods, where each pod has $(\kappa/2)^2$ server nodes, $\kappa/2$ access layer switches, and $\kappa/2$ aggregate layer switches and the core layer contains $(\kappa/2)^2$ switches. The total number of server nodes in the topology are $\kappa^3/4$. Fig. 14 shows a data center in fat-tree topology with four pods ($\kappa = 4$), each pod containing two aggregate switches, two edge switches, and four server nodes whereas the core layer contains four switches.

The parameters and their corresponding values (range) for media service requests, MFVi, and VMFs are listed in Table IV. The formats of the video streams are HD and UHD with resolutions 1920x1080 and 3840x2160, respectively, and a refresh rate of 30fps. The sub-sampling assumed here is 4:2:2 where each sample is encoded with 10 bits. Each server node carries 24 CPU cores and each VMF (un-decomposed) consumes 6 cores per 1000 Mbps of the input bandwidth. The physical network is based on devices with 10GbE interfaces.

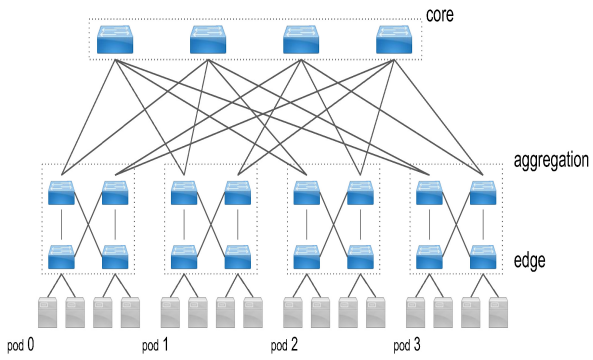

Fig. 14. Fat-tree topology of a data center with $\kappa = 4$ pods.

The VMF-FG decomposition algorithm and both VMF-PC algorithms are written in Python and we have used networkx package wherever necessary. We use Intel Xeon machine @2.40GHz and 12GB RAM to carry out the simulations. Each experiment is repeated ten times and the corresponding mean and standard deviation are reported. For each media service request, we select one VMF-FG from a set of three VMF-FGs shown in Fig. 15.

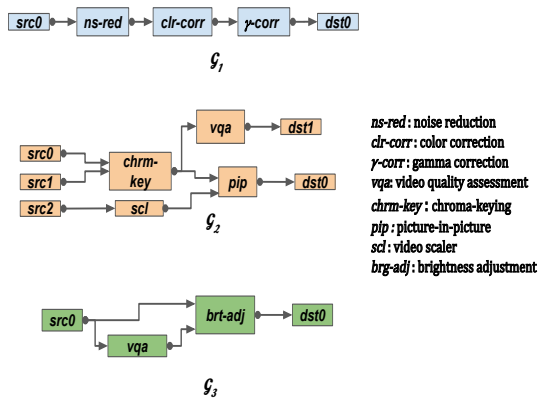

Fig. 15. Set of VMF-FGs used in the evaluation.

B. Acceptance ratio

In this evaluation, we report the average acceptance ratio of media service requests for deployment on a small data center topology with $\kappa = 4$. The arrival of requests is modeled as a Poisson process with an average arrival rate of 3 requests per 100 time units (tu) and the request lifetime is exponentially distributed with an average of 1000 tu; it is the average time a media service is deployed on the MFVi. A media service request is deemed accepted if its VMF-FG has been deployed successfully, i.e., all the VMFs have been assigned to server nodes and the virtual links are mapped to physical paths in the MFVi network. Acceptance ratio at time t is defined as the ratio of the total number of requests accepted until time t to the total number of requests that have arrived until t.

TABLE IV
DEFAULT VALUES/RANGE OF VARIOUS PARAMETERS INVOLVED IN THE SIMULATION EXPERIMENTS.

Parameter	Value or range
κ	4, 8
Request arrival rate (Poisson)	3/(100 units)
Request lifetime (Exponential)	1000 units
CPU/VMF/bw	6 cores/(1000 Mbps)
Video resolution	1920x1080p30
Frame rate	30fps
CPU/node	24 cores
Ph. link BW	10Gbps

The acceptance ratio variation for NFPC and k-cutPC for t = 0-10000 has been shown in Fig. 16 (a) and Fig. 16 (b), respectively. Also, to show the impact of VMF-FG decomposition, the acceptance ratio variation has been plotted for $M = 4$ and $M = 16$. Both in Fig. 16 (a) and Fig. 16 (b), roughly acceptance ratio decreases with time as the total number of deployed media services increases and the amount of available physical resources for the newly arriving requests decreases. It can be clearly observed from the plots that the acceptance ratio increases when M increases.

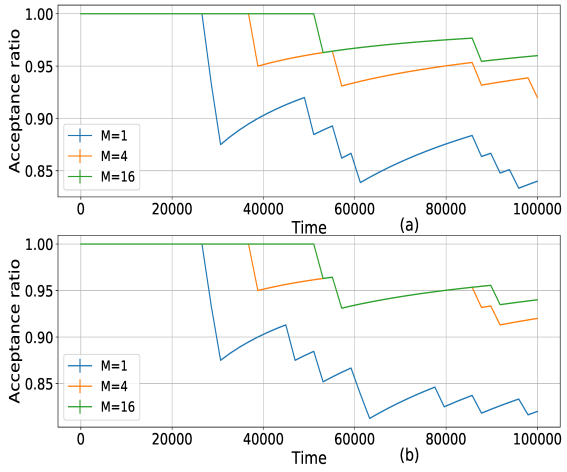

Fig. 16. Acceptance ratio variation over time for the deployment of 50 requests on the MFVi with data center topology $\kappa = 4$ (16 server-nodes) for (a) NFPC and (b) k-cutPC algorithms.

C. Resource reservation

Here we report the physical resources reserved by the VMF-PC algorithm when deploying a set of media

service requests on a bigger MFVi network in fat-tree topology, with $\kappa = 8$. The total amount of resources present in this topology is sufficient to deploy fifty media service requests. Fig. 17 (a) shows the total number of server nodes reserved by the NFPC and k-cutPC algorithms when deploying fifty requests with decomposition parameter $M = 1, 4, 16$. The total number of server nodes used by the NFPC algorithm is slightly less compared to the k-cutPC algorithm. This inefficiency in k-cutPC is because the placement is carried out at the VMF cluster level whereas the NFPC algorithm carries out placement at the VMF level. Another observation is the reduction in the number of used server nodes N_{srv} for both VMF-PC algorithms with increasing M. For instance, when M is increased to sixteen from one, N_{srv} decreases by about 20% for both NFPC and k-cutPC. Therefore, VMF-FG decomposition results in better consolidation of VMFs on the server nodes. Fig. 17 (b) shows the total number of physical links N_{lnks} reserved for both the VMF-PC algorithms. Similar to Fig. 17 (a), the NFPC algorithm performs a bit better than the k-cutPC algorithm because of sequential PC in NFPC as opposed to the cluster level PC in k-cutPC. Here, an increasing decomposition does not significantly affect the number of reserved physical links.

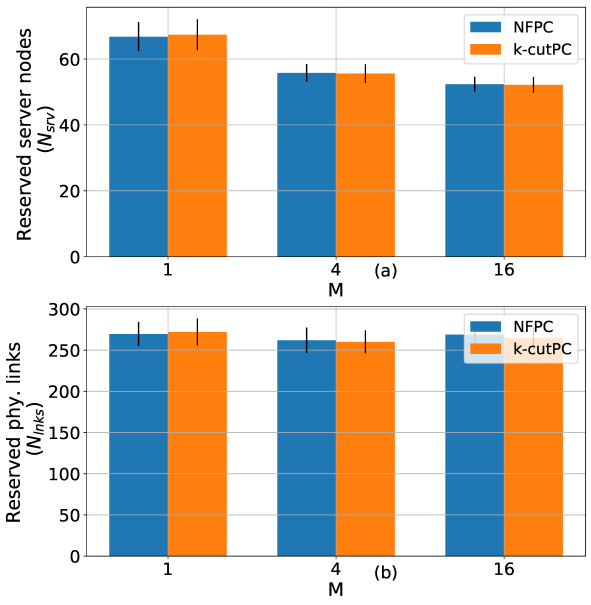

Fig. 17. Variation of resource reservation with M for 50 requests on a data center topology with 128 server-nodes ($\kappa = 8$). (a) Total reserved server nodes and (b) total reserved physical links.

D. Resource utilization

After the deployment of a media service on the MFVi network, the media traffic is processed by the VMFs. Based on the traffic and control signal values, the physical resource usage of the media service can vary. Here we report the mean and worst case resource utilization by the already deployed media services. The mean resource utilization of a deployed media service is averaged over the resource utilization values corresponding to all control signal values, whereas the worst case utilization refers to the minimum resource utilization value corresponding to a particular control signal value for that media service.

Fig. 18 (a) shows both the mean case and the worst-cast normalized CPU usage for the NFPC and k-cutPC algorithms. As expected, the CPU utilization is better for NFPC compared to k-cutPC because of better VMF consolidation in NFPC, which further improves with the increasing value of M resulting in the better CPU utilization for the worst case and the mean case for both VMF-PC algorithms.

In addition to the CPU utilization, inter-node bandwidth usage is also a useful parameter to compare VMF-PC algorithms. Fig. 18 (b) shows the mean and worst case bandwidth (in Gbps) consumed by the deployed media services for both VMF-PC algorithms. It is evident from the plots that the total bandwidth utilized by k-cutPC is less than with NFPC. Since k-cutPC attempts the deployment of a partitioned VMF-FG where inter VMF-cluster bandwidth is minimized resulting in lesser bandwidth utilization than NFPC. Again, with increasing M, the bandwidth utilization of the deployed media services is reduced as the chances of neighbouring VMF placed on the same node increases with M.

E. End-to-end Hops

For a media service, the end-to-end latency is an important parameter that can be used to check if the timing requirement of a media service is met. Since, in this section we only want to highlight the impact of VMF-FG decomposition on end-to-end delay, we use the number of hops along the longest path in VMF-FG as a metric instead of end-to-end delay. A stricter analysis for end-to-end delay requires packet scheduling of media streams on a time-sensitive network but it is out of the scope of this paper. As the impact of VMF-PC algorithm is different for different VMF-FG, we evaluate end-to-end hops separately for each VMF-FG. Fig. 19 (a), (b) and (c) shows average end-to-end hops for the VMF-FGs shown in Fig. 15. The NFPC algorithm results in poorer performance, i.e, more end-to-end hops for all VMF-FGs and decomposition parameters due to the BFS nature of the algorithm while this phenomenon is attenuated in k-cutPC due to the cluster-level VMF PC. It is expected

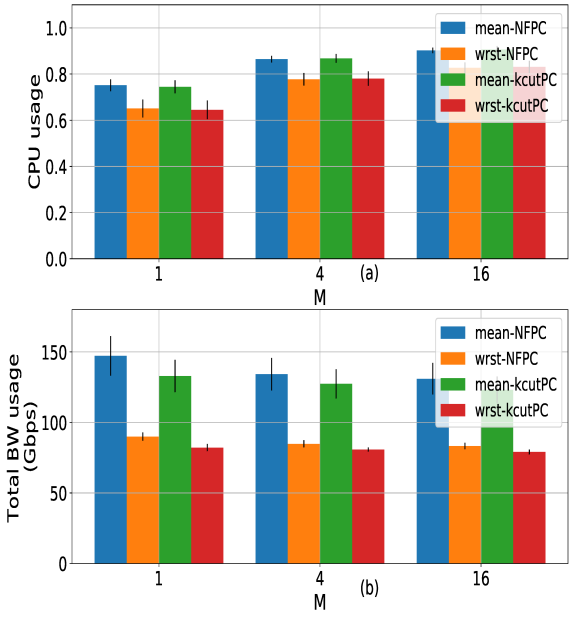

Fig. 18. Variation of resource utilization with M for 50 requests on a data center topology with 128 server-nodes ($\kappa = 8$). (a) Normalized CPU utilization and (b) Total inter-node bandwidth utilization.

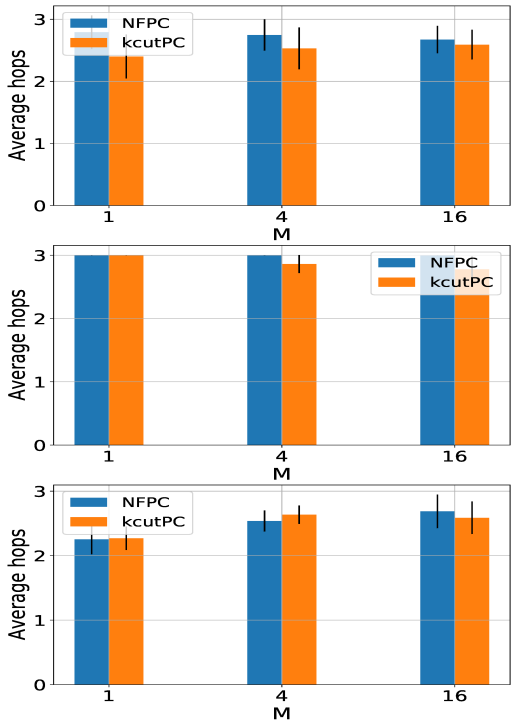

Fig. 19. Variation of end-to-end hops for (a) $\mathcal{G}_1$, (b) $\mathcal{G}_2$ and (c) $\mathcal{G}_3$ on a data center topology with 128 server-nodes ($k = 8$).

that end-to-end latency should reduce with increasing M as more and more neighboring VNF get placed on the same server node. However, for $\mathcal{G}_3$ the total end-to-end hops does not increase with M. This can be attributed to the decomposition of a specific VMF in the VMF-FG and also then its decomposition procedure. Here, it is due to the presence of the vqa VMF in the $\mathcal{G}_3$ whose decomposition results in increasing the end-to-end hops (Fig. 15).

VI. Conclusion

IP networking for media transport and general-purpose compute platforms for media production could help broadcasters to reduce the total expenditures along with several other benefits. The adoption of COTS platforms is expected to increase in the future. The success of the adoption largely depends on the efficiency of resource allocation algorithms. To improve the resource allocation efficiency, we have proposed a procedure for VMF-FG decomposition that can be used to transform a VMF-FG to an equivalent but lightweight VMF-FG. For media service deployment, we present two different VMF-PC algorithms– NFPC and k-cutPC aimed at improving the node and network resource usage, respectively. The evaluation compares the two VMF-PC algorithms in terms of four different metrics. The evaluation shows a significant improvement as a result of VMF-FG decomposition in terms of these metrics.

MFV requires the transportation of uncompressed media streams between VMFs in real-time. By scheduling packet transmissions for the media streams on a time-sensitive network, the timing metrics (end-to-end delay and jitter) can be bounded. The problem of VMF-FG deployment with scheduling will be addressed in our future research. Moreover, we plan to investigate the performance of media services implemented using open-source media processing frameworks. Later, the setup shall be used to implement the proposed VMF-FG decomposition procedure.

VII. Acknowledgment

This research was (partially) funded by the Flemish FWO SBO S003921N VERI-END.com (Verifiable and elastic end-to-end communication infrastructures for private professional environments) project.

References

[1] M. Fremeije, “The Rising Need for Media Function Virtualization,” RedHat, Tech. Rep., Feb. 2018.

[2] “The Road to COTS and the Cloud for real-time broadcast production,” Nevion, Tech. Rep., Jan. 2018.

[3] B. Research and Development. (2018). Compositing and Mixing Video in the Browser, [Online]. Available: https://www.bbc.co.uk/rd/blog/2017-07-compositing-mixing-video-browser (visited on 07/18/2017).

[4] *Agile media processing platform*, DS-PUB-2-0916D-EN, Grass Valley.

[5] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[6] "RP 2110-23:2019 - SMPTE Recommended Practice - Single Video Essence Transport over Multiple ST 2110-20 Streams," *RP 2110-23:2019*, pp. 1–27, 2020.

[7] G. P. Sharma, D. Colle, W. Tavernier, and M. Pickavet, "Improving resource utilization with virtual media function decomposition," in *2020 Fourth International Conference on Multimedia Computing, Networking and Applications (MCNA)*, IEEE, 2020, pp. 31–37.

[8] T. Fautier, "How ott services can match the quality of broadcast," *SMPTE Motion Imaging Journal*, vol. 129, no. 3, pp. 16–25, 2020. DOI: 10.5594/JMI.2020.2969763.

[9] Y. Reznik, J. Cenzano, and B. Zhang, "Transitioning broadcast to cloud," *Applied Sciences*, vol. 11, no. 2, p. 503, 2021.

[10] H. Koumaras, C. Sakkas, M. A. Kourtis, C. Xilouris, V. Koumaras, and G. Gardikis, "Enabling agile video transcoding over sdn/nfv-enabled networks," in *2016 International Conference on Telecommunications and Multimedia (TEMU)*, IEEE, 2016, pp. 1–5.

[11] T. Kojima, J. J. Stone, J. Chen, and P. N. Gardiner, "A Practical Approach to IP Live Production," in *SMPTE 2014 Annual Technical Conference Exhibition*, 2014, pp. 1–16.

[12] A. Kovalick, "Design elements for core ip media infrastructures," *SMPTE Motion Imaging Journal*, vol. 125, no. 2, pp. 16–23, 2016.

[13] *Grass Valley Technology Behind the world's largest SMPTE 2110 IP network at BBC cymru wales new central square HQ*, https://www.grassvalley.com/press-releases/2020/20201007-grass-valley-technology-behind-the-world-s-largest-smpte-2110-ip-network-at-bbc-cymru-wales-new-central-square-hq/, Accessed: 2021-05-22.

[14] "ST 2110-10:2017 - SMPTE Standard - Professional Media Over Managed IP Networks: System Timing and Definitions," *ST 2110-10:2017*, pp. 1–17, 2017.

[15] "ST 2110-20:2017 - SMPTE Standard - Professional Media Over Managed IP Networks: Uncompressed Active Video," *ST 2110-20:2017*, pp. 1–22, 2017.

[16] "ST 2110-30:2017 - SMPTE Standard - Professional Media Over Managed IP Networks: PCM Digital Audio," *ST 2110-30:2017*, pp. 1–9, 2017.

[17] "ST 2110-40:2018 - SMPTE Standard - Professional Media Over Managed IP Networks: SMPTE ST 291-1 Ancillary Data," *ST 2110-40:2018*, pp. 1–8, 2018.

[18] "ST 2022-6:2012 - SMPTE Standard - Transport of High Bit Rate Media Signals over IP Networks (HBRMT)," *ST 2022-6:2012*, pp. 1–16, 2012.

[19] "ST 2110-21:2017 - SMPTE Standard - Professional Media Over Managed IP Networks: Traffic Shaping and Delivery Timing for Video," *ST 2110-21:2017*, pp. 1–27, 2017.

[20] "ST 2110-31:2018 - SMPTE Standard - Professional Media Over Managed IP Networks: AES3 Transparent Transport," *ST 2110-31:2017*, 2018.

[21] D. Luzuriaga, C.-H. Lung, and M. Funmilayo, "Software-based video–audio production mixer via an ip network," *IEEE Access*, vol. 8, pp. 11 456–11 468, 2020.

[22] B. Research and Development. (2018). High Speed Networking: Open Sourcing our Kernel Bypass Work, [Online]. Available: https://www.bbc.co.uk/rd/blog/2018-04-high-speed-networking-open-source-kernel-bypass (visited on 05/03/2018).

[23] "Broadcast Video Infrastructure Implementation Using FPGAs," Altera, Tech. Rep., Mar. 2007.

[24] V. Bruns, T. Richter, B. Ahmed, J. Keinert, and S. Föel, "Decoding jpeg xs on a gpu," in *2018 Picture Coding Symposium (PCS)*, IEEE, 2018, pp. 111–115.

[25] B. Research and Development. (2014). The IP network behind the R&D Commonwealth Games 2014 Showcase, [Online]. Available: https://www.bbc.co.uk/rd/blog/2014-07-commonwealth-games-showcase-network (visited on 07/31/2014).

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[27] A. Gupta, E. Lee, and J. Li, "The karger-stein algorithm is optimal for k-cut," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, ser. STOC 2020, Chicago, IL, USA: Association for Computing Machinery, 2020, pp. 473–484, ISBN: 9781450369794. DOI: 10.1145/3357713.3384285. [Online]. Available: https://doi.org/10.1145/3357713.3384285.