FACULTY OF ENGINEERING AND ARCHITECTURE

Optimization Algorithms for the Deployment of Future Network and Media Services

Gourav Prateek Sharma

Doctoral dissertation submitted to obtain the academic degree of
Doctor of Computer Science Engineering

Supervisors
Prof. Wouter Tavernier, PhD - Prof. Didier Colle, PhD

Department of Information Technology
Faculty of Engineering and Architecture, Ghent University

June 2022

GHENT UNIVERSITY

ISBN 978-94-6355-602-6
NUR 986, 968
Wettelijk depot: D/2022/10.500/43

Members of the Examination Board

Chair

Prof. Patrick De Baets, PhD, Ghent University

Other members entitled to vote

Prof. Davide Careglio, PhD, Universitat Politècnica de Catalunya, Spain
Prof. Mario Pickavet, PhD, Ghent University
Prof. José Soler, PhD, Technical University of Denmark, Denmark
Prof. Dirk Stroobandt, PhD, Ghent University

Supervisors

Prof. Wouter Tavernier, PhD, Ghent University
Prof. Didier Colle, PhD, Ghent University

Acknowledgments

This journey would not have been possible without the support and help of many people who have contributed in their way.
I am first and foremost thankful to my promoters and supervisors, Prof. Wouter Tavernier, Prof. Didier Colle and Prof. Mario Pickavet for allowing me to do my PhD research under their guidance. Their immense knowledge, direction and timely feedback have been paramount to carry out research on various topics during my PhD.
Since the start of my PhD, I have been fortunate to share the office with the most kind and friendly colleagues– Thomas, Steven, Sander, Askhat, Pieter, Yuhui, Simon, Kenneth and Chong. I thank you all for the pleasant work environment you have helped to create.
I am thankful to the technical and administrative staff at IDLab: Martine, Davinia, Joeri and Sai to name a few. Their assistance has been crucial both to carry out my research as well as to complete administrative formalities during my stay in Gent.
This acknowledgment is incomplete without mentioning the friends who made my stay in Gent a memorable experience: Anil, Suhas, Pankaj and Ranjit. I am thankful to you for all the gossip, evening walks and parties. Playing badminton with my buddies: Vijay, Gopal, Manu, Kuber, Ashok and Arun, helped me stay motivated and high-spirited during this journey.
I am grateful to my cousin and cousin-in-law, Pritanshu *bhaiya* and Neha *bhabhi*, who have been extremely kind to me since I arrived in Belgium. Their company has never let me feel away from home.
Sam Harris from Waking Up deserves a special mention for his guided meditation lessons that have been invaluable, especially during the Corona pandemic.
I also want to express my gratitude towards my parents and my sister for being understanding of my decisions. Your compassion and love have been always with me. Last but not least, a special thanks goes to my fiance Annie. Without your unconditional support and encouragement over the past five years, this dissertation would not have been written.

Ghent, March 2022
Gourav Prateek Sharma

Table of Contents

List of Figures

List of Tables

List of Acronyms

A

ABR Adaptive Bit Rate
ACO Ant-Colony Optimization
AMPP Agile Media Processing Platform
API Application Programming Interface
ARPU Average Revenue Per User
AVB Audio Video Bridging

B

BBC British Broadcast Corporation
BRAM Block RAM

C

C-RAN Cloud RAN
CAPEX Capital Expenditure
CBQ Class-based Queuing
CBS Credit-based Shaper
CNC Centralized Network Controller
CO Control and Optimization
COTS Commercial Off-the-Shelf

CQF Cyclic Queuing and Forwarding
CRAN Centralized Radio Access Network
CSQF Cycle Specified Queueing and Forwarding

D

DAG Directed Acyclic Graph
DC Data center
DetNet Deterministic Networking
DiffServ Differentiated Service
DMA Direct Memory Access
DMPO Dynamic Multipath Optimization
DoD Department of Defense
DPP Dedicated path protection
DSCP Differentiated Services Code Point
DTH Direct-to-home

E

EMS Element Management System
EON Elastic Optical Network
ETSI European Telecommunications Standards Institute

F

FHD Full-High Definition
FIFO First-in, First-out
FPGA Field Programmable Gate Array
FRER Frame Replication and Elimination for Reliability

G

GCE Gate Control Entry
GCL Gate Control List
GPU Graphics Processing Unit

H

HDL Hardware Description Language
HVS High-Volume server

I

IDS Intrusion Detection System
ILP Integer Linear Program
IntServ Integrated Services
IP Internet Protocol
ISO International Organization for Standardization
ISP Internet Service Provider

K

KPI Key Performance Indicator

L

LAN Local Area Network
LUT Lookup table

M

MANO Management and Orchestration
MF Media Function
MFV Media Function Virtualization
MIMO Multiple Input Multiple Output
MIQCP Mixed Integer Quadratically Constrained Program
MISO Multiple Input Single Output

N

NAT Network Address Translation
NFPC Next-Fit Placement and Chaining
NF Network Function
NFV Network Function Virtualization
NFVO NFV Orchestrator
NIC Network Interface Card
NOS Network Operating System
NPU Network Processing Unit
NW-JSP No-wait Job-shop Scheduling Problem
NW-PSP No-wait Packet Scheduling Problem

O

ONAP Open Network Automation Platform
OPEX Operating Expenditure
OS Operating System
OSI Open Systems Interconnection
OSPF Open Shortest Path First

OTT Over-the-Top

P

PCE Path Computation Element
PCP Priority Code Point
PEF Packet Elimination Function
PHB Per-hop behavior
PIP Picture-in-Picture
PLC Programmable Logic Controller
PoC Proof-of-Concept
PRF Packet Replication Function
PROF Packet Re-ordering Function
PS Processing system
PTP Precision Time Protocol

Q

QoS Quality of Service

R

RoI Return on Investment
RSA Routing and Spectrum Allocation
RSVP Resource Reservation Protocol
RTP Real-time Transport Protocol
RTSCH Routing and Scheduling

S

SD Standard Definition
SDI Serial Digital Interface
SDN Software Defined Networking
SFC Service Function Chain
SLA Service-level Agreement
SLFL Simple Lazy Facility Location
SMPTE Society of Motion Picture and Television Engineers
SMT Satisfiability Modulo Theory
SPT Shortest Path Tree
SR Segment Routing
SRP Stream Reservation Protocol
SSM Service-Specific Manager

T

TAPRIO Time-aware Priority Shaper
TAS Time-aware Shaping
TDMA Time-division Multiple Access
TE Traffic Engineering
TG Task Group
TSN Time-Sensitive Networking
TT Time-Triggered
TTM Time-to-market

U

UDP User Datagram Protocol
UHD Ultra-High Definition

UNI User Network Interface

V

VIM Virtual Infrastructure Manager
VMF Virtual Media Function
VMF-FG VMF Forwarding Graph
VMF-PC VMF Placement and Chaining
VM Virtual Machine
VNE Virtual Network Embedding
VNFM Virtual Network Function
VNFD VNF Descriptor
VNFM VNF Manager
VNF Virtual Network Function
VoD Video on Demand
VPN Virtual Private Network

W

WAP Wireless Access Point
WFQ Weighted Fair Queuing
WG Working Group

Samenvatting

Mensen proberen al sinds mensenheugenis met elkaar te communiceren. In de afgelopen decennia heeft communicatie ongekende technologische ontwikkelingen gekend die hebben geleid tot het wereldwijd verbonden communicatienetwerk dat is uitgegroeid tot wat tegenwoordig het Internet wordt genoemd. Door het stijgende aantal verbonden gebruikers en de groeiende populariteit van toepassingen zoals videostreaming en conferencing, is de hoeveelheid verkeer die door telecomnetwerken stroomt exponentieel gegroeid. De groei van het verkeer in combinatie met de dalende Average Revenue Per User (ARPU) heeft telecomoperators gedwongen om kansen te creëren die kunnen helpen om extra inkomsten te genereren om dergelijke onheilspellende trends te compenseren.
De huidige Internet Service Providers (ISPs) bieden meerdere diensten aan hun klanten in aanvulling op alleen 'verbinden"hen met het Internet. Dit is een gevolg van het feit dat verschillende klanten (bijvoorbeeld een Bittorent, Industrie 4.0) verschillende eisen van het netwerk kunnen hebben, waardoor de netwerkoperator functies in het netwerk moet invoeren die complexe pakketverwerking kunnen uitvoeren naast het doorsturen van berichten. Deze functies, ook wel Netwerkfuncties (NFs) genoemd, kunnen aan elkaar worden gekoppeld om complexe netwerkdiensten te realiseren. Firewall, Network Address Translation (NAT) en media transcoders zijn een paar veel gebruikte NFs in telecom netwerken. NFs worden traditioneel geïmplementeerd via gespecialiseerde hardware apparaten ook bekend als middleboxes. Het creëren van netwerkdiensten via middleboxes is de afgelopen jaren een grote uitdaging geworden om de volgende redenen: (i) aanzienlijke uitgaven voor het verwerven en beheren van dure middleboxes, (ii) complexe implementatie-en operationele workflows en (iii) inflexibiliteit om te schalen met betrekking tot de vraag. Network Function Virtualization (NFV) heeft als doel de huidige architectuur van telecomnetwerken te transformeren naar een architectuur die gebaseerd is op Commercial Off-the-Shelf (COTS) hardware in plaats van middleboxen. NFV stelt voor om netwerkdiensten te bouwen via software NFs ook bekend als Virtual Network Functions (VNFs) gehost op een infrastructuur die bestaat uit standaard IT- reken -, opslag- en netwerkhardware. Het maken van netwerkservices vereist het routeren van verkeer via een reeks VNFs. Software Defined Networking (SDN) vult NFV aan door flexibele verkeersroutering tussen VNFs in de gewenste volgorde mogelijk te maken.

Ondanks alle voordelen die NFV biedt, is het essentieel dat de prestaties van de netwerkdienst niet verslechteren wanneer deze wordt gevirtualiseerd. Er wordt echter opgemerkt dat pakketverwerking in een software die draait op een CPU zeer uitdagend is wanneer de datasnelheden tientallen Gb/s overschrijden. bovendien vereisen bepaalde soorten VNFs CPU-intensieve bewerkingen (bijvoorbeeld cryptografie, mediacompressie) die resulteert in een hoog stroomverbruik in vergelijking met de overeenkomstige middleboxen.
Om de uitdagingen zoals verslechtering van de prestaties en een hoog stroomverbruik te overwinnen, wordt het gebruik van herconfigureerbare hardware voorgesteld. Hardwareversnellers worden verondersteld CPU-intensieve taken van VNFs te verwijderen, terwijl de rest van de VNF-operaties nog steeds op de CPU kunnen draaien. Als gevolg van hun opname in de NFV-Infrastructuur (NFVi) moeten naast andere nfvi-bronnen ook hardwareversnellers in aanmerking worden genomen bij het toewijzen van bronnen voor netwerkdiensten.
Dit brengt ons bij het eerste probleem dat in dit proefschrift wordt onderzocht. Het VNF Placement and Chaining (VNF-PC) probleem is aangepast aan het probleem van accelerator-aware VNF-PC (VNF-AAPC). Het VNF-AAPC probleem houdt rekening met de toewijzing van hardware accelerators aan VNFs bij het uitvoeren van VNF plaatsing en chaining. Om het VNF-AAPC probleem aan te pakken, wordt een exacte benadering voorgesteld gebaseerd op Integer lineaire programmering en een heuristische benadering. Simulatie-experimenten benadrukken de vereiste van de heuristische benadering om het schaalbaarheidsprobleem met de ILP-benadering aan te pakken. Ten koste van een kleine prestatie boete, kan de heurist grote gevallen van het VNF-AAPC probleem oplossen. Bovendien wordt de superioriteit van accelerator-aware benaderingen ten opzichte van acceleratoragnostische benaderingen, in termen van de totale kosten van knooppunten, waargenomen.
Analoog aan NFV, MFV is een architectuur waar media transport en verwerking wordt bereikt door het gebruik van COTS hardware in plaats van gespecialiseerde media hardware. Het transport van ongecomprimeerde media streams op IP-netwerken is aangetoond, maar de verwerking van dergelijke multi-gigabits/s streams in software is inefficiënt. De mogelijkheid om mediastromen met hoge bandbreedte op te splitsen in meerdere streams met lage bandbreedte en mediaverwerking te virtualiseren, biedt echter de mogelijkheid om virtuele mediadiensten te ontbinden. In dit proefschrift wordt een gegeneraliseerd VMF-FG-decompositiealgoritme gepresenteerd dat een functioneel equivalent maar een geïmplementeerde VMF-FG uitvoert. Daarnaast worden twee VMF-PC-algoritmen voorgesteld die zich bezighouden met de implementatie van een bepaalde VMF-FG. Een significante verbetering, zowel in termen van CPU-gebruik als netwerkbandbreedte, als gevolg van VMF-FG ontbinding wordt waargenomen in de simulatieresultaten.
Om de productie van MFV van uitzendkwaliteit te garanderen, is het es-

sentieel dat real-time communicatie wordt gerealiseerd tussen VMFs. Dit kan worden bereikt door latentie in netwerkknooppunten te beperken door speciale wachtrijmechanismen te gebruiken, zoals Cycle Specified Queuing and Forwarding (CSQF). Op basis van CSQF wordt een hebzuchtige heuristische gepresenteerd die pakketschema's genereert voor elke virtuele link in een bepaalde VMF-FG. Door de latentie tussen elke aangrenzende VMF van de VMF-FG te begrenzen, kan de end-to-end latentie van de implemented mediadienst worden gegarandeerd. De evaluatie van de VMF-FG scheduling heuristische heeft een vermindering van de end-to-end VMF-FG vertraging aangetoond, vooral voor hoogwaardige media formaten.
Naast on-premise studio productie, zijn omroepen op zoek naar uitbesteden van een aantal productie workflows naar de cloud. Door de strenge eisen in de omroepindustrie gaat cloudgebaseerde productie gepaard met verschillende uitdagingen, zoals tijdssynchronisatie van verschillende mediacomponenten (bronnen, spoelbakken, verwerkingsfuncties), voorspelbare verwerkingsvertragingen, enz. Toekomstig onderzoek is nodig om de bereidheid van de cloud om broadcast productie te ondersteunen te evalueren.
Wachtrijmechanismen zoals CSQF kunnen ervoor zorgen dat de communicatie tussen twee eindpunten wordt gegarandeerd met nul congestie naast tijdsgaranties. Pakketverliezen als gevolg van storingen (bijvoorbeeld knooppunt- of linkstoring) in het netwerk kunnen echter nog steeds leiden tot verstoring van de verkeersstroom. Dergelijke verstoringen kunnen nadelig zijn voor de uitzending van bijvoorbeeld een live concert of Sportevenement. Om storingen als gevolg van deze storingen te voorkomen, is speciale padbescherming voorgesteld. In emph1+1 dedicated protection dupliceert het bronknooppunt pakketten en stuurt ze langs twee disjuncte paden; de gedupliceerde pakketten worden geëlimineerd op het doelknooppunt. Door *1+1*-bescherming te combineren met CSQF-planning, kan nul pakketverlies naast begrensde latentie worden beloofd voor tijdgevoelige toepassingen. Hiertoe wordt het probleem van *1+1* routing en scheduling eerst geformuleerd en vervolgens opgelost met behulp van drie benaderingen: (I) ILP, (ii) Greedy en (iii) Tabu-search. De afweging tussen schaalbaarheid en gemiddeld geaccepteerd verkeer van de drie benaderingen wordt geëvalueerd via simulatie-experimenten.
Gezien de alomtegenwoordigheid van draadloze netwerken, is het essentieel om real-time toepassingen in het draadloze domein naast het bekabelde domein te ondersteunen. Onlangs zijn voorstellen gedaan om de nodige Time Sensitive Networking (TSN) kmechanismen op te nemen in het draadloze domein. In het bijzonder zijn time-synchronization en Time-aware Shaping (TAS) gedemonstreerd in bekabelde-draadloze gemengde netwerken. Het waarborgen van timing garanties in deze gemengde netwerken vereist het oplossen van het end-to-end pakket planning probleem. Het probleem wordt eerst geformuleerd als een ILP met een objectieve functie die de maximale flow-span tijd minimaliseert. Vervolgens wordt, om de schaalbaarheid van de ILP-aanpak aan te pakken, een hebzuchtige heuristische gepresenteerd.

Zoals bevestigd door de simulatie-experimenten, genereert de hebzuchtige heuristische pakketschema ' s voor realistische netwerktopologieën in een redelijke tijd, waarbij de ILP-benadering meestal onpraktisch is. Het is ook de moeite waard erop te wijzen dat de prestaties van de hebzuchtige heuristische blijkt te worden beïnvloed door de volgorde waarin real-time flow verzoeken worden gepland.
Concluderend, het eerste deel van het proefschrift gaat over problemen met de toewijzing van bronnen met betrekking tot de implementatie van gevirtualiseerde netwerk-en mediadiensten. De packet scheduling problemen in de context van real-time applicaties worden behandeld in het tweede deel van het proefschrift.

Summary

Human beings have been trying to communicate with each other since time immemorial. In the last few decades, communication has seen unprecedented technological advancements leading to the globally connected communication network which has become what is called today the Internet. Owing to the rising number of connected users and growing popularity of applications such as video streaming and conferencing, the amount of traffic flowing through telecom networks has been growing exponentially. Traffic growth accompanied by falling Average Revenue Per User (ARPU) has forced telecom operators to create opportunities that can help to generate additional revenues to compensate for such ominous trends.

Today's Internet Service Providers (ISPs) offer multiple services to their customers in addition to just "connecting" them to the Internet. This is a consequence of the fact that different customers (e.g., a Bittorent, Industry 4.0, etc) can have varying demands from the network, thus requiring the network operator to introduce functions in the network that can perform complex packet processing beyond message forwarding. These functions, also known as Network Functions (NFs), can be chained together to realize complex network services. Firewall, Network Address Translation (NAT) and media transcoders are a few widely used NFs in telecom networks. NFs are traditionally implemented via specialized hardware appliances also known as middleboxes. Creating network services through middleboxes has become very challenging in recent years due to the following reasons: (i) considerable expenditures to acquire and manage expensive middleboxes, (ii) complex deployment and operational workflows and (iii) inflexibility to scale with respect to demand. Network Function Virtualization (NFV) aims to transform the current architecture of telecom networks into the one that is based on Commercial Off-the-Shelf (COTS) hardware instead of middleboxes. NFV proposes to build network services via software NFs also known as Virtual Network Functions (VNFs) hosted on an infrastructure consisting of standard IT compute, storage and networking hardware. Network service creation requires routing traffic through a series of VNFs. Software Defined Networking (SDN) complements NFV by enabling flexible traffic routing between VNFs in the desired order.

Despite all the benefits NFV offers, it is essential that the performance of the network service does not deteriorate when it is virtualized. However, it is observed that packet processing in a software running on a CPU is

extremely challenging when the data rates exceed tens of Gb/s. Moreover, certain types of VNFs require CPU-intensive operations (e.g., cryptography, media compression) that results in high power consumption compared to the corresponding middleboxes.
To overcome the challenges such as performance deterioration and high power consumption, the use of reconfigurable hardware is proposed. Hardware accelerators are supposed to offload CPU-intensive tasks from VNFs, while the rest of the VNF operations can still run on the CPU. As a result of their inclusion in NFV Infrastructure (NFVi), hardware accelerators in addition to other NFVi resources should be taken into account when doing resource allocation for network services.
This brings us to the first problem investigated in this dissertation. The VNF Placement and Chaining (VNF-PC) problem is adapted to the problem of accelerator-aware VNF-PC (VNF-AAPC). The VNF-AAPC problem takes into account the allocation of hardware accelerators to VNFs when performing VNF placement and chaining. To address the VNF-AAPC problem, an exact approach based on Integer Linear Programming and a heuristic approach are proposed. Simulation experiments highlight the requirement of the heuristic approach to deal with the scalability issue with the ILP approach. At the cost of a small performance penalty, the heuristic can solve large instances of the VNF-AAPC problem. Moreover, the superiority of accelerator-aware approaches over accelerator-agnostic approaches, in terms of the total nodes cost, is observed.
Analogous to NFV, MFV is an architecture where media transport and processing is achieved through the use of COTS hardware instead of specialized media hardware. The transport of uncompressed media streams on IP networks has been demonstrated but the processing of such multi-gigabits/s streams in software is inefficient. However, the possibility to split high-bandwidth media streams into multiple low-bandwidth streams and virtualizing media processing opens up the opportunity to decompose virtual media services. A generalized VMF-FG decomposition algorithm is presented in this dissertation that outputs a functionally equivalent but an optimized VMF-FG. In addition, two VMF-PC algorithms are proposed that deal with the deployment of a given VMF-FG. A significant improvement, in terms of both CPU usage and network bandwidth, as a result of VMF-FG decomposition is observed in the simulation results.
To ensure broadcast-quality production of MFV, it is essential that real-time communication is realized between VMFs. This can be accomplished by bounding latency in network nodes by employing special queuing mechanisms such as Cycle Specified Queuing and Forwarding (CSQF). Based on CSQF, a greedy heuristic is presented that generates packet schedules for each virtual link in a given VMF-FG. By bounding the latency between each adjacent VMF of the VMF-FG, the end-to-end latency of the deployed media service can be guaranteed. The evaluation of the VMF-FG scheduling heuristic has shown a reduction in the end-to-end VMF-FG delay, especially

for high-quality media formats.
In addition to on-premise studio production, broadcasters are looking to outsource some production workflows to the cloud. Due to the stringent requirements in the broadcast industry, cloud-based production is accompanied by several challenges, e.g., time synchronization of various media components (sources, sinks, processing functions), predictable processing delays, etc. Future research is required to evaluate the readiness of the cloud to support broadcast production.
Queuing mechanisms such as CSQF can ensure the communication between two endpoints is guaranteed with zero congestion in addition to timing guarantees. However, packet losses due to malfunctioning (e.g., node or link failure) in the network can still result in traffic flow disruption. Such disruptions can be detrimental for broadcast production of, e.g., a live concert or sports event. To prevent disruptions due to these failures, dedicated path protection has been proposed. In *1+1* dedicated protection, the source node duplicates packets and routes them along two disjoint paths; the duplicated packets are eliminated at the destination node. By combining *1+1* protection with CSQF scheduling, zero packet loss in addition to bounded latency can be promised for time-sensitive applications. To this end, the problem of *1+1* routing and scheduling is first formulated and then solved using three approaches: (i) ILP, (ii) Greedy and (iii) Tabu-search. The trade-off between scalability and average accepted traffic of the three approaches is evaluated via simulation experiments.
Given the ubiquity of wireless networks, it is essential to support real-time applications in the wireless domain in addition to the wired domain. Recently, proposals have been made to incorporate the necessary Time-Sensitive Networking (TSN) mechanisms in the wireless domain. In particular, time-synchronization and Time-aware Shaping (TAS) have been demonstrated in wired-wireless mixed networks. Ensuring timing guarantees in these mixed networks necessitates solving the end-to-end packet scheduling problem. The problem is first formulated as an ILP with an objective function that minimizes the maximum flow-span time. Next, to address the scalability issue of the ILP approach, a greedy-based heuristic is presented. As confirmed by the simulation experiments, the greedy heuristic generates packet schedules for realistic network topologies in a reasonable time, where the ILP approach tends to be impractical. It is also worth pointing out that the performance of the greedy heuristic is found to be impacted by the order in which real-time flow requests are scheduled.
In conclusion, the first part of the dissertation deals with resource allocation problems concerning the deployment virtualized network and media services. The packet scheduling problems in the context of real-time applications are tackled in the second part of the dissertation.

1 Introduction

In recent years, there has been a remarkable growth in the amount of traffic flowing over computer networks. The global internet traffic grew threefold between 2016 and 2021, i.e., with a compound annual growth rate of 26% [1]. Due to the restrictions during the coronavirus pandemic, the Belgian internet traffic alone has doubled [2]. The growing traffic volume has forced the operators to frequently scale/upgrade their expensive network infrastructure to keep up the performance. The problem is further worsened by the declining Average Revenue Per User (ARPU). Fig. 1.1 shows that the ARPU in Europe has been steadily decreasing since 2012 [3]. The decrease from $30/month in 2012 to $20.40/month in 2017 is attributed to the highly competitive telecom market in Europe.
In the strive to remain relevant in the market, network operators and content providers offer various services beyond just connectivity. These services include firewalling, network caching, media transcoding, etc. Traditionally, network operators provide these services using specialized hardware appliances called middleboxes. The middlebox approach of delivering network services has several issues. Specialized hardware is expensive, has lengthy procurement time and is hard to deploy, manage and operate [4].

Traditional network architectures are now transformed to new architectures based on softwarized network processing hosted on Commercial off-the-shelf (COTS) hardware. These architectures are more adaptable and can potentially reduce the total expenditure. In order to truly exploit the

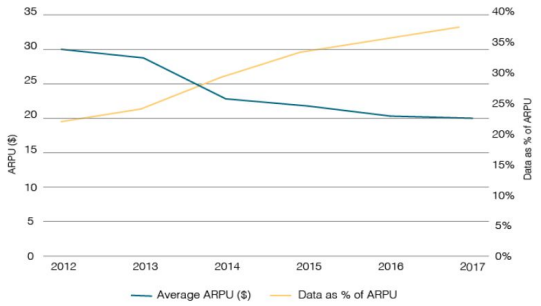


Figure 1.1: Average ARPU per month and data (% of ARPU) in Europe between 2012 and 2017 [3]

advantages of these architectures, efficient resource allocation is necessary. Furthermore, the network is envisaged to support applications with diverse requirements from the network. For instance, a BitTorrent client just needs a connection whereas the remote-control application for an industrial robot expects reliable communication with an end-to-end latency of a few milliseconds [5]. The same underlying network is expected to support both the standard traffic that requires no promises from the network and also the time-sensitive traffic that expects timing guarantees from the network. To this end, algorithms are required that schedule traffic in the network so that performance guarantees can be made for time-sensitive traffic. This dissertation deals with optimization algorithms concerning the deployment of virtualized network/media services and scheduling of time-sensitive traffic.

This chapter provides the necessary background on the various research themes covered in the dissertation. In Section 1.1, an overview of computer networks is provided. Next, in Section 1.2, the motivation behind software-based network processing and the Network Function Virtualization (NFV) architecture is discussed. The architectures for media-over-IP and Media Function Virtualization (MFV) are discussed in Section 1.4. In Section 1.5, various technologies that enable packet-switched networks to support real-time applications are described. The key research contributions of this dissertation are summarized in Section 1.6 and the list of publications is presented in Section 1.7.

1.1 Networking Overview

A computer network can simply be defined as a collection of nodes interconnected via links that exchange messages between the nodes. An endpoint in the network is a node where data originates from or terminates into. The endpoint where data originates is called the source and the endpoint where data terminates is called the destination. The most popular computer network in the 21st century is the Internet. Today's Internet is a global network interconnecting billions of endpoints across the world [1].
Data can be moved through a computer network using two ways: (i) circuit switching and (ii) packet switching [6]. In circuit switching, the resources that are needed to support communication, e.g., link bandwidth, between the endpoints are reserved for the duration of the communication session or connection. On the contrary, in packet switching, resources are not reserved but are shared on-demand by network messages resulting in *statistical multiplexing* [6]. Therefore, messages in a packet-switched network may have to wait or queue for access to the next link.
The most popular circuit-switched networks are telephone networks. Before any actual information is sent, a **circuit** has to be established between the two communicating endpoints as shown in Fig. 1.2 [6]. By doing so, the network reserves bandwidth on the required network links for the duration of the connection, thus ensuring that the endpoints can communicate at the required data rate without queuing in intermediate nodes. Due to the guaranteed rate of communication, circuit switching is also popular in professional media production, as explained in Section 1.4.

Most computer networks, including the Internet, are typically packet-

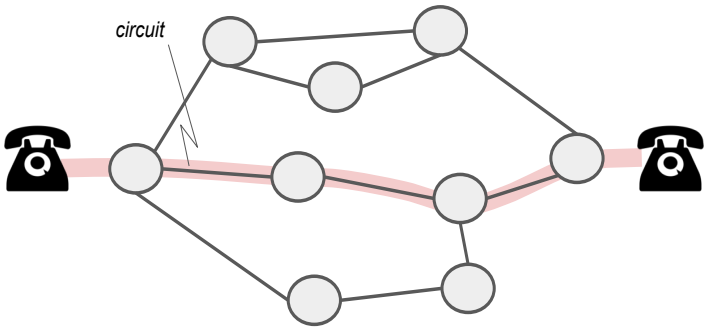


Figure 1.2: End-to-end connection between two endpoints in a circuit-switched network. Bandwidth at each link along the path needs to be reserved for the duration of the connection.

switched networks. As no resource reservation is done, packets have to wait in a node buffer if the following link is busy transmitting packets from other flows. Hence, the Internet does not make any guarantee when or if the packet will be delivered to the destination. This is known as *best-effort* service model.

Next, we look at how a packet traverses from the source to the destination in a packet-switched network.

Communication through a computer network requires that all nodes agree to a set of unambiguous rules that specify how different nodes can communicate with each other. A protocol is a set of rules that enable communication between any two nodes of the network. A standardized model called the Open Systems Interconnection (OSI) was proposed by the International Organization for Standardization (ISO) in the 1980s. The protocol-oriented version of the OSI model, proposed by the Department of Defense (DoD), is called the TCP/IP suite or the Internet protocol suite.

The TCP/IP suite describes various aspects of end-to-end communication, i.e., how the data should be packetized, addressed, routed and received. The suite groups these functionalities into five layers as shown in Fig. 1.3. The application layer or a process at the source produces the data that is intended to be transferred to the peer application or process on the destination. The transport layer receives the data produced by the application layer and attaches to it a header (containing port numbers) to allow process-to-process delivery of the data. The network layer (L3) receives the packet from the transport layer and routes datagrams across network boundaries using IP addresses. To this end, the network layer makes use of routing and addressing structures. The Internet Protocol (IP) is the ubiquitous network layer protocol and is considered the "Narrow Waist of the Internet" because almost all the Internet traffic flows through IP. By having a single network layer protocol (IP), a large number of applications can be supported over a large number of networks, i.e., IP act as an *inter-connector* of networks. In other words, Local Area Networks (LANs) based on different link layer technology can be connected as long as they speak IP. The fourth layer from the top in the TCP/IP stack is the data link layer. The data link layer (L2), e.g., Ethernet, is responsible for *switching* frames within the LAN. Here, the LAN refers to the network consisting of nodes accessible without crossing a router (explained later). The physical layer at the bottom contains all the functions required to carry the bitstream over a physical channel.

Although the network layer itself only provides connectionless service between any two endpoints, a connection-oriented service can be realized, e.g., by the transport layer [6]. For example, TCP provides a connection-oriented service to the application layer based on the connectionless IP. Before any

data is sent, a three-way "handshake" must be performed between the endpoints to mark the setup of a connection. User Datagram Protocol (UDP), in contrast, is connectionless because the endpoints can send data without requiring the handshake. In addition, TCP not only guarantees packet delivery (i.e., reliable delivery) but also ensures that packets will be delivered in the same order in which they were sent (i.e., in-order delivery). In addition, through flow control TCP ensures that receiver are not overwhelmed by fast senders.

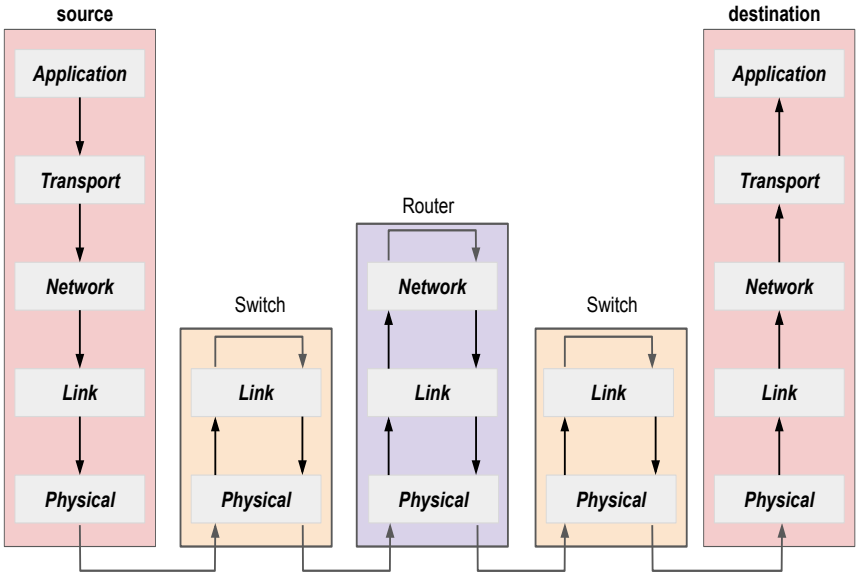


Figure 1.3: Data flow in a packet-switched network based on the TCP/IP model. Depending on the node (endpoints, switches and routers) functionality, packets pass through different protocol layers.

Every node in the network can have its *unique* address for every layer. A 32-bit IP address in the IP header is used to identify a host's interface in the network. At the data link layer, a Network Interface Card (NIC) of a host can be uniquely identified, e.g., using a 48-bit MAC address in the Ethernet header. Port numbers in the transport layer header (e.g., the 16-bit source or destination field in TCP) are used to identify the specific

process or service on the endpoint.
The fourth version of Internet Protocol, called IPv4, uses 32-bit addresses; thus has an address space of $2^{32} \approx 4.3B$ addresses. The IPv4 address space is running out of available unique addresses due to the proliferation of the large number of hosts connected to the Internet. A Network Address Translation (NAT) allows multiple computers in a local network to connect with the Internet while sharing one *public* IP address. By maintaining a mapping of local IP addresses and port numbers, a NAT makes all the outgoing traffic from a local network appear to be coming from one public address [6]. With NATs, the requirement to have a separate public IP address for each computer in local networks can be avoided. Consequently, NATs have slowed the speed with which the available IPv4 addresses are depleting.
NATs have tacked the address exhaustion problem of IPv4 to an extent. However, the exponential growth in the number of connected devices demands a different solution. Internet Protocol version 6 (IPv6) has been proposed as a successor to IPv4 to address the long-anticipated problem of IPv4 address exhaustion. The main difference between IPv4 and IPv6 is the address space. By using 128-bit addresses, IPv6 offers 2^{128} addresses, which is 1028 times more than IPv4. Though the adoption of IPv6 is increasing, it is widely expected that IPv4 still be used along with IPv6 for the foreseeable future. During the transitioning from IPv4 to IPv6, dual stacking, i.e., having both IPv4 and IPv6 on the same device, has been proposed.
Next, we describe how two hosts can communicate with each other over a packet-switched network using the TCP/IP protocol suite.
On the source side in the TCP/IP suite, each layer adds its own header to the data received from the upper layer, as illustrated in Fig. 1.4. This packing of data at each layer is known as encapsulation. When data traverses from the lowest layer to the highest layer at the destination side, each layer unpacks its header; this is called decapsulation. The consequence of the layering is that each layer provides a service to the layer above it while communicating with the corresponding layer on the other side. For example, in order to ensure correct *internetworking*, the IP layer relies on the data link layer for intra-network switching.

The transport of data using the TCP/IP suite requires datagram *forwarding* at the network layer and frame *switching* at the data link layer. A switch (L2 device), when receiving an Ethernet frame, looks up the destination MAC address (48-bit long field) of the frame in the lookup table to determine the port to which it should *switch* the frame [6]. In case no match is found, the frame is flooded to all ports. A router (L3 device), when receives a frame, it strips the data link headers to get the IP datagram. Then it looks for the destination IP address (32-bits long) of the datagram in its

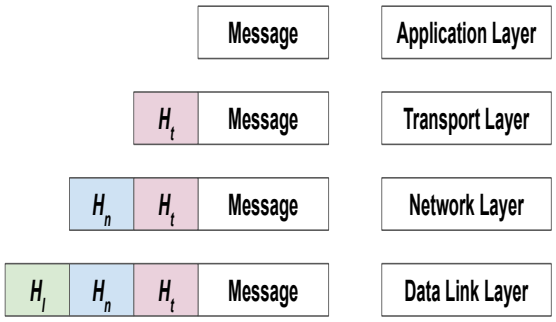


Figure 1.4: Encapsulation of application data as it passes through various layers of the TCP/IP model. H_t, H_n and H_l are the headers corresponding to the transport, network and data link, respectively.

routing table based on longest-prefix matching to switch the datagram to the required port [6]. The router then again appends the Ethernet header to generate a frame which is forwarded to the next node connected to the port. To summarize, the network nodes transport packets by simply looking at the data link and network layer headers.

In order to route a packet on a particular path, the networks nodes, i.e., routers and switches, should know which port the packet should be switched to. Routers may use, e.g., Open Shortest Path First (OSPF), to exchange link state information with their neighbouring routers in order to construct the topology of the network, based on which routing tables can be populated with appropriate entries. The processes (e.g., populating routing tables) responsible for controlling how packets are forwarded in the network are a part of the control plane. In contrast to the control plane, the actual forwarding in a node is defined by the data plane or forwarding plane of the router. The data plane needs to be fast and simple to ensure that packets can be switched at high-speed (tens of nanoseconds) whereas the control plane can relatively slower (few seconds). As discussed in section 1.3, a new architecture is being proposed where the two planes are separated that are usually coupled in traditional routers and switches.

The original TCP/IP based networks were based on a simple idea: delivering packets. This idea is highlighted in the end-to-end principle that states that the network should be relied on only for packet delivery and other functionalities such as error recovery, security, etc, should be placed at the

endpoints [7]. The network might implement a part of the functionality but only for the sake of performance enhancement. The initial growth and innovation are attributed to the end-to-end principle because it allowed to quickly add features to the endpoints and kept the Internet simple. However, the idea to keep the Internet as a dumb, fat, digital pipe has since been violated to support several features in the network.
In order to scale with the rapidly growing network traffic, operators introduce functionalities such as catching and load-balancing. Furthermore, in order to improve their ARPU, telecom operators offer addtional services beyond packet delivery. This is in line with the long list of features, e.g., minimum throughput, security, etc, commonly requested from the network [4]. Such functionalities are provisioned by carefully placing **Network Functions** (NFs) in the network. Other than packet forwarding, an NF can read and/or modify upper layers (above network layer) headers to implement the required functionality. NAT, Intrusion detection, address translation, media transcoding and load balancing are examples of a few NFs that can be found widely deployed in today's networks.
The aforementioned network processing functionalities are usually implemented using hardware nodes known as middleboxes [4]. In other words, a middlebox is an intermediary device that is capable of inspecting, transforming or manipulating network traffic en route to its destination. Some of the few examples of network middleboxes are as follows:

- **Firewalls** are used by network administrators to filter certain traffic based on protocol headers (e.g., network and/or transport), whereas malicious behavior can be detected by inspecting the contents of packets using **Intrusion Detection Systems** (IDS). LANs corresponding to the multiple sites of a corporate can be connected securely via the Internet using **Virtual Private Networks** (VPNs).
- **Network Address Translators** (NATs) are used to manipulate IP addresses in packets so that a single public IP address can be shared among multiple private IP addresses assigned to the endpoints behind the NAT.
- **Transcoders** allow encoding the media (audio or video) into different formats to make it compatible with a large variety of devices (e.g., laptops, mobiles and TVs). The high-quality media formats with huge bitrates are transcoded to low bitrate formats before transporting them over the Internet.

As middleboxes can empower networks to offer various functionalities, they have been widely deployed in enterprise as well as telecom networks. Typically, these functionalities are implemented using proprietary specialized

hardware. For example, a NAT box cannot be used for transcoding a video stream; in addition to that, it might not be possible to upgrade the transcoder middlebox to en/decode the latest format of video streaming. An elaborate discussion on the challenges faced by the network operators concerning middleboxes is presented in the next section.

1.2 Network Function Virtualization

Traditionally, the deployment of middleboxes is done in an ad-hoc basis, i.e., depending on the requirement of new functionality, a middlebox that implements the functionality is purchased and then deployed at the appropriate location in the network. As estimated by J. Sherry et al., about one-third of the total boxes in enterprise network consist of middleboxes which themselves comprise of various types (e.g., IDS, load-balancers, firewalls, etc) [4]. The traditional architecture of network service implementation using middleboxes poses the following challenges:

1. **High Expenditure**: The Capital Expenditure (OPEX) and Operating Expenditure (OPEX) associated with middleboxes are huge. The upfront investment to buy hardware equipment can go up millions of dollars. Further, huge trained manpower is required to manage such specialized hardware deployed across the network. These costs are further exacerbated by the requirement to periodically upgrade and/or replace the hardware to keep up with the increasing traffic volume.

2. **Deployment and Operational issues**: Installation, monitoring, diagnostics and configuration of middleboxes deployed across the network is quite challenging due to the unique expertise required for each type of middlebox. The operational tasks are made more difficult due to the proprietary Application Programming Interfaces (API) exposed by middleboxes supplied from different vendors.

3. **Inflexibility**: Network operators have to over-provision middleboxes keeping in view the peak demand of the day (not the average demand). Additional resources have to be kept as a backup in case some middleboxes fail. Furthermore, the innovation in creating network services is impeded due to the proprietary and blackbox nature of middleboxes.

The above-mentioned challenges need to be addressed in order to make the telecom business viable. This has pressured telecom operators to seek network architectures that can help lower the CAPEX and OPEX while being able to meet the customer requirements. By bringing software and

standard IT virtualization technologies into the network, these challenges are expected to be addressed.

1.2.1 Softwarized Network Functions

A middlebox is a hardware implementation of an NF; however, the same NF can be implemented in software and deployed on general-purpose hardware. Such an architecture where network processing is done using software NFs running over Commercial Off-the-Shelf (COTS) hardware instead of middleboxes is known as Network Function Virtualization (NFV). By avoiding vendor lock-in, NFV can potentially reduce huge expenditures [8]. The software implementation of an NF is known as Virtual Network Function (VNF). Next, we discuss an example use case for NFV: virtualization of mobile packet core.

A mobile network typically has two parts (i) the Radio access Network (RAN) connecting the base station and the User Equipment (UE) and (ii) the packet core that connects the RAN with external networks. A basic architecture of a mobile network with the component NFs is shown in Fig. 1.5. The core network is composed of the Service Gateway (SGW), the Packet Data Network (PDN) Gateway (PGW) and the Mobility Management Entity (MME) and the Home Subscriber Server (HSS). The MME and the HSS form the control plane whereas the SGW and the PGW form the user plane. The gateways: SGW and PGW are responsible for transporting the user IP traffic between the UE and the Internet. The SGW routes the traffic coming from the UE to the packet core and vice-versa. It serves as an anchor while the UE moves between different base stations. The PGW deals with the routing of traffic coming from and to external networks along with other operations such as IP address allocation and packet filtering.

The NFs in the packet core are seen as potential candidates for virtualization in future architectures. With the NFs in packet core virtualized and hosted on a standard NFVi, telecom operators can expect to significantly reduce their CAPEX and OPEX along with other NFV benefits [14].

Most often, the physical resources (i.e., servers, storage and network) in the NFVi are virtualized due to reasons such as efficient resource utilization, easy management and faster provisioning. Virtual Machines (VMs) and Docker containers are two standard virtualization technologies of today. A Virtual Network Function (VNF) consists of one or more such VMs or containers. Next, we give an introduction to these two virtualization technologies.

A VM is an emulation or virtual representation on a physical computer, i.e., instead of using hardware directly to run programs, a VM uses the software abstraction of hardware. The physical computer is referred to as

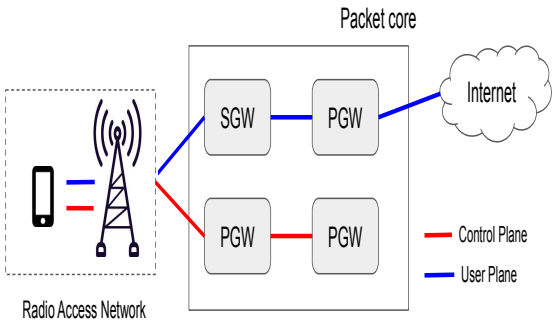


Figure 1.5: Overview of the mobile architecture [15].

the host whereas the VM is referred to as the guest. Multiple guests, each running its own Operating System (OS), can run on the same host, e.g., a virtual MacOS and UbuntuOS can run on the same host. A software called Hypervisor is responsible for allocating resources, i.e., CPU, memory and storage, to the guests. There are two main types of hypervisors: Type 1 hypervisors (e.g., Xen, Microsoft Hyper V) run directly above the host hardware thus avoiding the additional OS layer, while Type 2 hypervisors (e.g., Virtualbox, VMware Workstation) require an OS layer like a typical computer program. Each guest VM runs a full-fledged OS to support the application running in it.

In contrast to a VM, a container running on a host shares the host's OS, as

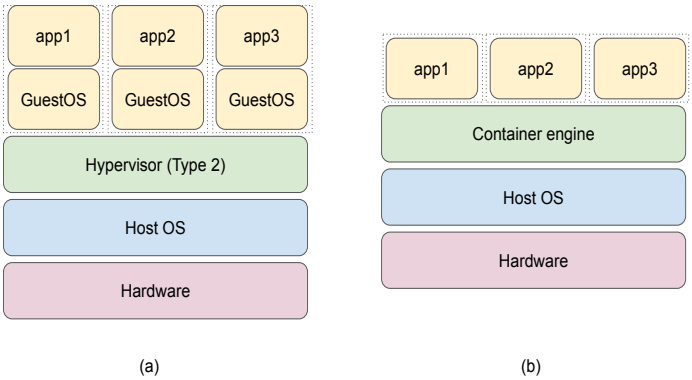


Figure 1.6: Comparison of (a) VM-based and (b) container-based virtualization.

illustrated in Fig. 1.6. Using a container engine an OS can be abstracted to run multiple containers similar to a hypervisor that virtualizes the underlying hardware. This makes VNFs based on containers very lightweight and less resource-intensive as compared to VM-based VMFs. However, multiple container VNFs sharing an OS are weakly isolated as compared to VM-based VNFs; this poses a security risk when VNFs from multiple tenants run on the same machine [9]. Both these virtualization technologies have their advantages and disadvantages and can be used in NFV on their own or in combination.

1.2.2 ETSI NFV architecture

NFV leverages the previously discussed virtualization technologies to host telecom services. The European Telecommunications Standards Institute (ETSI) has released a number of standards around specifications of various components and interfaces in the NFV architecture. The ETSI NFV architectural framework is shown in Fig. 1.7 [10]. The main components of this architecture relevant to this dissertation are (i) VNFs, (ii) NFV Infrastructure (NFVi) and NFV Management and Orchestration (MANO).

The NFVi layer consists of all the hardware and software components re-

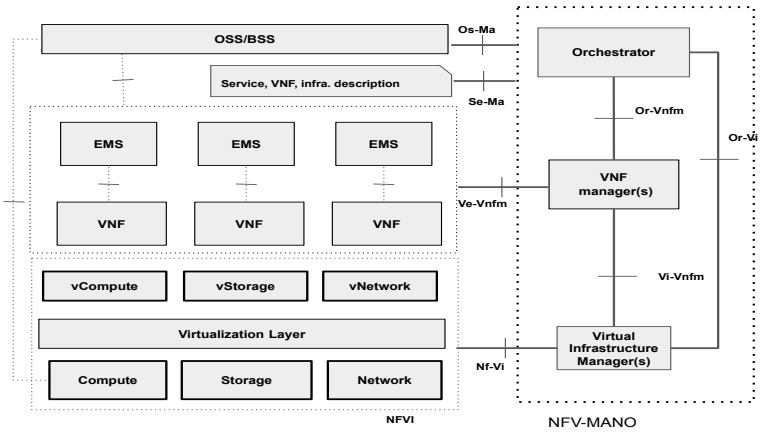


Figure 1.7: Reference NFV architecture by ETSI [10].

quired to host afore-mentioned VNFs. The physical infrastructure consists of COTS resources for compute, network, storage, etc. The physical hardware is usually abstracted via a virtualization layer (Hypervisor or container

engine) so that resources can be logically partitioned and allocated to VNFs. The deployment and re-assignment of VNFs on NFVi is easier and faster as compared to middleboxes. A complex network service can be implemented by chaining multiple VNFs together in a specific configuration. VNFs can be deployed, scaled and configured on-demand thus making the rolling out of upgraded or new network services rather quickly. The faster roll-out of services entails automating the orchestration of NFVi resources for network services. Other service management tasks such as monitoring, configuration and licensing are also required to ensure scalability and performance requirements [11]. In an NFV environment, the NFV MANO block is responsible for various tasks relevant to the management and orchestration of virtualized network services. These tasks are carried out using three subcomponents of MANO: (i) **Virtual Infrastructure Manager** (VIM) , (ii) **VNF Manager** (VNFM) (iii) **NFV Orchestrator** (NFVO).
The VIM controls and manages the compute, network and storage resources in NFVi. The VNF manager is responsible for the tasks concerning the lifecycle management of VNFs. This includes installation, scaling, updating and termination of VNFs. The NFVO is in charge of managing the lifecycle of network services and also orchestrating NFVi resources for network services. The NFVO coordinates with the VNFM to instantiate VNFs and manages the deployment of network services. For resource orchestration, it engages with the VIM to ensure optimized allocation of resources is achieved. OSM MANO and ONAP are two main open source Management and Orchestration (MANO) software stacks compliant with ETSI NFV architecture [12], [13].

1.2.3 Hardware-accelerated VNFs

Transitioning from middlebox-based network architecture to the one based on VNFs can potentially result in cost-savings, ease of management and upgrade, faster service roll-outs, etc. Despite these benefits, a major obstacle before the widespread adoption of NFV by network operators is performance degradation. With NFV, VNF-chains are expected to process network traffic exceeding tens of Gbps. Meeting the performance of middleboxes with VNFs running on COTS hardware is extremely challenging [24]. For example, at 10Gbps the time available to process a 64B packet is less than 52ns while the memory latency is in tens of ns. Moreover, VNFs are very inefficient as compared to middleboxes in terms of power consumption when processing the same amount of traffic [25].
The NFV performance challenge has compelled researchers to look at hardware acceleration techniques for VNFs. The idea of VNF hardware acceleration is to offload CPU-intensive tasks from a VNF to an externally attached

hardware, e.g., a Field Programmable Gate Array (FPGA), while the rest of the VNF operations can run on the CPU. An FPGA is a re-configurable integrated circuit that consists of an array of logic blocks that can be modified to implement a desired function. In contrast to the sequential execution of operations in a CPU, multiple operations can be parallelized on an FPGA. Massive parallelism combined with distributed on-chip memory renders FPGAs well-suited for packet processing as compared to CPUs. Therefore, hardware acceleration, e.g., using an FPGA, has the potential to improve packet-processing performance and at the same time to reduce the total energy spending due to better efficiency of hardware accelerators as compared to software NFs [26].
Keeping the benefits of hardware acceleration in mind, the NFVi layer of the NFV architecture (Fig. 1.7) can include accelerator resources along with compute, network and storage. Furthermore, hardware accelerator resources can be virtualized and allocated to the VNFs similar to compute resource [27]. In order to retain the flexibility of NFV, the use of reconfigurable hardware accelerators (e.g., FPGAs, GPUs) in conjunction with COTS hardware has been proposed [24], [25]. The idea is that compute-intensive workload from a VNF can be offloaded to an attached accelerator while the rest of the VNF functionality can run on the CPU. For example, by offloading en/decryption (e.g., Advanced Encryption Standard (AES)) and hashing (e.g., Secure Hash Algorithm (SHA)) operations from an IPSec VNF to an FPGA-based accelerator, the CPU usage can be halved [28].

In summary, the NFV performance and power consumption challenge can be addressed by incorporating hardware accelerators in NFVi. Incorporating hardware accelerators in NFVi is not enough; the resource allocation process in the NFVO should be made aware of hardware accelerators so that efficient resource utilization is achieved. This challenge is highlighted in Section 1.6 of this chapter.

1.3 Software Defined Networking

A similar but complementary transformation in the telecom domain is **Software Defined Networking** (SDN). SDN proposes to decouple the network's control plane from the data plane that reside together in routers and switches [16]. With such task separation, switches become simple forwarding nodes and the control plane is consolidated at a logically centralized controller. A few examples of open-source SDN controller are ONOS, RYU, etc [17], [18]. Fig. 1.8 shows a simplified SDN architecture. The control plane interacts with the decentralized data plane using some protocol. OpenFlow is an example of such a protocol that enables a controller to install

desired flow rules in the tables of OpenFlow switches [19]. The data plane can consist of both OpenFlow (hardware) switches and software switches, e.g., open-source projects such as Click, Open Virtual Switch (OvS), Cumulus [20], [21], [22]. These software SDN switches are widely employed to realize complex VNF chains in NFV environments. In the management plane, desired network policies can be defined and then communicated to the control plane (e.g., using REST API), which is responsible to enforce them via configuring the data plane accordingly.

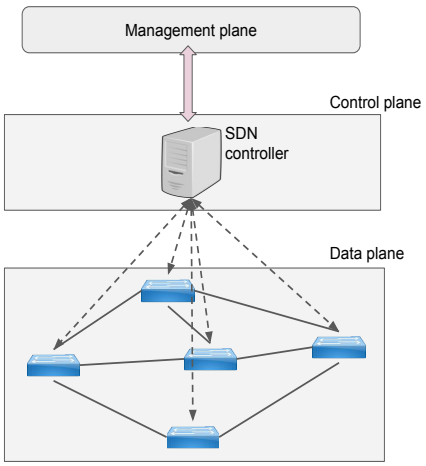


Figure 1.8: Overview of the SDN architecture [19].

To summarize, NFV can exploit network programmability resulting from SDN to implement network services efficiently and enhance their performance. For example using OpenFlow, the traffic flow between the adjacent VNFs of a VNF-chain can be optimized [23].

1.4 Media-over-IP and Media Function Virtualization

Since the transition of the broadcast industry from analog to digital infrastructure, Serial Digital Interface (SDI) has been a sole transportation method for carrying media signals (audio, video and auxiliary) across the studio network [29]. Every media source and sink is connected point-to-point to the back of the SDI router with BNC connectors and the switching

matrix in the router performs circuit switching between the input and output ports. The cable connections from sources/sinks to the SDI router are unidirectional.
The wide adoption of SDI in the broadcast industry can be attributed to its outstanding performance. In particular, the consistent latency offered by the switching matrix regardless of the number of high-bandwidth media streams being switched is one of the important features of SDI. Compatibility, robustness and reliability are some additional advantages offered by SDI that are indispensable in the broadcast industry.
Despite the outstanding performance, SDI-based transport faces the following challenges:

1. **High-Cost**: The CAPEX for SDI infrastructure is huge due to the bespoke hardware, which has a small market size as compared to standard IT appliances.

2. **Excessive cabling**: The unidirectional communication and the lack of multiplexing results in many cables running in parallel between SDI routers. As a result, the installation, maintenance and upgrade requires large manual labour.

3. **Upgrade costs**: The switching matrix in an SDI router is not agnostic to the format of media essence being switched. Therefore, upgrading to higher quality media formats (e.g., from HD to UHD) requires replacing the old SDI routers with the ones compatible with the new format.

1.4.1 IP Media

Back in the 1990s, the Standard Definition (SD) video was carried using SDI because IP networks could not support such multi-gigabits data rates. However, around early 2000s IP data rates surpassed that of SDI's. Despite the fact that the largest share ($> 80\%$) of the total internet traffic is from media streaming applications [1], the transport of uncompressed media streams over IP inside broadcast facility has garnered attention from broadcasters only lately. The recent attention towards IP was mainly due to the potential of reducing costs in a long run by building studio infrastructure based on standard IT equipment as opposed to proprietary SDI hardware [30]. The cost of IP network hardware is much less as it is less specialized than proprietary broadcasting hardware, thus kicking off the economies of scale effect. Media transport over IP would mean any media, irrespective of the type (e.g., audio, video or ancillary) or quality (e.g., HD or UHD) can be switched using COTS switches. This means that the Return on Investment

(RoI) can be sustained over many years in future when upgrading to 4K, 8K or other emerging high-quality formats.
IP also allows multiplexing, i.e., multiple media streams can share the same cable thus resulting in fewer cables implying easier installation, upgrade and management. Owing to the format-agnostic nature of IP, upgrading to high-resolution formats (e.g., from HD to UHD) does not necessitate upgrading the IP infrastructure, as long as sufficient network bandwidth is available. In an IP-based studio, a multi-layer switching architecture (e.g., leaf-spine network) is used to interconnect multiple endpoints (e.g., cameras, multi-viewer screens, etc) and media processing elements (e.g., vision mixers) located across the studio facility as shown in Fig. 1.9 [31]. Leaf-spine architectures have been widely deployed in modern data centers due to their simplicity, resilience and scalability. Further, there are at most three hops between the endpoints thus reducing the end-to-end latency. An SDN controller can be used to manage the switching core so that media streams can be transported optimally between the endpoints [32].

The shift towards IP studios was clearly highlighted when the Society

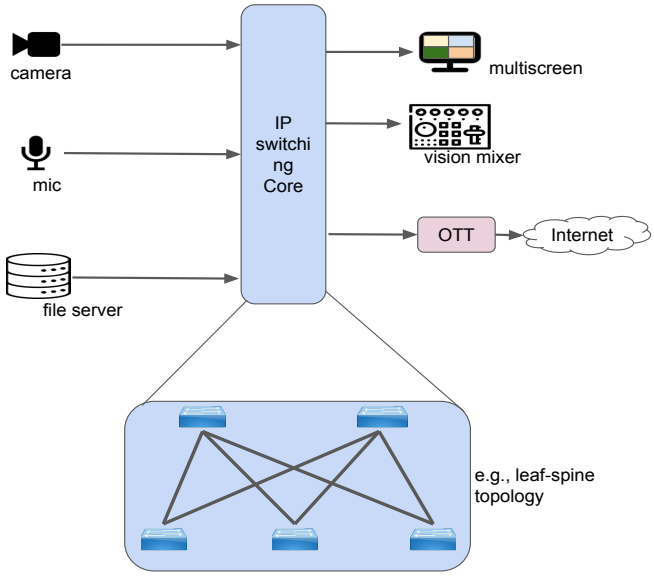


Figure 1.9: Architecture of an IP-based broadcast studio. The switching core can be built as, e.g., leaf-spine topology [31].

of Motion Picture and Television Engineers (SMPTE) released SMPTE ST2110, a suite of standards that describes how broadcast quality media can be transported or distributed over IP [33], [34], [35]. The idea is to

transport each essence (i.e., audio, video and ancillary data) as a separate stream as shown in Fig. 1.10 (b). This is in contrast to older standards, such as SMPTE ST 2022-6, that prescribe bundle-based media transport (Fig. 1.10 (a)) [36]. The independent transport of each essence results in much more efficient communication than with the bundle approach. For instance, an audio processor would require extra processing to unpack audio data from a bundled (audio, video and ANC) stream whereas the essence-based approach would not result in such overhead.

For decades, the Real-time Protocol (RTP) has been a proven applica-

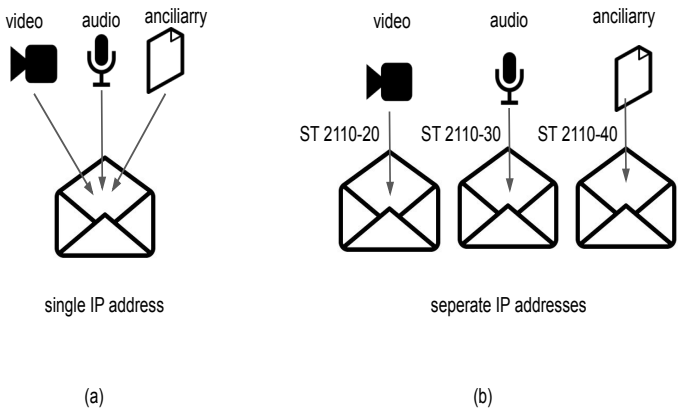


Figure 1.10: Comparison of (a) bundle-based (SMPTE ST 2022-6) and (b) essence-based media (SMPTE ST 2110) transport.

tion layer protocol for time-critical communications over IP networks, e.g., streaming media, teleconferencing, etc. RTP runs over UDP, which is faster and simpler than TCP as it does not have overheads like error correction, sequencing, flow and congestion, etc. The uncompressed media samples (video or audio frames) are grouped into datagrams [37]. As there is no sequencing in UDP, a 32-bit sequence number, i.e., 16 bits from RTP header and 16 bits from RTP payload header, is used. Further, the timestamp field in the RTP header contains the capture time of the frame to which the packet belongs. All RTP packets belonging to a frame contain the same timestamp value, which is derived from the capture device's clock synchronized with the Precise Time Protocol (PTP).

Using an example, we next explain the process of packetization of an (uncompressed) video frame. Assume a video frame with 4:2:2 subsampling,

i.e., the two chroma components (Cb and Cr) are sampled at half the horizontal sample rate of the luma component (Y'). This implies two horizontally adjacent pixels form a pixel group (pgroup) of five octets if each component is encoded using 10 bits. Therefore, a 4K video frame (3840x2160) has 4147200 pgroups resulting in a frame size of 20.736 MB. Assuming a UDP datagram of size 1460B, it would require about 14k such datagrams to transport one video frame. At 30frames/second, the corresponding video stream would require 4.976Gbps of bandwidth.
Fig. 1.11 shows the encapsulation of video pixel groups (pgroups) resulting from different network layers. To summarize, multiple pgroups are appended with an RTP payload header (PH) which is inserted in an RTP packet which is placed into a UDP packet which is attached with an IP header which is finally wrapped inside an Ethernet frame. The resulting Ethernet frame is then sent to the destination.
The network transports the packets to the destination where the pixels are extracted from each received packet and the full media frame is re-assembled using RTP sequence numbers.

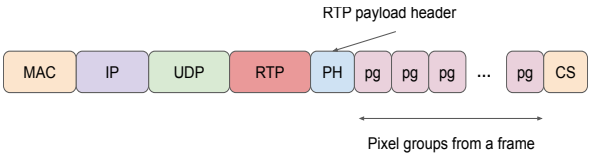


Figure 1.11: Encapsulation of video pixel groups as a result of different protocol layers.

1.4.2 Media Function Virtualization

Broadcasters need to perform various operations on uncompressed media streams to produce the desired content. Media processing is traditionally implemented via specialized hardware appliances. Analogous to an NF in NFV, a **Media Function** (MF) takes as input one or more (SMPTE 2110) media streams and performs some frame-level operations on it to produce the output stream. Fig. 1.12 shows an example of a vision mixer VMF that takes as input two video streams and the output stream switches between the two inputs. In addition, a wipe transition effect (from left to right) is added when output stream o is switched from i_0 to i_1. Producing broadcast-quality content requires many more types of MFs such as color correction, Picture-in-Picture (PIP), split-screen, etc. Complex media services can be realized by chaining MFs with each other in specific configurations. Due to

the stringent performance requirements in broadcast production, hardware-based MFs have been relied upon.

The possibility of transporting media using IP paves the way for a more

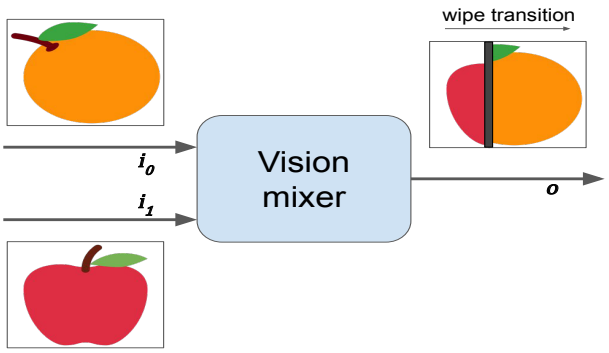


Figure 1.12: Illustration of a vision mixer functionality; the video stream on output port o is switched from i_0 to i_1 with a wipe transition.

scalable and future-proof architecture where media processing can be softwarized. Along these lines, the broadcast industry is in a phase of replacing specialized media processing hardware with infrastructure consisting of COTS hardware running media processing software for potential advantages such as cost-savings, flexibility and agility. The software implementation of an MF is referred to as the **Virtual Media Function** (VMF). Often, virtual hardware instances are created from physical hardware using virtualization technologies such as VMs or Docker containers.

Recently, there have been several attempts to produce broadcast-quality content using VMFs. The BBC partnering with Isotama developed a video mixing VMF that can be controlled through a browser application [38]. Moreover, Grassvalley's has released an Agile Media Processing Platform (AMPP) that leverages elastic compute of the COTS infrastructure to run a variety of media processing workflows on one platform [39].

We propose **Media Function Virtualization** (MFV) as an architecture where media services are implemented using VMFs hosted on COTS hardware, similar to the NFV architecture where network services are realized via VNF [40]. Fig. 1.13 illustrates the 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). A media service can be realized by chaining multiple VMFs in a particular configuration. This VMF configuration is represented using a directed graph referred to as VMF Forwarding Graph (VMF-FG). 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.

The idea behind MFV can be extended further by executing (part) of

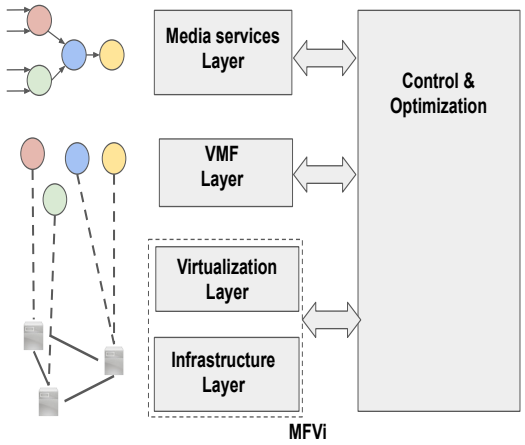


Figure 1.13: Illustration of various layers and components in the MFV architecture [40].

production workflows in the cloud as opposed to running them in the local facility. The "pay-as-go" model of the cloud can be attractive for small to medium broadcasters that might want to avoid high CAPEX in deploying broadcasting hardware; the cloud could also help to quickly scale resources for media productions based on the demand. However, there is some skepticism around cloud-based broadcast production due to performance and security considerations; therefore, currently, cloud resources have been used for non-live production workflows [41].

1.5 Deterministic Networking

Normally, routers and Ethernet switches in packet-switched networks operate on a best-effort basis. Best-effort service entails that a packet can get delayed excessively or in the worst-case gets dropped somewhere in the network. In other words, there is no guarantee that the packet will reach its destination within a bounded time or will even reach at all. The motivation behind this was to keep the networking infrastructure fast and simple thus making it possible for the network to route millions of flows. Although a large percentage of traffic in the networks is forwarded with the Best-effort service model, it is not adequate for various applications that require some assurances from the network, e.g., in terms of throughput, latency, jitter, etc. For instance, it might be okay for a google query result to be delayed by quarter of a second, though a jitter as low as 10ms can severely impact the performance of a remotely-controlled robot [42], [5]. The applications that require preferential treatment over the rest use differentiated Quality of Service (QoS); QoS parameters include minimum throughput, maximum tolerable delay or jitter, etc.

As networks are expected to support applications with diverse requirements, some applications require QoS guarantees from the network. Key Performance Parameters (KPIs) of certain applications such as packet loss, delay and jitter can be managed by prioritizing their network flows over the rest. A customer can enter into a contractual agreement called Service-level Agreement (SLA) with the network operator that guarantees a certain service assurance (e.g., throughput or latency) at an additional expense.

The connections in a circuit-switched network are accompanied by the reservation of resources so there is QoS baked into these networks. As the Internet and most of the LANs are based on packet switching, we focus on the relevant QoS approaches.

1.5.1 QoS in Packet-switched Networks

While the behavior of packet-switched networks is inherently unpredictable, a number of efforts have been made to support QoS. The two main traditional QoS models followed in the modern IP networks are as follows [6]:

1. **IntServ**: The Integrated Services (IntServ) model is based on a reservation-based approach where network resources are explicitly reserved for certain network flows. In order to request and reserve resources in the network, applications use an end-to-end signaling mechanism called Resource Reservation Protocol (RSVP). Routers must implement packet classifying and scheduling mechanisms, e.g., Class-Based Queuing (CBQ) and Weighted Fair Queuing (WFQ), to support QoS in IP networks.

2. **DiffServ**: The Differentiated Services (DiffServ) model is based on a reservation-less approach where the network traffic is differentiated into multiple classes which then get prioritized treatment in routers. IntServ gives per-flow guarantees whereas DiffServ operates at the class level. At the network boundary, the six-bit Differentiated Services Code Point (DSCP) field is marked with a value that determines how the packet will be treated as it traverses through the network [43]. There is no RSVP-like signaling mechanism thus no per-flow state is maintained in routers. DiffServ requires routers to implement per-hop behaviors (PHBs) that define how a packet belonging to a particular class needs to be treated. It is difficult to promise end-to-end guarantees with DiffServ as each node may implement different scheduling and queuing mechanisms.

IntServ and DiffServ do support some QoS but they are suitable for applications associated with KPIs such as average delay or packet loss. The IntServ model due to its per-flow management is not scalable for large networks whereas promising end-to-end guarantees is difficult with the DiffServ service model [42]. For real-time applications (e.g., a remotely-controlled robot) to work correctly, it is essential that the network can provide a deterministic behavior in terms of strict KPIs, e.g., maximum latency and jitter. The end-to-end latency on a network path consists of three components (i) the propagation delay, (ii) transmission delay and (iii) queuing delay. The first two components are fixed once the path is determined whereas the queuing delay is variable. Therefore, in order to guarantee the end-to-end latency, it is important to bound queuing delay in each node. This can be achieved by employing various traffic shaping mechanisms, as discussed next.

The efforts to bring time-sensitivity in Ethernet can be traced to the work done in the context of Audio Video Bridging (AVB) standards developed by the IEEE [44]. The set of AVB standards describes how audio or video streams can be transported over Ethernet along with QoS in terms of bounded latency and jitter. Due to the increased scope of the work, the Time Sensitive Networking (TSN) Task Group (TG) was formed in 2012 by renaming the AVB task group. The objective of the TSN TG is to enable real-time communication (i.e., bounded latency and availability guarantees) in Ethernet, which is a prerequisite for real-time applications such as industrial automation and broadcast production. There has been a growing interest in bringing determinism over the (Layer 3) routed segments. To this end, the IETF created Deterministic Networking (DetNet) TG that aims to extend determinism to the network layer, as well.
Next, we give an overview of the progress made within TSN and DetNet TG that is relevant to our work.

1.5.1.1 Time Sensitive Networking

The TSN TG has proposed a set of standards that specify techniques and mechanisms required to support real-time communication in Ethernet as listed in Tab. 1.1. These standards are an extension to an old IEEE standard referred to as IEEE802.1Q: Bridges and Bridged Networks. The set of standards defined can be grouped in three categories each representing an essential component required in a TSN network. An overview of the standards in the three categories is as follows:

- **Time synchronization**: To correctly implement traffic shaping and scheduling mechanisms correctly, a common understanding of time among all the participating nodes is quite *essential*, i.e., the nodes must be time-synchronized. To this end, the Precision Time Protocol (PTP) or IEEE 1588 is used [45]. Through a continuous exchange of messages across the network clocks can be synchronized to a reference clock. Through time synchronization all nodes can transmit packets at their scheduled times. To adapt PTP specifically for AVB and TSN, IEEE 802.1AS has been proposed [46].

- **Traffic shaping and scheduling**: To ensure zero packet loss and guaranteed latency all participating nodes must smooth out their output traffic. Without traffic shaping, the packet can form a burst that can, in worst-case, grow in size as the flow traverses through the network [47]. A subsequent node might have to drop the packets as the buffers are overwhelmed by the burst.
  Credit-based Shaper (CBS) or IEEE 802.1Qav is a traffic shaping

mechanism based on credit-based fair queuing. The credits are zero when there is no frame in the queue [48]. The credits increase @ *IdleSlope* rate when a frame is queued (other queue is sending) and decrease @ *SendSlope* rate when a frame is being sent. A queue can only send if the link is not busy and it has non-negative credits. For each priority, the latency per switch can be bounded. However, the per-node queuing delay with CBS could be as high as 250μs that might not be suitable for some real-time applications.
Two additional key traffic shaping mechanisms: Time-aware Shaping (TAS) and Cyclic Queuing and Forwarding (CQF), will be explained in detail later.

- **Resource reservation, Path Control and Redundancy**: IEEE 802.1Qca: Path Control and Reservation is a standard that specifies procedures to configure multiple paths based on the Intermediate Station (IS-IS) protocol in Ethernet networks [49]. It extends the IS-IS control beyond the Shortest Path Trees (SPTs) by adding options for non-shortest paths or explicit path configuration. A Path Computation Element (PCE) residing in a dedicated server can be used to compute explicit paths.
IEEE 802.1Qcc describes how to manage and control the network centrally [50]. Using the User-Network Interface (UNI), the endstations can communicate with the Centralized Network Controller (CNC), e.g., a flow reservation request.
Packet losses due to congestion in Ethernet switches can be prevented by applying the above traffic shaping mechanisms. However, to protect against packet losses due to equipment failures in the network, Frame Replication and Elimination for Reliability (FRER), as documented in IEEE 802.1CB, is proposed [51]. The TT flow packets are sequenced and replicated into multiple (e.g., 1+1) flows in the network. Near the destination node, packets from the replicated flows are identified and duplicate packets are eliminated.

Another widely employed traffic shaping mechanism in TSN is IEEE 802.1Qbv or TAS [52]. Time-aware shaping is based on the idea of dividing time into cycles of fixed length with each cycle further divided into slots that can be allocated exclusively to different flows. This Time-division Multiple Access (TDMA) -like approach ensures that the real-time traffic can co-exist with the non-critical best-effort traffic. Next, we explain the working of a TAS-enabled switch.
Fig. 1.14 illustrates the internals of a typical TSN switch. After the selec-

Table 1.1: Overview of various TSN standards.

Standard	Description
IEEE 802.1AS-Rev	Timing and synchronization for time-sensitive Applications [46].
IEEE 802.1Qav	Prioritization of time-sensitive streams over best-effort using Credit Based Shaper (CBS) [48].
IEEE 802.1Qbv	Scheduling of time-sensitive traffic using time-aware shaping [52].
IEEE 802.1Qca	Path control and reservation [49].
IEEE 802.1Qci	Per-stream Filtering and Policing [53].
IEEE Std 802.1CB-2017	Frame Replication and Elimination for Reliability [51].

tion of the egress port through the switching fabric, the packet is assigned to one of the multiple First-in-first-out (FIFO) queues based on the priority code point (PCP) value of the Ethernet header. Each queue is associated with a gate whose state (open/close) determines if the corresponding queue can transmit or not. The enqueued packets are transmitted on the egress port in the FIFO fashion. In case multiple queue gates are open at a time, the queue with the highest priority gets to transmit. A Gate Control Entry (GCE) defines which queue will be opened during a specific interval of time. For instance, the third GCE corresponds to a time interval of length T_3 seconds for which q_7 is opened whereas all other gates are closed q_0. The sequence of GCEs is called Gate Control List (GCL) that repeats after a fixed period referred to as the cycle time.

A TSN mechanism called frame preemption (IEEE 802.3br and IEEE 802.1Qbu) allows suspending the transmission of preemptable frames (e.g., BE frame) for a so-called express frame (e.g., TT frame) [54]. Frame preemption results in a decrease of jitter for express traffic.

1.5.1.2 DetNet

Similar to the IEEE TSN TG, the IETF DetNet WG aims to bring QoS like time synchronization, zero congestion loss, bounded delay and reliability in Layer 3 [55]. DetNet, in contrast to TSN, is expected to be deployed in a large scale network with the following characteristics:

- Long distance between network nodes leading to longer propagation delay

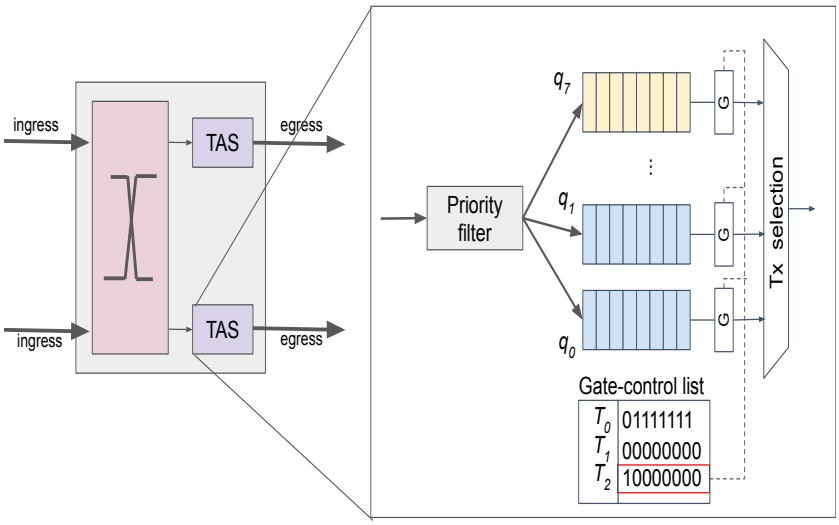


Figure 1.14: Illustration showing a simplified view of a TAS-enabled switch. During the interval corresponding to the third GCE, only q_7 is allowed to dequeue.

- Large number of devices in different network domains makes the task of precise synchronization among all nodes challenging.
- Large number (100s to 1000s) of flows on the network makes maintaining per-flow state in network nodes un-scalable.

Next, we investigate the queuing mechanism relevant to DetNet.
CQF or peristaltic shaper is a queuing mechanism described in IEEE 802.1Qch [56]. The time in CQF-enabled nodes is divided into cycles of lengths equal to T_{cyc}. Each egress port of a node contains two queues (q_0 and q_1). During cycle i, queue q_0 is allowed to dequeue its packets which are transmitted to the downstream node, whereas q_1 during this cycle enqueues packets received from its upstream nodes as shown in Fig. 1.15. In the next cycle $(i+1)$, the roles are reversed, i.e., q_0 enqueues and q_1 dequeues. The cyclic enqueuing and dequeuing of packets is also referred to as ping-pong buffering. Because of ping-pong buffering of queues, the packets transmitted by the node in a cycle (i) were received in the previous cycle $(i-1)$. This entails that the maximum delay experienced per hop is $2T_{cyc}$, i.e., corresponding to the packet enqueued at the beginning of cycle i and dequeued at the end of cycle $i+1$.

The simple queuing mechanism in CQF, i.e., receiving packets in the same

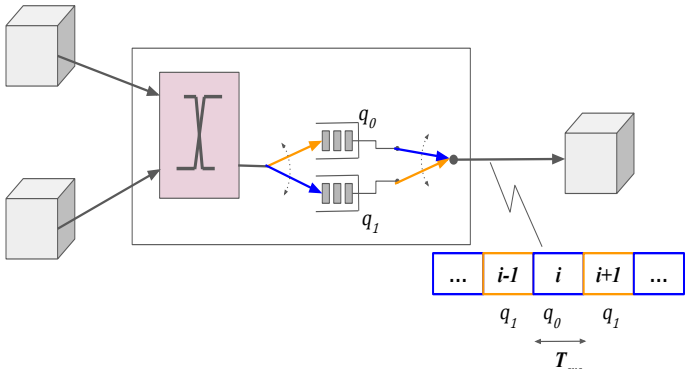


Figure 1.15: Illustration showing CQF operation in a node.

cycle in which they were transmitted, has a serious disadvantage. An upstream node is not allowed to be transmitted during the whole cycle; a guard band time at the end of the cycle has to be kept to ensure the above condition. Thus CQF is not suitable for long-distance for a given T_{cyc} value. To ensure this condition, a large amount of bandwidth is wasted as a guard time is kept for each cycle making CQF inefficient for DetNet.

The inefficiency of CQF can be mitigated by having at least three (one extra as compared to CQF) buffers per egress port and specifying the cycle time in which the packet should be transmitted. This queuing model is called Cycle Specified Queueing and Forwarding (CSQF). The common approach of specifying cycle time in a packet is discussed next. The CSQF operation with three queues is shown in Fig. 1.16 [57]. It can be observed that the packet sent by the upstream node in cycle i is received by the node during cycle j which then re-transmits it to the downstream node in cycle $j + 1$. At any time, one queue is used for transmission and the other two queues are used for reception. The time spent for transmission corresponds to one CSQF cycle time. After every T_{cyc} seconds, the next queue is considered for transmission resulting in a round-robin queue selection.

A centralized controller computes the transmission cycle for each node on the path and therefore the queue to which the packet should be enqueued. At the ingress node, packets are appended with a stack of queue identifiers. The controller with the global view of the network and flow requirements can ensure bandwidth allocation in each cycle does not exceed the available bandwidth. By using the stack of identifiers, the operation of network nodes is quite simple, i.e., to enqueue the received packet based on the queue identifier. A minimum of two bits (corresponding to three or more queues) per

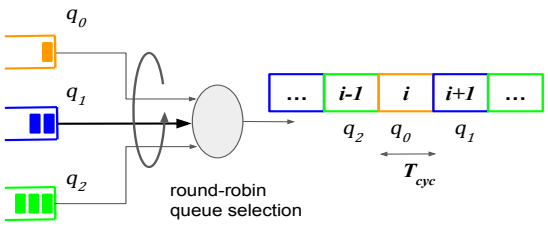


Figure 1.16: Illustration for queuing operation in a CSQF-enabled node.

egress port are required to identify a queue. The identifiers can be contained in, e.g., the DSCP field of IPv4 header, SID of SRv6, traffic class of IPv6 header, etc [58].
Consider three CSQF-nodes adjacent to each other as shown in Fig. 1.17. The packet received in cycle j is sent in cycle $j + 1$. It can be deduced that the end-to-end queuing delay on a path of length N hops is $2NT_{cyc}$, whereas the jitter or delay variation is equal to $2T_{cyc}$ [57]. It is worth pointing out that the jitter is still bounded although the total queuing delay is dependent on the number of hops. The key benefit with CSQF is that the constraint that transmission (from the upstream node) and reception (at the downstream node) has to occur in the same cycle is no longer applicable. Thus, the propagation delay guard time is not needed resulting in an efficient bandwidth utilization.

1.6 Outline and Research contributions

In this section, we provide an outline of this doctoral dissertation and highlight the key research contributions made. The dissertation focuses on optimization algorithms for (i) the deployment of virtualized network and media services and (ii) packet scheduling in the context of real-time applications. Thus, the contributions made in this dissertation can be grouped under these two main themes. The general outline of the dissertation is given in Fig. 1.18.
Next, we give an overview of various research problems considered under each theme.

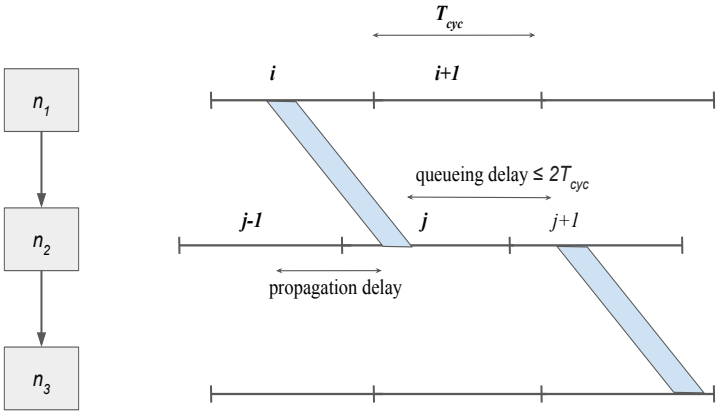


Figure 1.17: Packet forwarding between the three adjacent CSQF-enabled nodes. The packet sent by n_1 in cycle i is received by n_2 in cycle j which is re-sent to n_3 in cycle $j+1$.

1.6.1 Service Deployment

The first research theme in the dissertation is service deployment in the context of virtualized network and media services. Hardware accelerators are increasingly becoming important in NFV environments because of their potential to address the NFV performance and power consumption challenge. Efficient utilization of all resources including hardware accelerators is important in order to achieve overall cost reductions as envisaged by the NFV proposal [10]. To this end, we investigated NFV resource allocation with the awareness of hardware accelerators in Chapter 2. This problem is referred to as Accelerator-aware VNF Placement and Chaining (VNF-AAPC), i.e., how to allocate NFVi resources including hardware accelerators to VNF-chains in a cost-efficient manner. We propose an Integer Linear Program (ILP) model that jointly optimizes VNF placement, chaining and accelerator allocation process. A heuristic-based method is also proposed to solve the problem, though avoiding the scalability issues in the ILP approach.
Similar to telecom operators, broadcasters desire to bring down their expenditures in the times of falling APRU. To this end, architectures are foreseen where media services can be realized using Virtual Media Functions (VMFs) analogous to the NFV architecture where a network services is realized via VNFs. In addition to the benefits described in Chapter 2 that result from the NFV transformation, additional opportunities resulting from MFV can exploited to further improve resource utilization. In Chapter 3, a generalized algorithm is proposed that decomposes a given VMF Forwarding Graph

(representation of virtual media service). In addition, two VMF Placement and Chaining (VMF-PC) algorithms: Next-Fit and k-cut are proposed.
The solution to the VNF-AAPC problem presented in Chap 2 results in a static allocation of accelerators to VNFs. The static allocation of accelerators is not efficient in a scenario when the traffic flowing through VNFs, thus their CPU requirements, varies with time. To tackle this problem, we present a method to dynamically allocate hardware accelerators in an NFV environment. A PoC for the scheme is presented and experimentally evaluated. As this PoC supplements the contributions made in Chap 2, it is contained in Appendix A.
A failure in the NFVi is followed by the recovery of the impacted VNF-chains. Appendix B deals with the problem of accelerator-aware VNF-chain recovery. In case of failures in the NFVi, the recovery of VNF-chains must take into account the (re)-allocation hardware accelerator. The accelerator-aware VNF-chain recovery problem extends the VNF-AAPC problem by restoring the impacted VNF-chains with prioritization. The accelerator-aware VNF-chain recovery As these contributions augment the contributions of Chapter 2, they are contained in Appendix B. The problem is formulated as an ILP and a heuristic is proposed that matches the performance of the ILP's performance in regard to the restoration of high and medium priority VNF-chains by tolerating a small penalty for low-priority VNF-chains.

1.6.2 Scheduling algorithms

The second research theme deals with the problem of end-to-end packet scheduling for time-sensitive applications.
The VMF-PC algorithms proposed in Chapter 3 do not provide any guarantees on the end-to-end latency of a deployed media service. In Chapter 4, we revisit the VMF-PC problem of MFV but guaranteeing the maximum latency and jitter between the adjacent VMFs of the deployed service. This can be ensured by employing a queuing mechanism like CSQF for packet scheduling between adjacent VMFs of the VMF-FG. The VMF-FG scheduling problem is addressed to ensure end-to-end timing guarantees.
Chapter 5 combines the packet scheduling mechanism described in Chapter 4 with dedicated protection. The traffic shaping mechanism like CSQF ensures zero packet loss due to congestion in the network but node or link failure can still cause service disruption. This can be avoided by employing a dedicated protection scheme like *1+1*. In other words, the problem is to find (i) two paths (*1+1*) between the endpoints of a DetNet flow and (ii) schedule packet transmissions on these paths. The packet scheduling on two paths should take into account the end-to-end latency on these paths such

that reliable recovery can take place at the destination. The problem is first formulated as an ILP and then two heuristics: greedy and Tabu-search, are proposed.
Wireless connectivity is preferable over wired connections in many environments (e.g., to control multiple robotic arms in a factory). To support real-time applications in a mixed wired-wireless network, packet scheduling is required not only on wired but also on wireless segments of the network. Chapter 6 focuses on the problem of end-to-end packet scheduling in such a wired-wireless mixed network. For a given set of time-sensitive flows and the mixed wired-wireless TSN-enabled network, the problem is to select routes for the flows and calculate packet schedules on these routes. An ILP formulation of the problem is proposed and a greedy-based heuristic is proposed to solve the problem in a reasonable amount of time. The influence of various request ordering criteria on the performance of heuristic is reported. In addition, the impact of wireless requests on the performance of the scheduling is investigated.
Finally, Chapter 7 concludes the dissertation with the key outcomes and discusses the possible future work.

1.6.3 Chapter Ordering

In Chapter 2, we optimize resource allocation in the context of NFV deployments. Chapter 3 continues the goal of efficient resource allocation but for MFV deployments. As media services require real-time network guarantees, Chapter 4 adapts the VMF-PC procedure of Chapter 3 to include packet scheduling in MFV deployments. Chapter 5 builds on Chapter 4 by adding dedicated protection (*1+1*) to CSQF-based packet scheduling for DetNet flows. Finally, Chapter 6 extends the goal of time sensitivity beyond wired networks by tackling the end-to-end scheduling problem in mixed wired-wireless networks.

1.7 Publications

The research results obtained during this PhD research have been published in various scientific journals and presented at a series of international conferences. The following list provides an overview of the publications in chronological order.

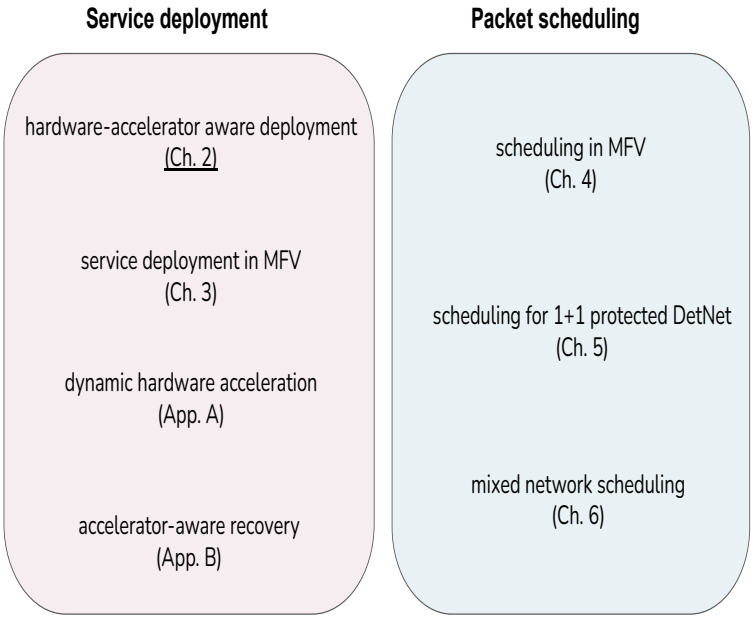


Figure 1.18: An overview of the key contributions made in this dissertation. For each contribution, the related chapter or appendix is indicated.

1.7.1 Publications in international journals (listed in the Science Citation Index [1])

1. Gourav Prateek Sharma, Wouter Tavernier, Didier Colle, and Mario Pickavet, "VNF-AAPC: Accelerator-aware VNF Placement and Chaining," Published in *Computer Networks*, vol. 177, 2020.

2. Gourav Prateek Sharma, Didier Colle, Wouter Tavernier, and Mario Pickavet, "On Decomposition and Deployment of Virtualized Media Services," Published in the *IEEE Transactions on Broadcasting*, vol. 67, no. 3, pp. 761–775, 2021.

3. Gourav Prateek Sharma, Wouter Tavernier, Didier Colle, and Mario Pickavet, "Routing and Scheduling for *1+1* Protected DetNet flows," Accepted in *Computer Networks*, 2022.

4. Gourav Prateek Sharma, Wouter Tavernier, Didier Colle, Mario Pickavet, Jetmir Haxhibeqiri, Jeroen Hoebeke and Ingrid Moerman, "End-to-end Scheduling for Wired-wireless Mixed Networks," Submitted to *Computer Communication*, 2022.

1.7.2 Publications in other international journals

1. Gourav Prateek Sharma, Wouter Tavernier, Didier Colle, and Mario Pickavet, "Scheduling for Media Function Virtualization," Published in *Future Internet*, vol. 13, no. 7, 2021.

1.7.3 Publications in international conferences (listed in the Science Citation Index [2])

1. Gourav Prateek Sharma, Wouter Tavernier, Didier Colle, and Mario Pickavet, "Dynamic accelerator provisioning for SSH tunnels in NFV environments," Published in the Proceedings of the *IEEE Conference On Network Softwarization (Netsoft)*, Paris, France, 2019, pp. 242–244.

[1] The publications listed are recognized as 'A1 publications', according to the following definition used by Ghent University: A1 publications are articles listed in the Science Citation Index Expanded, the Social Science Citation Index or the Arts and Humanities Citation Index of the ISI Web of Science, restricted to contributions listed as article, review, letter, note or proceedings paper.

[2] The publications listed are recognized as 'P1 publications', according to the following definition used by Ghent University: P1 publications are proceedings listed in the Conference Proceedings Citation Index - Science or Conference Proceedings Citation Index - Social Science and Humanities of the ISI Web of Science, restricted to contributions listed as article, review, letter, note or proceedings paper, except for publications that are classified as A1.

2. Gourav Prateek Sharma, Wouter Tavernier, Didier Colle, and Mario Pickavet, "Dynamic hardware-acceleration of VNFs in NFV environments," Published in the Proceedings of the *International Conference on Software Defined Systems (SDS)*, Rome, Italy, 2019, pp. 254–259.

3. Gourav Prateek Sharma, Wouter Tavernier, Didier Colle, and Mario Pickavet, "hardware accelerator aware VNF-chain recovery," Published in the Proceedings of the *International Conference on the Design of Reliable computer networks (DRCN)*, Milan, Italy (virtual), 2020.

1.7.4 Publications in other international conferences

1. Gourav Prateek Sharma, Didier Colle, Wouter Tavernier, and Mario Pickavet, "VNF-AAP: Accelerator-aware Virtual Network Function Placement," Published in the Proceedings of the *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Dallas, USA, 2019.

2. Gourav Prateek Sharma, Didier Colle, Wouter Tavernier, and Mario Pickavet, "Improving resource utilization with Virtual Media Function decomposition," Published in the Proceedings of the *International Conference on Multimedia Computing, Networking and Applications (MCNA)*, Valencia, Spain (virtual), 2020, pp. 31–37.

References

[1] *Cisco Annual Internet Report (2018–2023).* Whitepaper, 2020. Available from: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.pdf.

[2] *Coronavirus: Belgian internet traffic nearly doubles.* https://www.brusselstimes.com/brussels-2/100816/coronavirus-belgian-internet-traffic-nearly-doubles. (Accessed on 02/16/2022).

[3] *Wireless Services in Europe: A Mixed Bag for Operators.* https://blog.telegeography.com/wireless-services-subscribers-in-europe-2g-3g-4g-5g. (Accessed on 02/16/2022).

[4] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar. *Making middleboxes someone else's problem: Network processing as a cloud service.* ACM SIGCOMM Computer Communication Review, 42(4):13–24, 2012.

[5] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury. *Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research.* IEEE Communications Surveys & Tutorials, 21(1):88–145, 2018.

[6] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach Edition.* Addision Wesley, 2007.

[7] J. H. Saltzer, D. P. Reed, and D. D. Clark. *End-to-end arguments in system design.* ACM Transactions on Computer Systems (TOCS), 2(4):277–288, 1984.

[8] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. *Network function virtualization: State-of-the-art and research challenges.* IEEE Communications surveys & tutorials, 18(1):236–262, 2015.

[9] D. Gedia and L. Perigo. *Performance evaluation of SDN-VNF in virtual machine and container.* In 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pages 1–7. IEEE, 2018.

[10] *ETSI GS NFV-SEC 012 Network Functions Virtualisation (NFV) Release 3; Security; System architecture specification for execution of sensitive NFV components.* Group specification, ETSI ISG, January 2017. Available from: https://www.etsi.org/deliver/etsi_gs/nfv-man/001_099/001/01.01.01_60/gs_nfv-man001v010101p.pdf.

[11] J. G. Herrera and J. F. Botero. *Resource allocation in NFV: A comprehensive survey.* IEEE Transactions on Network and Service Management, 13(3):518–532, 2016.

[12] ETSI. *Open Source MANO.* Available from: https://www.onap.org/.

[13] The Linux Foundation. *Open Network Automation Platform (ONAP).* Available from: https://www.onap.org/.

[14] M. T. Raza, D. Kim, K.-H. Kim, S. Lu, and M. Gerla. *Rethinking LTE network functions virtualization.* In 2017 IEEE 25th International Conference on Network Protocols (ICNP), pages 1–10. IEEE, 2017.

[15] L. Peterson and O. Sunay. *5G mobile networks: A systems approach.* Synthesis Lectures on Network Systems, 1(1):1–73, 2020.

[16] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. *Software-defined networking: A comprehensive survey.* Proceedings of the IEEE, 103(1):14–76, 2014.

[17] Open Networking Foundation. *Open Network Operating System.* Available from: https://opennetworking.org/onos/.

[18] *RYU the Network Operating System(NOS).* Available from: https://ryu-sdn.org/resources.html.

[19] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. *OpenFlow: enabling innovation in campus networks.* ACM SIGCOMM computer communication review, 38(2):69–74, 2008.

[20] E. Kohler. *The Click Modular Router.* Available from: https://github.com/kohler/click.

[21] T. L. Foundation. *Open Virtual Switch.* Available from: https://www.openvswitch.org/.

[22] NVIDIA. *NVIDIA Cumulus Linux.* Available from: https://www.nvidia.com/en-us/networking/ethernet-switching/cumulus-linux/.

[23] D. Bhamare, R. Jain, M. Samaka, and A. Erbad. *A survey on service function chaining.* Journal of Network and Computer Applications, 75:138–155, 2016.

[24] L. Linguaglossa, S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, and G. Bianchi. *Survey of performance acceleration techniques for network function virtualization.* Proceedings of the IEEE, 107(4):746–764, 2019.

[25] C. Kachris, G. Sirakoulis, and D. Soudris. *Network function virtualization based on FPGAs: A framework for all-programmable network devices.* arXiv preprint arXiv:1406.0309, 2014.

[26] Z. Bronstein, E. Roch, J. Xia, and A. Molkho. *Uniform handling and abstraction of NFV hardware accelerators.* IEEE Network, 29(3):22–29, 2015.

[27] X. Ge, Y. Liu, D. H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu. *OpenANFV: Accelerating network function virtualization with a consolidated framework in openstack.* ACM SIGCOMM Computer Communication Review, 44(4):353–354, 2014.

[28] Z. Martinasek, J. Hajny, D. Smekal, L. Malina, D. Matousek, M. Kekely, and N. Mentens. *200 Gbps hardware accelerated encryption system for FPGA network cards.* In Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security, pages 11–17, 2018.

[29] T. Kojima, J. J. Stone, J.-R. Chen, and P. N. Gardiner. *A practical approach to IP live production.* SMPTE Motion Imaging Journal, 124(2):29–40, 2015.

[30] A. Kovalick. *Design elements for core ip media infrastructures.* SMPTE Motion Imaging Journal, 125(2):16–23, 2016.

[31] F. Poulin, P. Keroulas, S. Nyamweno, W. Vermost, P. Ferreira, and I. Kostiukevych. *How CBC/radio-Canada tested media-over-IP devices to build its new facility.* SMPTE Motion Imaging Journal, 129(4):35–44, 2020.

[32] S. Sneddon, C. Swisher, and J. Mayzurk. *Large Scale Deployment of SMPTE 2110: The IP Live Production Facility.* In SMPTE 2019, pages 1–31. SMPTE, 2019.

[33] *ST 2110-20:2017 - SMPTE Standard - Professional Media Over Managed IP Networks: Uncompressed Active Video.* ST 2110-20:2017, pages 1–22, 2017. doi:10.5594/SMPTE.ST2110-20.2017.

[34] *ST 2110-30:2017 - SMPTE Standard - Professional Media Over Managed IP Networks: PCM Digital Audio.* ST 2110-30:2017, pages 1–9, 2017. doi:10.5594/SMPTE.ST2110-30.2017.

[35] *ST 2110-40:2018 - SMPTE Standard - Professional Media Over Managed IP Networks: SMPTE ST 291-1 Ancillary Data.* ST 2110-40:2018, pages 1–8, 2018. doi:10.5594/SMPTE.ST2110-40.2018.

[36] *ST 2022-8:2019 - SMPTE Standard - Professional Media Over Managed IP Networks: Timing of ST 2022-6 Streams in ST 2110-10 Systems.* ST 2022-8:2019, pages 1–10, 2019. doi:10.5594/SMPTE.ST2022-8.2019.

[37] L. Gharai and C. Perkins. *RTP payload format for uncompressed video.* RFC, 4566, 2005.

[38] B. Research and Development. *Compositing and Mixing Video in the Browser*, 2018. Available from: https://www.bbc.co.uk/rd/blog/2017-07-compositing-mixing-video-browser.

[39] Grass Valley. *Agile Media Processing Platform.*

[40] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet. *VNF-AAPC: Accelerator-aware VNF placement and chaining.* Computer Networks, 177:107329, 2020.

[41] Y. Reznik, J. Cenzano, and B. Zhang. *Transitioning broadcast to cloud.* Applied Sciences, 11(2):503, 2021.

[42] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan. *Reducing web latency: the virtue of gentle aggression.* In Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM, pages 159–170, 2013.

[43] *DiffServ – The Scalable End-to-End QoS Model.* Whitepaper, Cisco, 08 2005. Available from: https://www.cisco.com/en/US/technologies/tk543/tk766/technologies_white_paper09186a00800a3e2f.html.

[44] *IEEE Standard for Local and metropolitan area networks–Audio Video Bridging (AVB) Systems.* IEEE Std 802.1BA-2011, pages 1–45, 2011. doi:10.1109/IEEESTD.2011.6032690.

[45] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.* IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008), pages 1–499, 2020. doi:10.1109/IEEESTD.2020.9120376.

[46] *IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*. IEEE Std 802.1AS-2011, pages 1–292, 2011. doi:10.1109/IEEESTD.2011.5741898.

[47] S. Sharma, D. Colle, W. Tavernier, M. Pickavet, and P. Demeester. *Inter-burst segregation protocol guaranteeing loss-free packet-switched networks*. IEEE Communications Letters, 20(10):1959–1962, 2016.

[48] *IEEE Standard for Local and Metropolitan Area Networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams*. IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005), pages C1–72, 2010. doi:10.1109/IEEESTD.2009.5375704.

[49] *IEEE Standard for Local and metropolitan area networks— Bridges and Bridged Networks - Amendment 24: Path Control and Reservation*. IEEE Std 802.1Qca-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qcd-2015 and IEEE Std 802.1Q-2014/Cor 1-2015), pages 1–120, 2016. doi:10.1109/IEEESTD.2016.7434544.

[50] *IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements*. IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018), pages 1–208, 2018. doi:10.1109/IEEESTD.2018.8514112.

[51] *IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability*. IEEE Std 802.1CB-2017, pages 1–102, 2017. doi:10.1109/IEEESTD.2017.8091139.

[52] *IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic*. IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015), pages 1–57, 2016. doi:10.1109/IEEESTD.2016.8613095.

[53] *IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing*. IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE

Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016), pages 1–65, 2017. doi:10.1109/IEEESTD.2017.8064221.

[54] *IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption.* IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014), pages 1–52, 2016. doi:10.1109/IEEESTD.2016.7553415.

[55] N. Finn, P. Thubert, B. Varga, and J. Farkas. *Deterministic networking architecture.* RFC 8655, 2019.

[56] *IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 29: Cyclic Queuing and Forwarding.* IEEE 802.1Qch-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd(TM)-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, IEEE Std 802.1Qbz-2016, and IEEE Std 802.1Qci-2017), pages 1–30, 2017. doi:10.1109/IEEESTD.2017.7961303.

[57] M. Chen, X. Geng, and Z. Li. *Segment Routing (SR) Based Bounded Latency.* Internet Engineering Task Force, Internet-Draft draft-chendetnet-sr-based-bounded-latency-00, 2018.

[58] E. L. Qiang, X. Geng, B. Liu, and E. T. Eckert. *Large-Scale Deterministic IP Network.* Internet Engineering Task Force, Internet-Draft draft-qiang-detnet-large-scale-detnet-04, 2019.

2

VNF-AAPC: Accelerator-aware VNF Placement and Chaining

Hardware acceleration is currently being proposed as a solution to address the two key challenges faced by telecom operators when adopting NFV: performance degradation and high energy consumption. By offloading compute-intensive tasks from VNFs to hardware accelerators, packet processing performance is improved along with a reduction in CPU utilization. This chapter proposes resource allocation mechanisms that take into account hardware accelerators in NFV environments. These mechanisms optimize resource utilization so that telecom operators can reduce their overall expenditure.

⋆⋆⋆

G.P. Sharma, W. Tavernier, D. Colle, and M. Pickavet

Published in Computer Networks, vol. 177, 2020.

Abstract In recent years, telecom operators have been migrating towards network architectures based on Network Function Virtualization in order to reduce their high Capital Expenditure (CAPEX) and Operational Expenditure (OPEX). However, the virtualization of some network functions is accompanied by a significant degradation of Virtual Network Function (VNF) performance in terms of their throughput or energy consumption.

To address these challenges, the use of hardware accelerators, e.g. FPGAs, GPUs, to offload CPU-intensive operations from performance-critical VNFs has been proposed.
Allocation of NFV infrastructure (NFVi) resources for VNF placement and chaining (VNF-PC) has been a major area of research recently. A variety of resources allocation models have been proposed to achieve various operator's objectives i.e. minimizing CAPEX, OPEX, latency, etc. However, the VNF-PC resource allocation problem for the case when NFVi incorporates hardware accelerators remains unaddressed. Ignoring hardware accelerators in NFVi while performing resource allocation for VNF-chains can nullify the advantages resulting from the use of hardware accelerators. Therefore, accurate models and techniques for the accelerator-aware VNF-PC (VNF-AAPC) are needed in order to achieve the overall efficient utilization of all NFVi resources including hardware accelerators.
This paper investigates the problem of VNF-AAPC, i.e., how to allocate usual NFVi resources along with hardware accelerators to VNF-chains in a cost-efficient manner. Particularly, we propose two methods to tackle the VNF-AAPC problem. The first approach is based on Integer Linear Programming (ILP) which jointly optimizes VNF placement, chaining and accelerator allocation while concurring to all NFVi constraints. The second approach is a heuristic-based method that addresses the scalability issue of the ILP approach. The heuristic addresses the VNF-AAPC problem by following a two-step algorithm.
The experimental evaluations indicate that incorporating accelerator awareness in VNF-PC strategies can help operators to achieve additional cost savings from the efficient allocation of hardware accelerator resources.

2.1 Introduction

The incessant expansion in the number of connected users and network services has resulted in an exponential growth of traffic on the networks of telecom operators. Telecom infrastructure thus needs to be scaled periodically to cope with the increasing traffic demands which result in high Capital Expenditure (CAPEX) and Operational Expenditure (OPEX). However, the growth in Average Revenue Per User (ARPU) has been very marginal due to the cut-throat competition among the operators. As a result, operators are forced to seek new network architectures that are scalable, agile and cost-efficient [1].
Network Function Virtualization (NFV) is a technology that leverages IT virtualization techniques for consolidating network appliances onto commercial-off-the-shelf (COTS) server machines. NFV aims to replace Network Func-

tions (NFs) based on proprietary ASICs, also known as *middleboxes*, by their software instances running on the general-purpose platforms consisting of x86 or ARM-based high-volume servers (HVS). The software implementation of an NF running in a virtualized environment is called Virtual Network Function (VNF). Fig 2.1 shows the reference architecture of NFV as proposed by ETSI [2].

The purpose of the virtualization layer is to abstract the NFV Infrastructure (NFVi) layer, which includes the compute, storage and networking resources, from VNFs running over it. Various virtualization technologies, e.g. VMs, containers, are exploited for the realization of the virtualization layer.

Replacing network services based on middleboxes with VNF-chains running

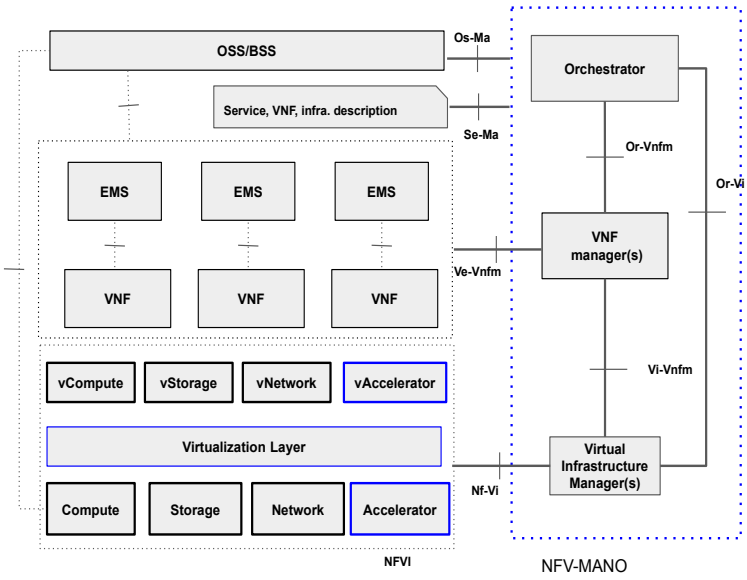


Figure 2.1: Reference NFV architecture by ETSI [2]. Components shown in blue needs to be added/updated as a result of inclusion of hardware-acceleration in NFVi.

on COTS servers has several advantages, such as the reduction in CAPEX and OPEX, faster time-to-market (TTM) of services, ease of service management and upgrade, etc [1]. Although, NFV offers several advantages, replacing NFs middleboxes with VNFs can have a detrimental effect on their packet-processing performance, e.g. loss of throughput and/or undeterministic latency. Furthermore, the growth in the computational ca-

pacity of CPUs is flattening with time due to an expected end of Dennard's scaling and Moore's law in the coming years [3]. Performance improvement of software-based packet-processing platforms is expected to fall short as compared to the increasing data traffic on telecom networks. Therefore, matching the performance of middleboxes will be one of the key challenges faced by operators in the future too with regards to the widespread NFV adoption. This challenge has led to a recent interest in hardware-acceleration techniques for VNFs using externally connected hardware devices, e.g. Graphics Processing Units (GPUs), Field Programmable Gate Arrays (FPGAs), Network Processing Units (NPUs), etc. Hardware accelerators and CPUs can be used in conjunction such that CPU-intensive tasks can be offloaded from VNFs to hardware accelerators and the rest of the VNF operations can be performed by the CPU of general-purpose hardware (COTS servers). As a consequence, an improvement in the overall packet-processing performance can be achieved.

Due to the upward trend of outsourcing network processing to the cloud, data centers (DCs) are being considered as NFVi. The share of energy costs in a DC, which includes the cost of energy spent in servers, switches and cooling of DCs, mainly constitutes the OPEX cost. A large number of VNF CPU cycles are consumed in packet-processing tasks which otherwise consume a fraction of energy if implemented in the hardware. For example, using hardware-acceleration to offload iFFT/FFT in cloud-RAN (C-RAN) scenarios to FPGAs, GPUs or DSP can result in power saving by about 70% per carrier [4]. As a consequence, additional VNFs can be accommodated on the same NFVi as some CPU cores are freed because of the offload to hardware accelerators.

Accelerators resources are being increasingly integrated with the NFVi layer along with the usual compute, network and storage (Fig. 2.1). However, the current Management and Orchestration (MANO) layer is mostly unaware of the acceleration requirements of VNFs and the location of hardware accelerators in NFVi. As a result, the resource allocation decisions taken by the orchestrator are agnostic to VNF requirements and the locality of hardware accelerator resources. This could lead to sub-optimal utilization of NFVi resources. Particularly, the inefficient allocation of hardware accelerator resources can negate the advantages resulting from the use of hardware accelerators in NFV environments.

The overview of the accelerator-agnostic and accelerator-aware resource allocation procedure for VNF instantiation is depicted in Fig. 2.2 (a) and (b), respectively [2]. For the regular accelerator-agnostic VNF orchestration procedure (Fig. 2.2 (a)), the NFV Orchestrator (NFVO) first validates the received VNF instantiation request and passes the corresponding VNF

descriptor (VNFD) to the VNF Manager (VNFM). As the VNFM is agnostic to accelerator requirements of the VNF or existence of any offload capability in NFVi, it requests the reservation of regular NFVi resources (compute, storage, and network) via the Virtual Infrastructure Manager (VIM) which in turn allocates VMs/containers for the VNF and attach them to the network. The VIM acknowledges the NFVO when the resource reservation is complete. Further, a deployment-specific configuration of VMs/containers can be performed through the corresponding VNFM after the VNF instantiation is completed. The instantiated VNF cannot offload its operations to a hardware accelerator as it is not allocated any such special resource. However, the NFVO can ask the VIM to reserve hardware accelerator resources for the VNF if it is aware of specific VNF requirements and the presence of offload capabilities in NFVi as shown in Fig. 2.2 (b). After processing the accelerator requirements mentioned in the VNFD, the VNFM requests resource allocation including hardware accelerator resources. The instantiated VNF can now offload specific operations depending on the available types of accelerator implementations and the amount of resources.

As discussed above, accelerator resource management involves coordina-

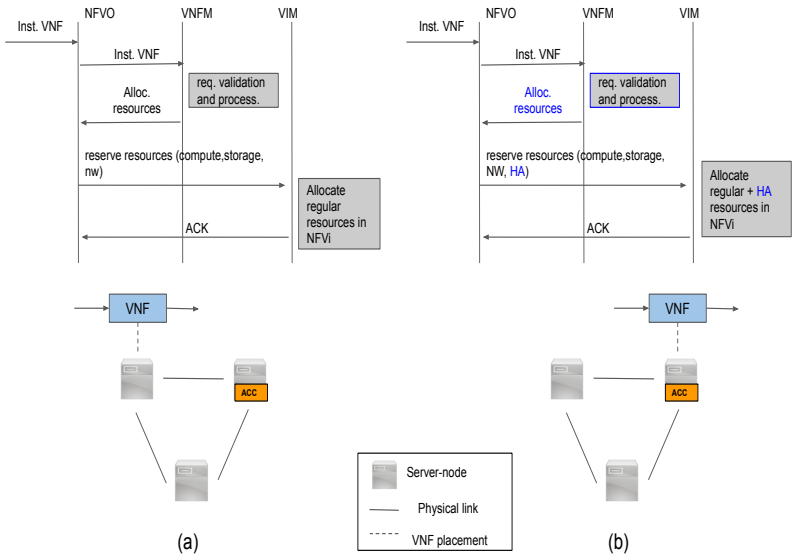


Figure 2.2: Processes involved in (a) accelerator-agnostic and (b) accelerator-aware VNF instantiation.

tion between various MANO elements such as the NFVO, the VNFM and

the VIM. The management requires acceleration-specific MANO functions such as acceleration feature discovery, acceleration lifecycle management, acceleration management. We shall not describe these aspects here as the focus this chapter is on resource allocation. A detailed discussion on the relevant functions and processes for acceleration management in NFV can be found in [5].
In order to achieve efficient utilization of all NFVi resources, it is imperative to incorporate accelerator awareness in the existing resource allocation models for NFV. VNF-PC is the most important component of the NFV resource allocation procedure. The VNF-PC problem is considered as an NP-hard problem and has been a widely researched topic in the literature [6]. With the inclusion of hardware accelerator resources in NFVi, solving only the VNF-PC problem is not sufficient to obtain an efficient allocation of NFVi resources. The VNF-PC problem needs to be altered in order to incorporate the resource allocation component for hardware accelerators. We refer to this new problem as the Accelerator-aware VNF Placement and Chaining Problem (VNF-AAPC). Our objective in this paper is to model the VNF-AAPC problem and propose a scalable approach to solve this problem in a time-efficient manner. In order to address the above-mentioned objective, we make the following contributions in this paper:

1. To obtain optimal solutions for the VNF-AAPC problem, we present an Integer Linear Program (ILP) formulation of this problem. It is a single-step exact method which jointly optimizes three decisions, namely– (i) VNF placement and (ii) chaining and (iii) accelerator allocation.

2. We design an efficient heuristic to solve the VNF-AAPC problem for DC topologies. This heuristic is particularly useful for large-size instances of the VNF-AAPC problem where the ILP model becomes too time-consuming to solve.

3. We also evaluate the performance of the ILP model and the proposed heuristic on two different data center topologies. Furthermore, we compare the performance of accelerator-agnostic and accelerator-aware heuristics. Finally, we present an analysis of the achievable cost savings resulting from the use of hardware accelerators in NFVi.

Section 2.2 of this paper deals with the discussion about hardware-acceleration in NFV environments. Relevant literature in the domain of NFV resource allocation is presented in Section 2.3. Section 2.4 describes the ILP formulation of the VNF-AAPC problem. The proposed heuristic to solve the VNF-AAPC problem is discussed in section 2.6. Performance

evaluation results and comparison of the ILP model with our heuristic are reported in Section 2.7. Finally, the future works and conclusions of this paper are presented in section 2.8.

2.2 Hardware-acceleration in NFV

The transition of telecom's network architectures from the purpose-built network appliances to VNFs running on COTS servers still faces multiple challenges [1], [7]. One of the key obstacles is the virtualization of all NFs without breaking the Service Level Agreements (SLAs) of network services. However, it has been observed in many instances that the performance of VNFs is significantly degraded as compared to their hardware counterpart [8]. Authors in [8] investigated the impact of virtualizing firewall on its packet-processing performance. Processing latency in the virtual firewall could even reach ten times the processing latency in the case of the hardware firewall. Performance bench-marking of IPSec is reported in a white paper by Intel in [9]. The results show that the processing of 48Gbps IPSec traffic requires on average 9.5 CPU cores. The same traffic can, however, be processed using 4.6 CPU cores when accelerating AES-GCM de/encryption using a hardware accelerator resulting in a saving of about half of the CPU cores. Therefore, not just the performance boost of VNFs but also the overall reduction in CPU utilization paves the way for hardware accelerators in NFV environments.

A large number of VNFs involve CPU-intensive tasks like de-duplication, cryptography, compression, etc [9], [10], [11], [12], [13], [14], [15]. The software implementation of these tasks has been found to be very energy inefficient (numbers of operations performed/energy consumed) as compared to their hardware implementation resulting in excessive CPU utilization. The motivation behind using hardware accelerators in NFV environments is that specific VNF components run more efficiently if implemented in hardware as opposed to software running on a CPU of a general-purpose COTS server. For example, GPUs have been used to speedup video-transcoding applications (H.264 and H.265) by 9.6x over software-only solutions while being 6.4x more energy efficient [14].

Packet-processing in a VNF is usually accomplished by sequentially executing instructions of the VNF (VM/container) on one or more CPU cores. The packet-processing paradigm in architectures like FPGAs, NPUs, and GPUs is fundamentally different from that of a CPU. A GPU chip consists of thousands of computational cores that can be delegated execution units which are also known as GPU threads. Each GPU core executes the same NF on different packets sent by the CPU in the GPU memory [16]. With

the large thread-level parallelism of GPUs and good memory communication (low latency and high bandwidth), high packet-processing performance can be achieved. GPUNFV is a GPU-based NFV system which demonstrated line rate packet-processing for stateful VNFs (e.g. flow-monitor, firewall) by exploiting the parallelism of GPUs [17]. FPGAs, on the other hand, contain millions of logic elements each of which contains lookup tables (LUTs) for implementing combinational logic and registers to store intermediary results. FPGAs also contain Block RAM (BRAM) to store a large amount of data that needs to be read (written) from (to) the main memory (RAM). Logic elements on an FPGA can be configured to realize different packet-processing functionalities. The parallelism in CPUs and GPUs is limited to the number of cores it has. Due to the massive amount of parallelism available on an FPGA at the gate-level, many processing tasks can be easily pipelined [18]. As a result, packet-processing tasks in VNFs can be offloaded to an FPGA very efficiently.
Hardware acceleration can be applied to a variety of VNFs that can benefit from the different kinds of parallelism available on hardware accelerators. Table 2.1 lists various VNFs alongside their sub-tasks that can benefit from offloading to hardware accelerators. VNFs which contain components like cryptography, compression, de/encoding, etc, can be very efficiently offloaded to hardware accelerators. Using an example of IPSec VNF, we next describe the most common approach of hardware-acceleration in NFV, i.e., FPGA look-aside acceleration.

2.2.1 VNF hardware-acceleration example

IPSec tunneling is one of the most popular ways of securing inter-network communication between branch offices of an enterprise or LTE networks via encrypted tunnels [19]. Fig 2.3 (a) shows a standard IPSec setup. At one end of the IPSec tunnel, a VM containing IPSec application (e.g. libreswan [1]) is running on a server. The IPSec VNF must perform all the required cryptographic functions (en/decryption and SHA) on IPSec packets. These functions are usually provided by a software library (e.g. SSL) which contains implementations for various ciphers (e.g. DES-128, AES-128,256) and hashes (e.g. md5, SHA-256,512). Nowadays, certain CPU architectures (e.g. x86 and AMD) offer AES-NI and SHA-NI instructions dedicated for de/encryption and hashing operations which results in a better performance as compared to the traditional CPU architectures. Despite this improvement, a large number of CPU cores are still required to process the IPSec

[1] https://libreswan.org/

Table 2.1: List of VNFs whose performance was improved after the indicated tasks were offloaded using hardware accelerators.

VNF	Component functions for acceleration	References	Improvements
IPSec, SSH	AES en/de-cryption, SHA hash	[9], [10], [11], [12]	CPU usage reduction of 50% and 94% at packet size of 578B and 9000B, respectively [9].
DPI	Multihash, Bloomfilter, regex,	[12], [13]	20x throughput improvement [13].
Media Transcoding	VP8, H.264, H.256	[14]	9.6x gain in performance (FPS) and 6.4x overall efficiency (performance/watt).
vRAN	RS, FFT/iFFT, Turbo de/-coding	[4], [15]	C-RAN power consumption reduction from 70W/carrier to 18W/carrier when i/FFT are offloaded. Turbo decoding time can be reduced by 50-60% by offloading it to a accelerator.
Dedup	Rabin hash, marker selection, chunk hash	[13]	8.2x improvement in throughput over software-only Dedup VNF.

traffic at the line-rate, e.g., 9.5 CPU cores are required to handle IPSec traffic @ 48Gbps [9] as compared to only 3.3 CPU cores for processing of plain IP traffic (without IPSec). Moreover, packet-processing cost (CPU cycles/packet) varies with the packet size which makes software-based IPSec solution inefficient for the IPSec packets of longer lengths ($>$ 1200 B).
Next, we describe the look-aside VNF hardware-acceleration approach taking IPSec as an example. A hardware designer typically first writes the required hardware accelerator (e.g. AES-256, SHA-512) in a Hardware Descriptor Language (HDL), e.g. VHDL or Verilog. The HDL design is then compiled to a programming file, called bitfile using FPGA synthesis and implementation tools. The bitfile is then used to program the FPGA fabric in order to instantiate the desired accelerator function. The accelerator can then be modified or a new accelerator could be instantiated by re-programming the FPGA fabric with the bitfile corresponding to the new accelerator. This makes FPGAs re-programmable, unlike ASICs which offer a limited amount of configuration. In Fig. 2.3 (b), the AES (encryption

and decryption) and SHA hash accelerators are instantiated by downloading their bitfiles to the FPGA card. Now, AES-256 de/encryption and SHA-512 hash operations can be offloaded from the IPSec VNF to accelerators running on the FPGA card [9]. For each IPSec packet, its payload is sent

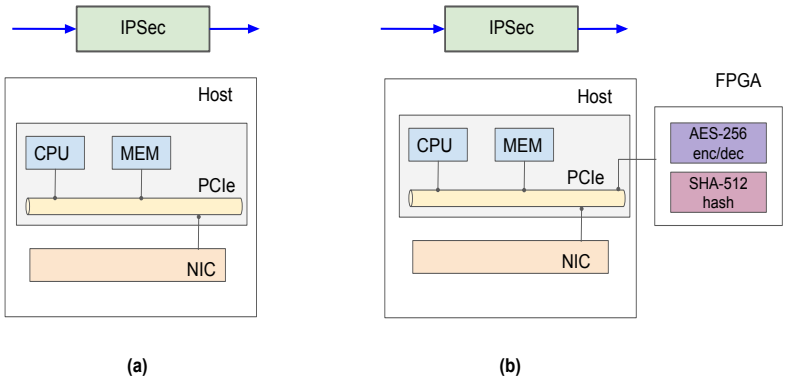


Figure 2.3: Illustration for the setup in (a) non-accelerated and (b) accelerated operation of IPSec VNF.

to the accelerator memory in order to perform the required cryptographic functions. After the function computation is over, the result of the operation is copied back from the accelerator memory to the main memory. The communication between the main memory and accelerators is accomplished via the PCIe bus. The overhead due to communications between CPUs and accelerators becomes insignificant for large packet sizes. Moreover, hybrid chips like Intel Xeon+FPGA integrated-FPGA CPUs provide tight coupling between CPUs and FPGAs thereby both CPUs and FPGAs can access the same memory and can avoid excessive overhead due to data transfers between them [20]. Nevertheless, many CPU cores are relieved from performing intensive cryptographic operations, thus a large number of CPU cores are free to run other workloads or VNFs [9].

2.2.2 Trade-offs

Although the highest programmability and flexibility can be achieved by running VNFs on CPUs (x86 or ARM), NF implementations based on technologies like GPUs, FPGAs, NPUs could be necessary for some performance-critical VNFs. Therefore, a spectrum of VNF implementation technologies

results in a variety of solutions, ranging at one end, the highly-flexible and full-software NFs and at the other end, the high-performance ASIC implementation with hardware accelerated VNFs situated in between. Authors in [19] proposed an architecture for the unified handling and abstraction of hardware accelerators in order to ease the manageability of accelerators. A virtual accelerator layer along with standard interfaces can be used in order to avoid compatibility and portability issues. This also helps to separate the concerns of VNF developers and hardware accelerators designers. By abstracting hardware accelerators, the same VNF image can be used for many hardware accelerators without any modification.
Fig. 2.4 illustrates the comparison between various VNF implementation technologies based on their performance and flexibility metrics [19]. A purpose-built ASIC implementation of an NF will offer the highest packet-processing performance but a very limited configuration will be possible e.g. updating the forwarding tables of a router. On the other hand, platforms based on COTS servers offer huge programmability/flexibility, e.g. update of protocols, at the cost of performance. Although, devices have intermediate performance and flexibility, e.g. GPUs and FPGAs can also be used to realize full VNFs, however; the more complex the packet processing task is, the more challenging it is to implement on an FPGA or GPU. Hybrid platforms with a combination of CPU + hardware accelerator (CPU+FPGA or CPU+GPU) are the most popular approach to achieving high-performance without losing too much programmability/flexibility. In hybrid platforms, the performance-critical tasks, e.g. en/decryption and hashing, etc, are implemented in the hardware and other complex tasks are still run in software running on a CPU.
Keeping into account service requirements and trade-offs of various technologies, telecom operators or third-party VNF developers have to select the right platform for their VNF implementation. For example, IPSec VNF running on a CPU when offloaded to an FPGA can improve its throughput and halve its CPU usage with a fraction of more investment.

There are two popular modes of using hardware accelerators in the NFV environments, namely– look-aside and bump-in-the-wire [21]. The look-aside mode of hardware-acceleration is generally used to offload compute-intensive algorithms, e.g., offloading crypto-operations of IPSec to an FPGA. "Bump-in-the-wire" (in-line) is another mode where packet processing is done on the fly, e.g. on P4 switches or smartNICs, as they are transferred to/from the network. Bump-in-the-wire mode is therefore preferred mode to accelerate first/last VNFs of a VNF-chain [3] as accelerating VNFs. In this paper, our focus will be on modeling scenarios with the look-aside mode of acceleration. However, to accommodate scenarios with bump-in-the-wire

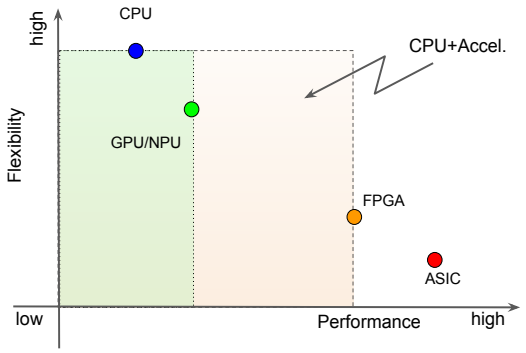


Figure 2.4: Comparison of various technologies for VNF implementation [19]. Green region: CPU+GPU and Orange region: CPU+FPGA.

acceleration, appropriate constraints regarding required position-aware acceleration and latency requirements can be added to the proposed model. Multiple VNFs running on a server node can also share the same accelerator instance deployed on hardware accelerator cards. The network packets reaching a VNF running on a host are transferred to a particular accelerator instance over the PCIe bus. Offloading packet processing from stateless VNFs to accelerator instances is straightforward, as the order of incoming packets is not important. To offload stateful VNFs, where the state of a VNF is required to process packets, input packets along with the VNF state are transferred to the accelerator instance [17]. The state in the VNF is updated after the completion of processing in the accelerator instance.

2.3 Related Works

Various mathematical models and algorithms have been proposed to tackle the VNF-PC problem. The solution to the VNF-PC problem attempts to allocate NFVi resources for the placement and chaining of VNFs. This problem is similar to Virtual Network Embedding (VNE) problem, a well-known problem in the area of network virtualization [6]. In the VNF-PC problem, VNFs are equivalent to virtual nodes of VNE which are chained by virtual links. In addition to that, VNF-PC has an accompanying optimization goal which is described by the objective function of the problem. The objec-

tive function could be the minimization of power consumption, the required number of server nodes, and links or maximization of resiliency, QoS, net profit, etc.

The VNF-PC problem has been tackled using two different approaches in the past. The first approach is to exploit exact methods that result in an optimal solution but this approach is generally useful for small-scale instances of the VNF-PC problem. Another approach to solve the VNF-PC problem is to use heuristics and thereby compromise a small amount of efficiency for scalability.

Using an ILP formulation, authors in [22] modeled resource allocation in a hybrid NFV scenario where services are provided using both dedicated hardware appliances and VNFs. This model was evaluated using two types of service chain requests and a small service provider scenario. A Mixed Integer Quadratically Constrained Program (MIQCP) model for the VNF-PC optimization problem was introduced in [23]. Pareto set analysis was performed to investigate the trade-offs between three different objective functions. The evaluation of the model shows the objective function (e.g. minimization of latency, link utilization or allocated nodes) has a direct impact on the VNF placement and chaining. The authors in [24] formulated the multi-objective VNF-PC problem considering both legacy Traffic Engineering (TE) ISP goals and combined TE-NFV goals.

Because of the inherent complexity of the VNF-PC problem, exact approaches based on ILP/MILP become impractical for realistic network sizes. Therefore, many heuristic-based algorithms have also been proposed to solve this problem in a reasonable time.

The problem of Elastic VNF Placement (EVNFP) was studied in [25] and an ILP model was presented for minimizing operational costs in NFV scenarios. The authors also developed an algorithm called Simple Lazy Facility Location (SLFL) in order to solve the EVNFP problem in polynomial time. Evaluations show that SLFL reduced operational costs by 5-8% and also increased the request acceptance rate by 2x as compared to the first-fit and random alternatives.

S. Sahhaf et al. studied the decomposition and embedding of network services in [26]. An ILP model was proposed whose objective was to minimize the total cost due to the mapping of different decomposed VNF components (e.g. VM, container, DPDK) to the physical nodes in NFVi. A heuristic algorithm, consisting of two phases– backtracking and mapping, was also proposed. The experimental results show a decrease in mapping cost and an increase in the request acceptance ratio in the long run for both ILP and heuristic approaches.

F. Carpio et al. studied the problem of network load balancing for the de-

ployment of Service Function Chains (SFCs) [27]. In particular, the authors addressed the problem of distance-to-data center by the use of VNF replicas in order to load balance the network. Three approaches: ILP model, Genetic Algorithm, and random fit placement algorithm were designed and compared to realize efficient VNF placement and replication method in an NFV environment.

Although a lot of resource allocation studies have been carried out in the past, only two studies have considered hardware accelerators in their models. H. Fan et. al. proposed an architecture to implement uniform deployment and allocation of accelerator resources in NFV environments [28]. The authors proposed an algorithm to achieve efficient allotment of accelerator resources in forwarding and server nodes. The algorithms take as an input the network topology and the capacity of physical resources and output the amount of accelerator resources that should be provided on the forwarding and server nodes. This study concerns the optimization of accelerators resource provisioning not with the optimization of accelerator allocation to VNFs.

The concept of heterogeneous components has been described in [29]. A heterogeneous service consists of multiple implementation options that could be deployed to serve the dynamic requirements of the service. The paper studied the problem of joint Scaling, Placement and Routing (SPRING) for heterogeneous services. To address the SPRING problem, a MILP formulation and a heuristic algorithm were proposed. The SPRING model focuses on efficient resource allocation in heterogeneous infrastructure with lower processing times. This paper does not consider the distribution of hardware accelerator resources in a data center. Furthermore, VNF-PC decisions did not take into account the communication between the hardware accelerator and the CPU on a server node.

This work is a major extension to our previous work where we only modeled VNF placement in a heterogeneous NFV environment [30] using a best-fit based approach. Here, we address the complete problem of accelerator-aware VNF placement and chaining along with a thorough evaluation of the ILP model and heuristics.

2.4 Problem Overview

Services in the NFV domain are realized by processing network traffic through a sequence of VNFs. In order to fully exploit the benefits of NFV technology, it is necessary to efficiently allocate NFVi resources to VNF-chains. Resource allocation requires a mapping of the service's VNF Forwarding Graph (VNF-FG) to NFVi resources [6]. A VNF-FG consists of

nodes representing VNFs and edges stand for virtual links between VNFs. Therefore, the mapping process can be thought of as a two steps process, namely (i) VNF placement and (ii) VNF Chaining. "VNF placement" involves the assignment of VNFs to COTS servers, whereas the "VNF chaining" step involves the allocation of a path in the physical network to every virtual link of VNF-FG. "VNF chaining" ensures the appropriate steering of network traffic through the sequence of VNFs constituting the service. Together this problem is referred to as the VNF placement and chaining (VNF-PC) problem.

In addition to the usual compute, network and storage resources, NFVi also includes hardware accelerator resources. With the inclusion of hardware accelerators in NFVi, VNF-PC models must be revised. In order to ensure efficient utilization of all NFVi resources, both placement and chaining decisions should take into account the accelerator resources (e.g. total logic elements and BRAM of FPGAs, cores/threads of GPUs) along with the usual NFVi resources, i.e. compute, storage and network. This problem will be referred to as the accelerator-aware VNF placement and chaining (VNF-AAPC) problem.

We motivate the importance of modeling the VNF-AAPC problem by a simple example illustrated in Fig 2.5. As an input, NFVi consists of five server nodes each with 5 CPU cores and connected with each other as shown in Fig 2.5. One of the server nodes is equipped with a hardware accelerator card connected over the PCIe bus. The objective of the VNF-PC problem is to deploy VNF-chains s_1 and s_2 using as few server nodes as possible. The CPU requirement of all VNFs is indicated in the boxes above each VNF. VNF f_{12} is an 'accelerate-able' VNF, i.e., it consumes 4 CPU units when it is not accelerated and 2 CPU units when it is able to offload its operations to an accelerator on a hardware accelerator card. For the sake of simplicity, we assume sufficient bandwidth is available on physical links for the chaining of VNFs. The result of the usual (accelerator-agnostic) VNF placement method, where only CPU resources are considered, is shown in Fig. 2.5 (a). In total, five server nodes are required for the deployment of s_1 and s_2. With the accelerator-aware strategy, however, only four server nodes are required for the placement of the same VNF-chains as shown in Fig 2.5 (b). This is because the VNF f_{12} is deployed on a server node attached with a hardware accelerator card and is able to reduce its CPU requirement by half.

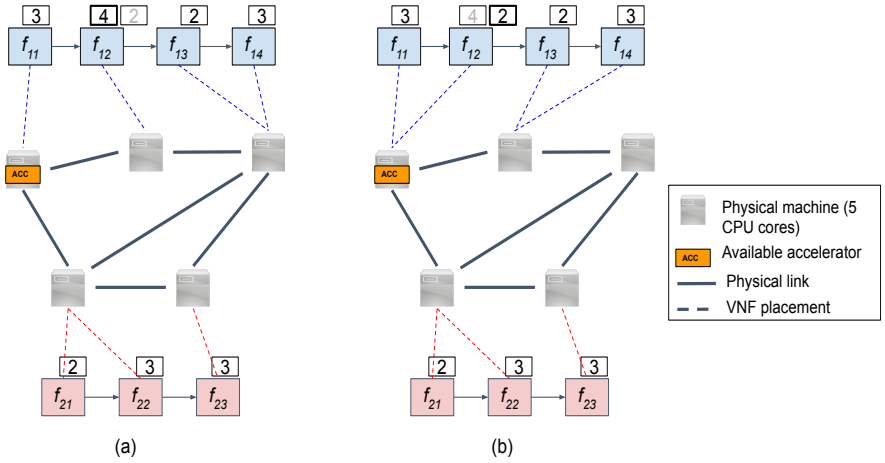


Figure 2.5: Illustration comparing VNF placement in accelerator-agnostic and accelerator-aware VNF placement scenarios. The CPU requirement of each VNF is indicated in the box above it.

2.5 ILP Formulation

Next, we introduce notations, decision variables, objective function and constraints required for the ILP formulation of the VNF-AAPC problem. The ILP model for the VNF-AAPC problem provides a single-step method for obtaining optimal resource allocation.

Table 2.2 gives the description of the notations used in the formulation. NFVi network is represented by a connected directed graph $G = (N, E)$. Set N consists of all the physical nodes in the NFVi network and E represents all the physical links between nodes. A node can be a computational device (e.g. COTS server) or a forwarding device (e.g. switch). $N^c \subset N$ denotes a set of all COTS servers having computational resources required to run VNFs.

The capacity of different resource types in a node $n \in N^c$ is denoted by the following three parameters: $\mathcal{R}_{cpu}(n)$, $\mathcal{R}_{bus}(n)$ and $\mathcal{R}_{acc}(n)$. $\mathcal{R}_{cpu}(n)$ denotes the total number of CPU cores available for running VNFs on node n.

The IO communication capacity of a node is dependent on the bandwidth (Mbps) of the PCI(e) bus which is denoted by $\mathcal{R}_{bus}(n)$. The same PCI(e) bus is shared for two tasks. First, for communication with accelerators and secondly for sending/receiving packets to/from the network using NIC card.

The amount of resources present on the hardware accelerator card attached to server node n is represented by $\mathcal{R}_{acc}(n)$. We use $\mathcal{R}_{acc}(n)$ only to denote the total number of logic elements present on an FPGA board. However, other resources like the amount of BRAM available on an FPGA can also be represented similarly.

A is a catalog of all types of accelerator implementations available for instantiation on the hardware accelerator cards attached to server nodes. For example, if FPGA bitfile implementations for only AES and SHA accelerators are available, i.e. $A = \{AES,\ SHA\}$, en/decryption (AES) or hashing (SHA) tasks from an IPSec VNF can be offloaded to AES and SHA accelerators running on an FPGA card. However, tasks like en/decoding involved in the vTC (video transcoding) VNF cannot be offloaded using any accelerator implementation present in A.

Each implementation of an accelerator type $a \in A$ requires a certain amount of resources, represented by $r(a)$, on the hardware accelerator card. Again, $r(a)$ can be used to represent the requirement of any type of resource on a hardware accelerator card. In our formulation, $r(a)$ denotes only the required number of logic elements to implement an accelerator of type a on an FPGA card.

We assume each service-request s received by the telecom operator consists of a VNF-FG $\mathcal{G}^s$ and corresponding bandwidth requirement (t_s). The VNF-FG $\mathcal{G}^s = (\mathcal{F}^s, \mathcal{L}^s)$ of the service-request s consists of a set of VNFs $\mathcal{F}^s$ and a set of virtual links between VNFs denoted by $\mathcal{L}^s$. Two consecutive VNFs of the service-request s denoted by f^s_k and f^s_{k+1} are joined by a virtual link $(f^s_k, f^s_{k+1}) \in \mathcal{L}^s$. For the sake of simplicity, we assume the traffic compression ratio of every VNF is 1. This implies that the amount of traffic (t_s) doesn't change while passing through a sequence of VNFs.

CPU requirement for a VNF $f^s \in \mathcal{F}^s$, in terms of the total number of cores required, is denoted by $cpu_0(f^s)$ and the reduction in the total number of cores due to offloading is denoted by $cpu_r(f^s)$. In other words, $cpu_0(f^s)$ denotes the number of CPU cores required by the VNF f^s to process the network traffic coming at the rate t_s.

The type of accelerator required for offloading a VNF f^s is denoted by $atype(f^s)$.

$\alpha^n_{f^s}$ is a binary variable used to indicate if VNF f of the service-request s is placed on node n. Allocation of an accelerator to VNF f^s placed on node n is indicated by $\beta^n_{f^s}$. A computational node $n \in N^c$ is said to be in use if at least one VNF is placed on n. This is denoted by a binary variable x_n. Instantiation of an accelerator of type a on the hardware accelerator card attached to n is indicated by a binary variable δ^n_a.

The binary variable $\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}}$ is an indicator variable which denotes if the

virtual link $(f_k^s, f_{k+1}^s) \in \mathcal{L}^s$ mapping to a path in G contains physical-link (n_i, n_j) or not.
In a scenario when a telecom operator leases server nodes from an Infrastructure Provider (InP) to deploy VNF-chains, she ought to acquire a minimum number of server nodes as possible. The cost of a server node is included in the total cost if that node is used to host at least one VNF. The cost of using a computational node $n \in N^c$ is denoted by C_n (in \$). Usually, parameters like $\mathcal{R}_{cpu}(n)$, $\mathcal{R}_{bus}(n)$ and $\mathcal{R}_{acc}(n)$ determine the value of c_n.

Next, we discuss the objective function and constraints describing the ILP model for the accelerator-aware VNF placement and chaining problem.

2.5.1 Objective

The objective (2.1) of our ILP formulation is to minimize the total cost incurred to the operator from the use of server nodes, some of which are attached to a hardware accelerator card. The decision variable x_n is used to determine whether a server node is used or not.

$$obj : \min \big(\sum_{n \in N^c} c_n x_n \big) \tag{2.1}$$

2.5.2 Constraints

We classify all the constraints in four categories: (i) Physical node constraints, (ii) Link Mapping constraints, (iii) Accelerator Constraints and (iv) Auxiliary Constraints, which are explained as follows.

2.5.2.1 Physical Node Constraints

The sum of effective CPU usage of all VNFs placed on any node should not surpass its maximum CPU capacity. This constraint in depicted in (2.2).
The constraint in (2.3) indicates the finite availability of resources on the hardware accelerator card for the instantiation of accelerators.
The rate of communication between VNFs and accelerators instantiated on the hardware accelerator card is bounded by the maximum bandwidth of the PCIe bus, as indicated in (2.4). The first term in the LHS of (2.4) is the bus bandwidth consumption due to the traffic between neighboring VNFs. First, summation over the traffic coming from VNFs (f_k^s) placed on server node n_i to its neighboring VNFs (f_{k+1}^s) placed on n_j is carried out and a factor of two is there to represent the traffic both coming to and from the VNFs running on server node n_i. The term $2t_s\beta_{f^s}^{n_i}$ represents the

Table 2.2: Description of parameters and decision variables

Input parameters	
Notation	Description
G	Directed graph $G = (N, E)$ represents the network.
N	Set of all forwarding and computational nodes within the network.
N^c	Set $N^c \subset N$ contains all nodes of the network with positive computational resources (all server nodes).
$b(n_i, n_j)$	Maximum bandwidth (in Mbps) of a physical-link $(n_i, n_j) \in E$. **
$\mathcal{R}_{cpu}(n)$	Maximum CPU resources (in total number of CPU cores) available on $n \in N^c$.
$\mathcal{R}_{acc}(n)$	Maximum accelerator-fabric resources (in total number of logic elements) available on $n \in N^c$.
$\mathcal{R}_{bus}(n)$	Maximum bandwidth (in Mbps) of the PCIe bus of node $n \in N^c$.
A	Set of all available accelerator types (in NFVi).
$r(a)$	Resource requirement (logic elements) of the accelerator type $a \in A$.
S	Set of all VNF-chains.
$\mathcal{G}^s$	Directed graph $\mathcal{G}^s = (\mathcal{F}^s, \mathcal{L}^s)$ represents VNF-FG of request $s \in S$.
$\mathcal{F}^s$	Set of all VNFs in VNF-FG of the VNF-chain $s \in S$.
$\mathcal{L}^s$	Set of all directed virtual links in the VNF-FG of the VNF-chain $s \in S$.
t_s	Throughput requirement (Mbps) of the VNF-chain $s \in S$.
$cpu_0(f^s)$	CPU requirement (cores) of VNF $f \in \mathcal{F}^s$.
$cpu_r(f^s)$	CPU reduction (cores) for VNF $f \in \mathcal{F}^s$.
$atype(f^s)$	Type of accelerator needed for acceleration of VNF $f\mathcal{F}^s$.
c_n	Cost ($) of running a computational node $n \in N^c$.
Decision variables	
Notation	Description
$\alpha^n_{f^s}$	Binary variable indicates if VNF f^s of VNF-chain s is placed on n.
$\beta^n_{f^s}$	Binary variable indicates if VNF f^s of VNF-chain s is accelerated on n.
x_n	Binary variable indicates if computational node $n \in N^c$ is used for hosting at-least one VNF.
δ^n_a	Binary variable indicates if accelerator of type a is instantiated on the node n
$\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}}$	Binary variable indicates if the virtual link (f^s_k, f^s_{k+1}) mapping to a path in the physical-network contains the physical-link (n_i, n_j), $(n_i, n_j) \in E$.

bandwidth utilization due to communication between the VNF f^s and accelerator fabric on the node n.

$$\sum_{s\in S, f^s\in\mathcal{F}^s} \alpha^n_{f^s} cpu_0(f^s) - \beta^n_{f^s} cpu_r(f^s) \leq \mathcal{R}_{cpu}(n) \quad \forall n \in N^c \tag{2.2}$$

$$\sum_{a\in A} r(a)\delta^n_a \leq \mathcal{R}_{acc}(n) \quad \forall n \in N^c \tag{2.3}$$

$$\sum_{\substack{\forall n_j\in N \\ ((n_i,n_j)\in E)}} \sum_{\substack{s\in S, \\ (f^s_k,f^s_{k+1})\in\mathcal{L}^s}} 2t_s\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}} + \sum_{\substack{s\in S, \\ f^s\in\mathcal{F}^s}} 2t_s\beta^{n_i}_{f^s} \leq \mathcal{R}_{bus}(n_i) \quad \forall n_i \in N^c \tag{2.4}$$

2.5.2.2 Physical link constraints

The flow-conservation constraint is described in (2.5). This constraint ensures that a virtual link (f^s_k, f^s_{k+1}) is always mapped to a physical path in the network. Also, it ensures that for a non-computation node $n \in N \setminus N^c$, the net-traffic outflow or inflow is always zero.
The constraint in (2.6) guarantees that the sum of bandwidths allocated to virtual links on a physical-link (n_i, n_j) never exceeds its capacity $b(n_i, n_j)$.

$$\sum_{\substack{\forall n_j\in N \\ ((n_i,n_j)\in E)}} (\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}} - \gamma^{n_j,n_i}_{f^s_k,f^s_{k+1}}) = (\alpha^{n_i}_{f^s_k} - \alpha^{n_i}_{f^s_{k+1}}) \quad \forall s \in S, \forall (f^s_k, f^s_{k+1}) \in \mathcal{L}^s, \forall n_i \in N \tag{2.5}$$

$$\sum_{\substack{s\in S \\ (f^s_k,f^s_{k+1})\in\mathcal{L}^s}} t_s\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}} \leq b(n_i, n_j) \quad \forall (n_i, n_j) \in E \tag{2.6}$$

2.5.2.3 Accelerator constraints

The constraint in (2.7) is a consequence of the fact that a VNF f^s can be given access to an accelerator on a node n only if it is placed on it.
The constraint in (2.8) ensures that an accelerator of a particular type is instantiated if a non-zero number of VNFs are using that accelerator type. This constraint is easily linearized by replacing it with a pair of constraints indicated in (2.9a-2.9b). M_1 (big M) in constraint (2.9b) is a constant with a value greater than the total number of VNFs f^s in all the service-chain requests $s \in S$.

$$\beta^n_{f^s} \leq \alpha^n_{f^s} \quad \forall n \in N, \forall s \in S, \forall f^s \in \mathcal{F}^s \tag{2.7}$$

$$\delta_a^n = \begin{cases} 1, & \text{if } \sum\limits_{\substack{\forall s \in S, \forall f^s \in \mathcal{F}^s, \\ a = atype(f^s)}} \beta_{f^s}^n \geq 1 \\ 0, & \text{otherwise} \end{cases} \qquad \forall n \in N, \forall a \in A \tag{2.8}$$

$$\delta_a^n \leq \sum_{\substack{\forall s \in S, \forall f^s \in \mathcal{F}^s \\ a = atype(f^s)}} \beta_{f^s}^n \quad \forall n \in N, \forall a \in A \tag{2.9a}$$

$$\sum_{\substack{\forall s \in S, \forall f^s \in \mathcal{F}^s \\ a = atype(f^s)}} \beta_{f^s}^n \leq M_1 \delta_a^n \quad \forall n \in N, \forall a \in A \tag{2.9b}$$

2.5.2.4 Auxiliary Constraints

The set of constraints in this subsection restrict the value of decision variables x_n, $\alpha_{f^s}^n$, β_{f^s}, δ_a^n, $\gamma_{f_k^s, f_{k+1}^s}^{n_i, n_j}$.
A server node is considered to be running if at least one VNF is mapped onto it, as indicated by the constraint in (2.10). The pair of constraints (2.11a - 2.11b) forces x_n to be equal to 1 if at least one VNF is placed on node n. In constraint 2.11b, M_2 is a constant with a value greater than the total number of VNFs f^s in all service-chain requests $s \in S$.

$$x_n = \begin{cases} 1, & \text{if } \sum\limits_{\forall s \in S, \forall f^s \in \mathcal{F}^s} \alpha_{f^s}^n \geq 1 \\ 0, & \text{otherwise} \end{cases} \qquad \forall n \in N \tag{2.10}$$

$$x_n \leq \sum_{s \in S, f^s \in \mathcal{F}^s} \alpha_{f^s}^n \quad \forall n \in N \tag{2.11a}$$

$$\sum_{s \in S, f^s \in \mathcal{F}^s} \alpha_{f^s}^n \leq M_2 x_n \quad \forall n \in N \tag{2.11b}$$

Each VNF in a service-chain request must be placed only once. This is represented by the constraint in (2.12). The set of constraint in (2.13) ensures decision variables $\alpha_{f^s}^n, \beta_{f^s}^n, \delta_a^n$ and $\gamma_{f_k^s, f_{k+1}^s}^{n_i, n_j}$ can only take binary (0 or 1) values.

$$\sum_{n \in N} \alpha_{f^s}^n = 1 \quad \forall s \in S, \forall f^s \in \mathcal{F}^s \tag{2.12}$$

$$\begin{gathered} x_n, \alpha_{f^s}^n, \beta_{f^s}^n, \delta_a^n, \gamma_{f_k^s, f_{k+1}^s}^{n_i, n_j} \in \{0, 1\} \\ \forall n \in N, \forall (n_i, n_j) \in E, \forall s \in S, \forall f^s \in \mathcal{F}^s, \forall (f_k^s, f_{k+1}^s) \in \mathcal{L}^s \end{gathered} \tag{2.13}$$

The above ILP formulation implements a single-step method to solve the VNF-AAPC problem. For a given NFVi graph G and a set of requested

service-chain requests S, the above ILP formulation not only gives an optimum VNF placement $\alpha^n_{f^s}$ and chaining $\gamma^{n_i,n_j}_{f^s_k,f^s_{k+1}}$ solution but also gives an optimum accelerator allocation $\beta^n_{f^s}$ for VNFs. As the VNF-PC is considered to be an NP-hard problem, it does not scale with the problem size. The accelerator awareness further increases its complexity. As a result, the ILP formulation of the VNF-AAPC problem is challenging to solve for the networks of realistic sizes. In order to address the non-scalability issue with ILP, we propose a method based on heuristics for solving the VNF-AAPC problem in a time-efficient manner.

2.6 Proposed Heuristics

Next, we describe two heuristic-based algorithms for solving the VNF-PC problem for NFVi containing hardware accelerator resources along with the usual resources. The first heuristic we propose is an accelerator-agnostic algorithm that does not take into account the presence of hardware accelerators in NFVi while performing VNF-FG mapping. This algorithm will serve as a baseline for the evaluation of our second algorithm, i.e., accelerator-aware VNF-PC heuristic.

2.6.1 Accelerator-agnostic VNF-PC heuristic

The accelerator-agnostic VNF-PC heuristic involves a hierarchical deployment of VNF-chains [31]. Hierarchical deployment exploits the classification of nodes into different levels of DC topologies, e.g. different levels in a leaf-spine DC topology are server, rack, and cluster. Starting from the lowest level, i.e. server node level, VNF-PC is attempted at each level until the VNF-chain is deployed at a level. Also, the previously used server node is checked for the placement of subsequent VNF of a VNF-chain resulting in the localization of VNFs of the same VNF-chain.
The pseudo-code for the accelerator-agnostic VNF-PC algorithm is described in Alg. 2.1. The procedure `AgPlaceChain` is called from Alg. 2.2 in order to map VNF-FG ($G^s = (\mathcal{F}^s, \mathcal{L}^s)$) corresponding to all service-requests (line 3) onto NFVi. The mapping of each VNF-FG is attempted at different levels of the data center, e.g., in leaf-spine topology, first at *node* level, then at *rack* level, and at last at the *cluster* level (Alg. 2.2). Mapping on *node* level is done by assigning $NodesSet$ equal to the set of all server nodes $\forall n \in N^c$. If no one server node is able to allocate all the VNFs of a chain, $NodesSet$ is assigned all nodes per *rack*. If the mapping of a VNF-chain is not possible to any of the *rack*, $NodesSet$ is assigned all the nodes in the *cluster* and

VNF-PC is attempted again.
In Alg. 2.1, for each VNF $f^s \in \mathcal{F}^s$ placement is first tried on the previously used node n_p. A new node is only selected if enough CPU resources aren't available on n_p (line 7-13). An attempt for accelerator allocation is done on node n_p (line 14) by invoking procedure `AccelVNF`. When all VNFs $\forall f^s \in \mathcal{F}^s$ are placed, virtual-links are mapped to physical-paths in G using the procedure `ChainVNFs`. If the placement of any VNF $f^s \in \mathcal{F}^s$ fails or procedure `ChainVNFs` returns **False**, all resources are updated to their previous values just after the start of the procedure `AgPlaceChain` (lines 23-25).
The procedure `AccelVNF` (Alg. 2.3) checks whether an accelerator can be granted to VNF f^s node n_p. This is done by verifying whether enough CPU and bus resources are available on n_p (line 2). If $atype(f^s)$ is not already instantiated on the hardware accelerator card attached to node n_p, it is checked whether enough accelerator resources are available on the card (lines 5-10) to instantiate the accelerator type $atype(f^s)$. All the required resources are updated accordingly if f^s is allocated an accelerator (lines 9-13) in this procedure.
The chaining procedure for mapping of virtual links to physical paths is described in Alg. 2.4. For each virtual link $(f^s_k, f^s_{k+1}) \in \mathcal{L}^s$, all set of shortest paths between two physical nodes hosting f^s_k and f^s_{k+1} are stored first in P (line 7). Each path is checked sequentially for its available bandwidth on all of its physical links using the procedure `bw` (lines 9). If a path with enough bandwidth is available, $\gamma[f^s_k, f^s_{k+1}]$ (line 13) along with bus (line 12) and link bandwidths (line 11) are updated for every physical link $(\forall(n_i, n_j) \in p)$ in the path p. If any virtual link cannot be mapped to a physical path, the variables values are reverted to their previous values at the start of the procedure (lines 16-19).
An example illustrating the working of the accelerator-agnostic VNF-PC heuristic is shown in Fig. 2.6. Consider two VNF-chains ($s_1 : f_{11} \rightarrow f_{12} \rightarrow f_{13} \rightarrow f_{14}$, $s_2 : f_{21} \rightarrow f_{22} \rightarrow f_{23}$) supposed to be deployed on a given NFVi, which here is a DC in the leaf-spine topology. The heuristic starts with the chain s_1 and first tries its deployment on the server node level. As no server node can accommodate all VNFs of this VNF-chain, the heuristic moves to the next level of the topology, i.e., rack level. Again, no rack has enough resources to host the complete VNF-chain s_1. Therefore, the heuristic now considers all server nodes of the cluster and uses $rack0$ and $rack1$ for the placement of VNF-chain s_1. After the placement of all VNFs of the first chain is completed, network bandwidth is then allocated to the virtual links of the VNF-chain via the `ChainVNFs` procedure as shown in Fig. 2.6. The same process will be followed for the deployment of the VNF-chain s_2 which

is deployed in $rack2$.

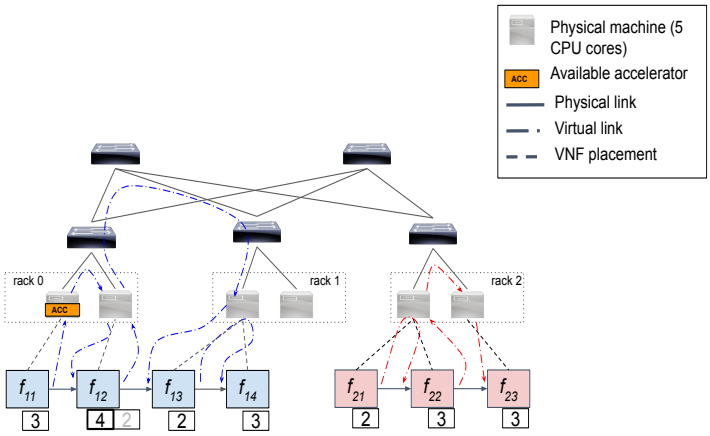


Figure 2.6: Illustration showing placement and chaining in accelerator-agnostic VNF-PC heuristic on the leaf-spine topology.

2.6.2 Accelerator-aware VNF-PC heuristic

The accelerator-aware VNF-PC heuristic combines hierarchical deployment with the segmentation of VNF-chains. A VNF-chain is first split at 'accelerate-able' VNFs, i.e. VNFs which require hardware accelerators $\forall s, \forall f^s, atype(f^s) \in A$. VNF-chain deployment is then performed in two phases. In the first phase, VNF placement along with accelerator-allocation is performed for all accelerate-able VNFs. The sub-graphs (VNF-FGs) corresponding to the remaining VNF-chain segments are mapped to the NFVi using the hierarchical deployment in the second phase.

The procedure for allocation of accelerators to VNFs (`PlaceAccelVNFs`) is shown in Alg. 2.5. First, a list of all server nodes attached with a hardware accelerator card is stored in N_a. Accelerate-able VNFs constituting the VNF-chain request s are assigned to F^s_{acc}. Accelerator allocation is then attempted for every VNF in F^s_{acc}. A list of server nodes with enough resources and having accelerator $atype(f^s)$ already instantiated on its attached hardware accelerator card is stored in N_{f^s} (line 6). Out of all server nodes in N_{f^s}, a node, the closest node where (any) previous VNF of the same VNF-chain request s was placed, is assigned to n_a (line 9). If no node has sufficient resources and accelerator of type $atype(f^s)$ instantiated on it, a node with the highest CPU utilization is then selected from N_a (line

Algorithm 2.1: Accelerator-agnostic VNF-PC procedure.

```
1  Procedure AgPlaceChain(NodesSet, α, β, γ, (F^s, L^s)):
2      tries, plc, n_p ← 0, True, φ
3      while tries ≤ MAX_TRIES do
4          for Nodes in NodesSet do
5              α_0, β_0, Nodes_0 ← α, β, Nodes
6              for f^s in F^s do
7                  if n_p == φ or (cpu_0(f^s)) > R_cpu(n_p)) then
8                      N = {n : ∀n ∈ Nodes, R_cpu(n) ≥ (cpu_0(f^s)}
9                      if N == φ then
10                         plc ← False
11                         break
12                     else
13                         n_p ← Random(N)
14                 if AccelVNF(f^s, n_p, node_accels, t_s) == False then
15                     R_cpu(n_p) ← R_cpu(n_p) − cpu_0(f^s)
16                 else
17                     β[f^s] ← n_p
18                 α[f^s] ← n_p
19             if ChainVNFs(α, γ, L^s, G) == False or plc == False then
20                 α, β, Nodes ← α_0, β_0, Nodes_0
21             else
22                 return True
23         tries ← tries + 1
24 end
```

Algorithm 2.2: Main service-chain allocation procedure.

```
1  Procedure AllocateChain(G, N^c, racks, cluster, G^s, α, β, γ):
2      for NodesSet in {{{n} : n ∈ N^c}, racks, clusters} do
3          if AgPlaceChain(NodesSet, α, β, γ, G^s) then
4              break
5  end
```

Algorithm 2.3: VNF acceleration procedure

```
1  Procedure AccelVNF(f^s, n_p, node_accels, t_s):
2      if atype(f^s) ∉ A or R_cpu(n_p) < (cpu_0(f^s) − cpu_r(f^s)) or
         R_pci(n_p) < 2t_s then
3          return False
4      else
5          if atype(f^s) ∉ node_accels[n_p] then
6              if r(atype(f^s)) > R_acc(n_p) then
7                  return False
8              else
9                  R_acc ← R_acc − r(atype(f^s))
10                 node_accels[n] ← node_accels[n] ∪ {atype(f^s)}
11         R_cpu(n) ← R_cpu(n) − (cpu_0(f^s) − cpu_r(f^s))
12         R_bus(n) ← R_bus(n) − 2t_s
13         return True
14 end
```

Algorithm 2.4: VNF chaining procedure

```
1  Procedure ChainVNFs(α, γ, L^s, G):
2      G_0(N_0, E_0) ← G(N, E)
3      γ_0 ← γ
4      for (f_k^s, f_{k+1}^s) in L^s do
5          done ← False
6          if α[f_k^s] ≠ α[f_{k+1}^s] then
7              P ← ShortestPaths(G, α[f_k^s], α[f_{k+1}^s])*
8              for p in P do
9                  if bw(p) >= t_s* then
10                     for (n_i, n_j) in p do
11                         b(n_i, n_j) ← b(n_i, n_j) − t_s*
12                         R_bus(n_i) ← R_bus(n_i) - 2t_s
13                         γ[f_k^s, f_{k+1}^s] ← (n_i, n_j)
14                         done ← True
15                     break
16             if done == False then
17                 G(N, E) ← G_0(N_0, E_0)
18                 γ ← γ_0
19                 return False
20     return True
21 end
```

11). Using the procedure `AccelVNF` (Alg. 2.3), placement and accelerator allocation for VNF f^s is attempted on n_a (line 12).
In Alg. 2.6, each server node used in the previous step is iterated over for the complete mapping of the remaining VNF-chain segments (lines 4-20). The unmapped segments of all service requests are identified for which at least one adjacent VNF is placed on node n (lines 5-8). An attempt is then made to map each segment seg in S^n_{seg} with as much proximity to n as possible. The process followed for the mapping of each VNF-FG segment $\forall seg \in S^n_{seg}$ is similar to the one followed in the accelerator-agnostic VNF-PC heuristic (Alg. 2.2). The mapping is attempted first on the *nodel* level, then on the *rack* level containing *node* and at last on the whole *cluster* level using the procedure `AgPlaceChain` (line 13). In addition, newly placed VNF segment seg and its adjacent VNFs, which were previously placed using `PlaceAccelVNFs`, are linked via procedure `ChainVNFs` (line 18).
At last, VNF-chain requests which haven't been yet mapped to NFVi are identified (line 21). Set S_R contains all those VNF-chain requests $s \in S$ which either (i) do not have any VNF with an accelerator implementation available in A or (ii) enough resources were not available to allocate accelerator to VNFs during the first step (line 2). Mapping of all service requests in S_R is attempted in a hierarchical way (lines 22-25) discussed in Alg. 2.2.
Again, consider the deployment of two VNF-chains ($s_1 : f_{11} \rightarrow f_{12} \rightarrow f_{13} \rightarrow f_{14}$, $s_2 : f_{21} \rightarrow f_{22} \rightarrow f_{23}$) on the same NFVi topology as shown in Fig. 2.7. In accelerator-aware PC heuristic, accelerate-able VNFs of two VNF-chains are placed in the first phase, so f_{12} is placed on the first server node of $rack0$ which has an attached hardware accelerator card. In the second phase, the heuristic loops over the server nodes which have VNFs placed on them, while determining and placing the remaining segments of VNF-chains. Therefore, deployment of the VNF-chain segment f_{11} and $f_{13} \rightarrow f_{14}$ is then attempted using the same procedure as discussed in accelerator-agnostic heuristic. After the successful deployment of VNF-chain segments of s_1, two segments f_{11}, $f_{13} \rightarrow f_{14}$ are chained to the VNF f_{12} via `ChainVNFs` procedure. At last, the second VNF-chain (no accelerate-able VNFs) s_2 is then deployed in $rack1$ using the same procedure as followed in the accelerator-agnostic heuristic. It can be observed that the accelerator-aware VNF-PC heuristic results in using one less server node as compared to the accelerator-agnostic heuristic for the deployment of VNF-chains s_1 and s_2.

Algorithm 2.5: Placement procedure for accelerate-able VNFs.

Procedure `PlaceAccelVNFs`(α, β, γ, *node_accels*, S)**:**
  $N_a \leftarrow \{n \in N^c : \mathcal{R}_{acc}(n) > 0\}$
  for s **in** S **do**
    $F^s_{acc} \leftarrow \{f^s \in \mathcal{F}^s : atype(f^s) \in A\}$
    for f^s **in** F_{acc} **do**
      $N_{f^s} \leftarrow \{n \in N^c : atype(f^s) \in node\_accels[n], 2t_s < \mathcal{R}_{bus}, \big(cpu_0(f^s) - cpu_r(f^s)\big) < \mathcal{R}_{cpu}(n)\}$
      if $N_{f^s} \neq \phi$ **then**
        $n_p \leftarrow$ select node from N_{f^s} where previous VNF of the service chain s was placed
        $n_a \leftarrow \arg\min_n$ `PathLen`(n, n_p)
      else
        $n_a \leftarrow$ select a node from N_a with sufficient resources
      if `AccelVNF`(f^s, n_a, *node_accels*, t_s) **then**
        $\alpha[f^s], \beta[f^s] \leftarrow n_a, n_a$
  used_nodes $\leftarrow$ all used nodes in N
  return *used_nodes*
end

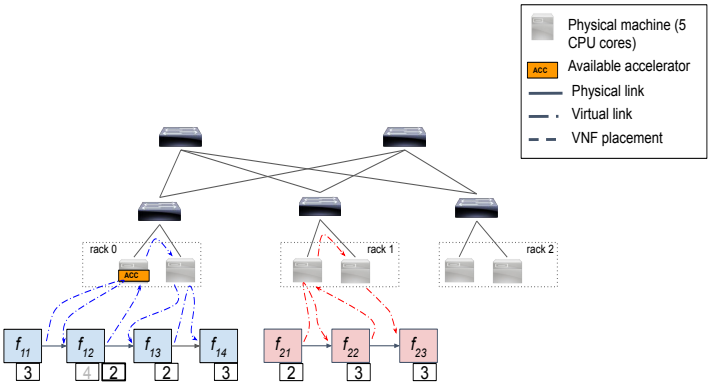


Figure 2.7: Illustration showing placement and chaining in accelerator-aware VNF-PC heuristic on the leaf-spine topology.

2.7 Performance evaluation

The objective of this section is to assess the scalability and efficiency of the ILP model and VNF-PC heuristics using simulation experiments. We first describe the simulation environment used in our evaluation and then present the obtained results after performing experiments on the ILP model

Algorithm 2.6: Accelerator-aware VNF-PC procedure.

Procedure `AccelAwarePlaceChain(`α, β, γ, S`):`
  $used\_nodes \leftarrow$ `PlaceAccelVNFs(`α, β, S`)`
  $S^p_{seg} \leftarrow \{\}$
  for n **in** $used\_nodes$ **do**
    $node\_chains \leftarrow \{s \in S : \exists f^s \in \mathcal{F}^s$ placed on $node\}$
    $S^n_{seg} \leftarrow \{\}$
    for $chain$ **in** $node\_chains$ **do**
      $S^n_{seg} \leftarrow S^n_{seg} \cup$ {all possible chain segments in $chain$};
    `/* place remaining segments of chains */`
    for seg **in** S^n_{seg} **do**
      $\mathcal{G}^{seg} \leftarrow$ VNF forwarding sub-graph corresponding to seg
      if $seg \not\subset S^p_{seg}$ **then**
        for $NodesSet$ **in** $\{\{node\}, rack\_node, cluster\_node\}$ **do**
          if `AgPlaceChain(`$\{NodesSet\}$, α, β, γ, $\mathcal{G}^{seg}$`)` **then**
            $f^{seg}_l \leftarrow$ leftmost VNF of seg
            $f^{acc}_l \leftarrow$ VNF which needs to be linked with the leftmost VNF of seg
            $f^{seg}_r \leftarrow$ rightmost VNF of seg
            $f^{acc}_r \leftarrow$ VNF which needs to be linked with the rightmost VNF of seg
            if `ChainVNFs(`α, γ, $\{(f^{acc}_l, f^{seg}_l)\}$, G`)` **and** `ChainVNFs(`α, γ, $\{(f^{seg}_r, f^{acc}_r)\}$, G`)` **then**
              $S^p_{seg} \leftarrow S^p_{seg} \cup \{seg\}$
              break
  $S_R \leftarrow S \setminus \{s \in S : \alpha[f^s] \neq \phi, \forall f^s \in \mathcal{F}^s\}$
  `/* placement and chaining of remaining service-chains */`
  for s **in** S_R **do**
    for $NodesSet$ **in** $\big\{\{\{n\} : n \in N^c\}, racks, clusters\big\}$ **do**
      if `AgPlaceChain(`$NodesSet, \alpha, \beta, \gamma, \mathcal{G}^s$`)` **then**
        break
end

and heuristics.

2.7.1 Setup and Parameters

The ILP model for the VNF-AAPC problem has been built using the Python API of IBM's ILOG CPLEX called DOcplex (Decision Optimization CPLEX Modeling). DOcplex provides a user-friendly API to write the ILP model which is then solved by the CPLEX solver. All heuristic algorithms are written in the Python programming language. We used an Intel

Table 2.3: Default values/range of various parameters involved in simulation experiments.

Parameter	Value or range	Parameter	Value or range		
$\mid S \mid$	[5, 250]	$c_o(f^c)$	3-5 (cores)		
$R_{cpu}(n)$	24 (cores)	$c_i(f^c)$	(0.40 - 0.60)$c_o(f^c)$		
$R_{acc}(n)$	0,100k (LUTs)	ρ^{vnf}_{acc}, ρ^{n}_{acc}	0.20, 0.30		
$R_{bus}(n)$	80 (Gbps)	$b(n_i, n_j)$	10, 40 (Gbps)		
VNF-chain length	4-6	accel. type (a)	a_1	a_2	a_3
t_s	100-500 (Mbps)	$r(a)$ (LUTs)	40k	28k	30k
c_n	1, 1.20				

Xeon server machine with quad-core CPU @ 2.40GHz with 16GB of RAM memory running Ubuntu-16.04 OS to carry out evaluations of the ILP and heuristics. Each data point reported in the evaluations indicates an average over 10 iterations along with the corresponding confidence interval of one standard deviation (68%).

For evaluation of heuristics, we have considered two different DC topologies for simulating the physical network: (i) three-tier and (ii) leaf-spine. For the three-tier topology, we vary the value of k to adjust the size of the network. For example, when k=6 we will have k=6 pods, each pod containing $k/2$ =3 access switches and $k/2$ = 3 aggregate switches. Each access switch (ToR switch) is connected to 6/2=3 server nodes and therefore the total number of server nodes in all the pods is equal to 54. For the leaf spine topology, we have considered 4 core switches and 16 leaf switches (ToR switch). Each leaf switch is connected to 16 server nodes, therefore, resulting in a total of 320 server nodes. In both the topologies, the links connecting server nodes with ToR switches are 10Gbps, whereas the links connecting switches are 40Gbps.

Each server node has 24 CPU cores, 16GB/s of PCIe bandwidth, and has 100k LUTs if a hardware accelerator card is attached to the server node. For simplicity we assume the cost c_n of a server node to be 1.20$ if it is attached with a hardware accelerator, otherwise 1$. The fraction of VNFs which are accelerate-able (ρ^{vnf}_{acc}) and the fraction of server nodes attached with hardware accelerator (ρ^{n}_{acc}) are set to 0.20 and 0.30, respectively, before generating VNF-chains for the simulation. The other parameters considered in evaluations are given in Table 2.3.

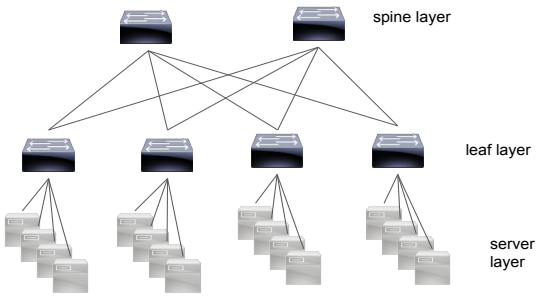


Figure 2.8: Leaf-spine topology used for the evaluation of the ILP approach and the heuristic.

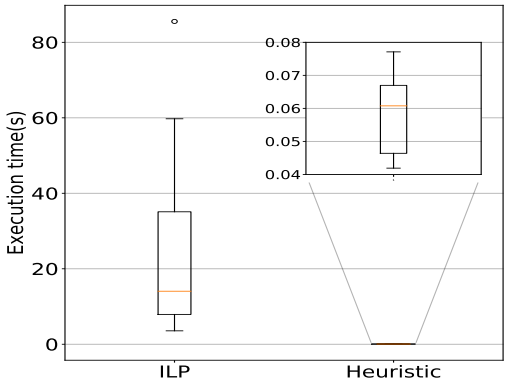


Figure 2.9: Comparison of the ILP model and the heuristic in terms of total execution times for the leaf-spine topology.

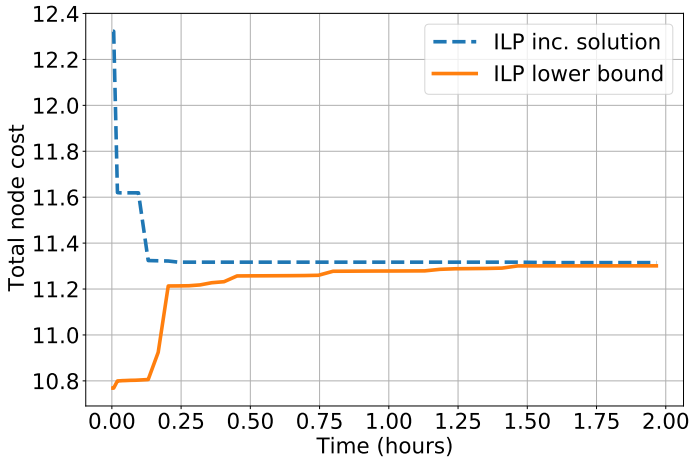


Figure 2.10: Evolution of ILP's incumbent solution and lower-bound for the full execution of CPLEX.

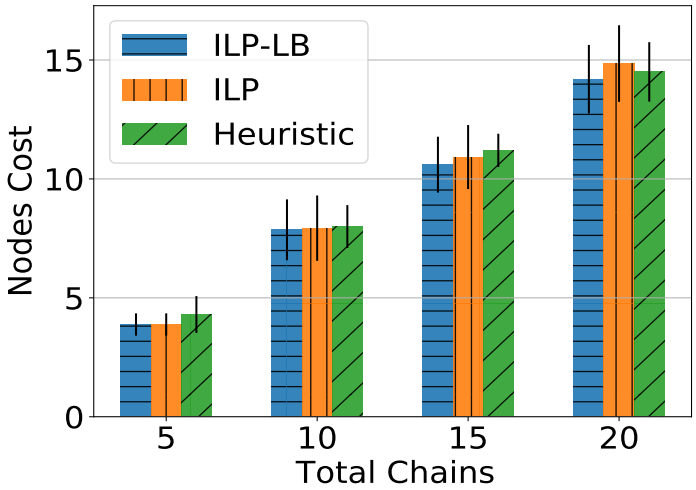


Figure 2.11: Comparison of ILP model and heuristic in terms of total node costs in the leaf-spine topology for different number of VNF-chains.

2.7.1.1 Comparison of ILP and Heuristic

Before presenting evaluation results regarding total node costs, we first report total execution times for the ILP and heuristic approach. Fig. 2.9 shows the distribution of total execution times for both approaches when deploying 5 VNF-chains on a leaf-spine topology shown in Fig. 2.8. As expected, it can be observed that the execution time of the ILP approach is orders of magnitude larger than the heuristic approach. Moreover, when the number of VNF-chains to be deployed on the given topology becomes large $\mid S \mid \geq 15$ the total execution time could reach up to several hours.
Fig. 2.10 shows the evolution of CPLEX solutions with time for the deployment of 15 VNF-chains. It can be observed that CPLEX takes about 2 hours to complete the execution for this instance, yet the gap between the incumbent solution and the lower-bound estimated by CPLEX after one hour is negligible (0.5%). Nevertheless, only small sizes instances of the VNF-AAPC problem can be solved using the ILP approach in a reasonable time.
Fig. 2.11 gives the comparison between ILP and heuristic in terms of total nodes cost for the deployment of different numbers of VNF-chains for a given NFVi. Here we have limited the maximum execution time of CPEX instances to one hour. The bar chart shows (i) ILP incumbent solution (ILP) and (ii) best lower-bound (ILP-LB) estimated by CPLEX until one hour and (iii) VNF-AAPC heuristic solution. We can observe that there exists a small penalty (on average $\sim$5%) when using the heuristic approach instead of the ILP approach. As mentioned earlier, the gap between ILP-LB and ILP is almost negligible after one hour of CPLEX execution time.
For the deployment of 20 VNF chains, we can observe that the cost of the total nodes using the ILP approach is higher than with the heuristic method. As the total time of execution is limited, CPLEX was not able to reach the optimal solution in the given time and the VNF-AAPC heuristic method is able to achieve more efficient allocation than the ILP approach. Although CPLEX can find the optimal solution if allowed to run without any time limitation, the performance of the heuristic is still very close to the estimated lower bound by CPLEX. As mentioned earlier, it will be impracticable to use ILP to solve problem instances of a size larger than 15 VNF-chains.

2.7.1.2 VNF-PC Heuristic Comparison

Here, we compare the performance of heuristic algorithms among themselves in terms of the three performance metrics. The following metrics were evaluated when deploying a given set of VNF-chains on a fixed NFVi

infrastructure with sufficient resources.

1. First, the total node cost is the cost of using (switching-on) server nodes, which also includes additional costs due to the installation of hardware accelerators in a subset of server nodes. The comparison of node costs will indicate the resulting cost saving in NFVi by using a particular VNF-PC scheme.

2. Second, β/α is the ratio of total VNFs allocated hardware accelerators to the total VNFs in all VNF-chains. It is possible that the VNF-PC algorithm might not allocate an accelerator to an accelerate-able VNF. This metric shows the efficiency of the VNF-PC algorithm in terms of the utilization of hardware accelerator resources. A higher value of β/α indicates efficient allocation of hardware accelerator resources by the VNF-PC algorithm.

3. Third, CPU_{rem} is the average amount of CPU cores remaining per server node left unallocated after the completion of VNF-PC. A high CPU_{rem} indicates the poor consolidation of VNFs, thereby resulting in overall inefficient allocation of resources.

To benchmark the performance of the proposed VNF-AAPC heuristic, we also evaluate the performance of the accelerator-agnostic VNF-PC heuristic. The accelerator-agnostic VNF-PC heuristic will serve as the baseline for the evaluation of our VNF-AAPC heuristic.
Fig. 2.12 (a) and (d) show the total node costs incurred to the operator as a result of the deployment of different numbers of VNF-chains on three-tier and leaf-spine DC topologies, respectively. In both topologies, the results show the lowest resource cost in the case of accelerator-aware VNF-PC heuristic. This arises from the efficient consolidation of VNFs, as explained below, by the accelerator-aware heuristic in contrast to the poor VNF consolidation in accelerator-agnostic heuristic.
For the accelerator-agnostic VNF-PC, the VNF placement process is unaware of the presence of accelerator resources on a server node. The chance of an accelerator being allocated to a VNF depends on the odds of an accelerate-able VNF being placed on a server node with a hardware accelerator. The probability (p_{acc}) of allocation of an accelerator to a VNF using the accelerator-agnostic VNF-PC heuristic is thus given by the product $p_{acc} = \rho_{acc}^{vnf} \rho_{acc}^{n}$. Here, ρ_{acc}^{vnf} is the fraction of VNF that can be offloaded using a hardware accelerator (accelerate-able VNFs) and ρ_{acc}^{n} is the fraction of server nodes attached with a hardware accelerator. Therefore, p_{acc} gives odds of accelerator allocation to a VNF with the accelerator-agnostic heuristic. This can also be verified from the resulting β/α ratios depicted in

Fig. 2.12 (b) and (e). The β/α ratio for the accelerator-agnostic heuristic remains smaller than the accelerator-aware heuristic for any value of total VNF-chains. The explicit allocation of accelerators to VNFs occurs in the accelerator-aware heuristic leading to a superior probability of accelerator allocation $p_{acc} \approx \rho_{acc}^{vnf}$; which is in contrast to the accelerator-agnostic heuristic where accelerator-allocation is arbitrary. Moreover, we observed an increase in β/α ratio with the increasing number of total VNF-chains. This observation can be attributed to the fact that the accelerator-aware heuristic attempts to reuse the deployed accelerator instances and nodes attached with hardware accelerators. Therefore, increasing the number of VNF-chains causes an increase in the number of candidates for accelerator allocation; thus resulting in a better β/α ratio.

CPU_{rem} metrics for both heuristics are depicted in Fig. 2.12 (c) and (f). VNF consolidation on server nodes tends to increase with the total number of VNF chains as the chance of placing VNF on a server node will increase with the increase in the total number of VNFs. This is confirmed by the decreasing CPU_{rem} with the increasing number of VNF-chains for both topologies. Moreover, as more VNFs are granted accelerator using the accelerator-aware heuristic compared to the accelerator-agnostic heuristic, the corresponding CPU_{rem} is smaller and therefore more VNF consolidation is achieved.

We also try to show the impact of changing the fraction of nodes ρ_{acc}^{n} with an attached hardware accelerator card on overall performance metrics; when deploying the same set of VNF-chains. For this experiment, we decreased the fraction of server nodes attached with a hardware accelerator in the three-tier topology with $k = 10$ and measured the performance metrics for both heuristics. It can be observed from Fig. 2.13 that the total nodes cost for accelerator-aware heuristic remains less than that of the accelerator-agnostic heuristic for all values of ρ_{acc}^{n}. Also, as the fraction of nodes with hardware accelerator ρ_{acc}^{n} is reduced, the additional accelerator cost is decreased for both accelerator-agnostic and accelerator-aware VNF-PC heuristics which is negated by the additional costs due to the requirement of extra server nodes.

As expected, β/α ratio decreases with decreasing ρ_{acc}^{n} for both accelerator-agnostic and accelerator-aware heuristics. The β/α ratio decreases almost linearly with the decrease in ρ_{acc}^{n}. This, again, arises from the fact that the probability of accelerator allocation in the accelerator-agnostic heuristic is directly proportional to ρ_{acc}^{n} value which is not the case with the accelerator-aware heuristic. As placement decisions for accelerate-able VNFs are separate in accelerator-aware VNF-PC heuristic, there is no drastic impact on its β/α ratio with a decrease in ρ_{acc}^{n} value.

There isn't any significant change in CPU_{rem} for both heuristics with the change in ρ^n_{acc} values. However, VNF consolidation for accelerator-aware heuristic is better than the accelerator-agnostic heuristic, as was expected.

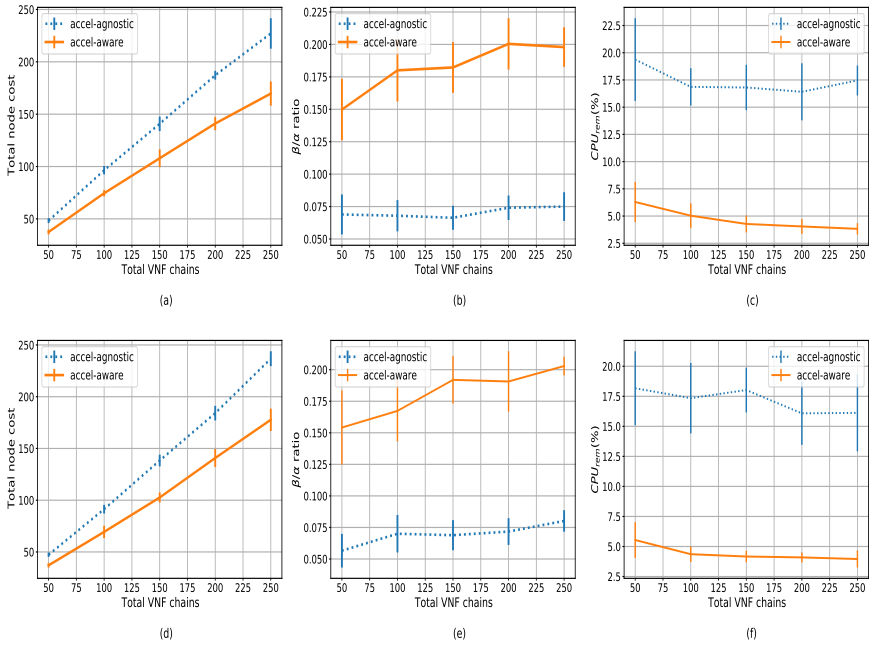


Figure 2.12: Comparison of accelerator-agnostic and accelerator-aware heuristics in terms of total node costs, β/α ratio and CPU_{rem} for the three-tier and leaf-spine topologies. Plots for the three-tier topology are shown in (a), (b) and (c) and plots in (d), (e) and (f) correspond to the leaf-spine topology.

2.7.2 Overall cost analysis

In this section, we analyze the cost-saving achieved as a result of incorporating hardware acceleration in NFVi. We assume the total number of server nodes (without any hardware accelerator) required for the deployment of a given set of VNF-chains is N. The total cost $Cost_0$ incurred to the operator as a result of running server nodes can be expressed as follows:

$$Cost_0 = Nc_0 \tag{2.14}$$

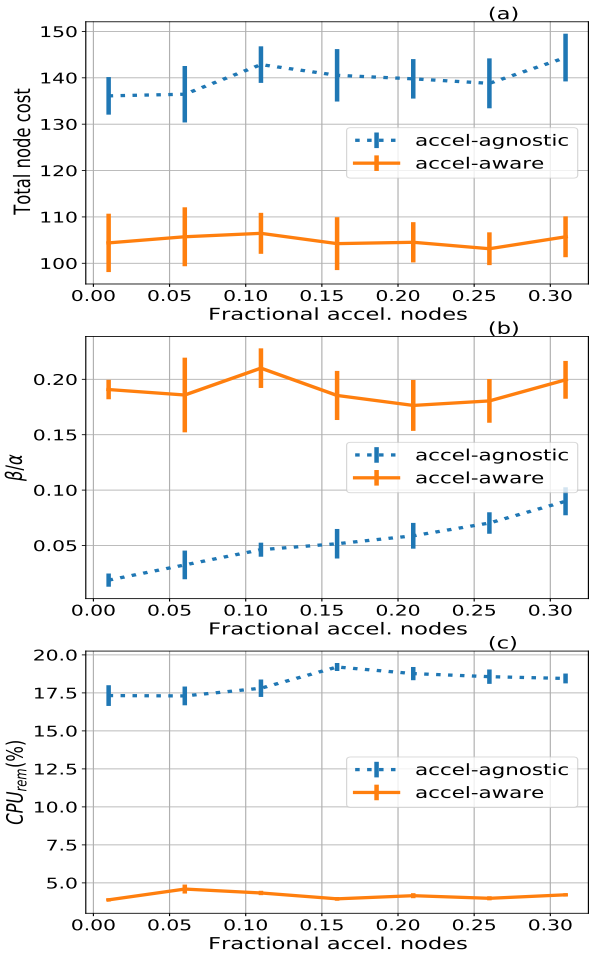


Figure 2.13: Impact of fraction of nodes with accelerator ρ^n_{acc} on (a) the total number of required nodes, β/α ratio and CPU_{rem} for the deployment of 100 VNF-chains on the three-tier topology ($k = 10$) using accelerator-agnostic and accelerator-aware VNF-PC heuristics.

Here, c_0 is the cost of running a single server node (without hardware accelerator) and N is the total number of server nodes required for the deployment of a set of VNF-chains.
After the installation of hardware accelerators in server nodes, the total cost of deployment $Cost_{acc}$ for the same set of VNF-chains can be expressed as follows:

$$Cost_{acc} = (1 + c_{acc})c_0 N(1 - \rho_{red})\rho_{acc}^n + c_0 N(1 - \rho_{red})(1 - \rho_{acc}^n) \quad (2.15)$$

The first term and the second term of eq. 2.15 refer to the cost of using server nodes attached with and without hardware accelerators, respectively. c_{acc} is the additional cost of installation of hardware accelerator in a server node relative to the original server node cost, ρ_{acc}^n is the fraction of server nodes that are installed with a hardware accelerator and ρ_{red} is the relative (total number of server nodes) reduction in the number of server nodes after hardware-acceleration for VNFs.
Fig. 2.14 compares the total server nodes required for the deployment of 100 VNF-chains on a leaf-spine topology in two cases, (i) when server nodes are not attached with any hardware accelerator card and (ii) when server nodes are attached with a hardware accelerator card. We have used the accelerator-agnostic heuristic for the case when NFVi does not contain any hardware accelerators and for the case when NFVi contains hardware accelerators, we have used the accelerator-aware heuristic. It can be observed that the relative reduction ρ_{red} in the total number of server nodes by using hardware accelerators is 18-20%.

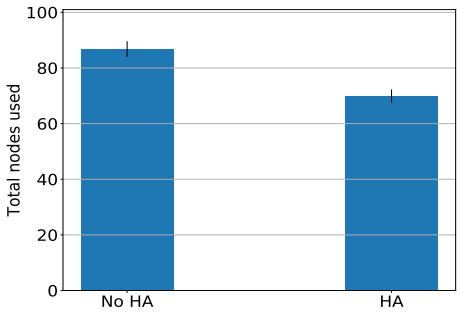


Figure 2.14: Total server nodes required for the deployment of 100 VNF-chains on a leaf-spine topology in two cases, (i) when server nodes are not attached with any hardware accelerator card and (ii) when sever-nodes are attached with a hardware accelerator card.

Relative cost savings (G) is the relative reduction in the total cost as a

result of using hardware acceleration in NFVi. G can be obtained by using the expression shown below:

$$G = (Cost_0 - Cost_{acc})/Cost_0 = 1 - (1 - \rho_{red}).\left(1 - \rho_{acc}^n + (1 + c_{acc}).\rho_{acc}^n\right) \tag{2.16}$$

Eq. 2.16 gives an expression for the achievable cost-saving in terms of relative server node reduction ρ_{red} and additional costs of hardware accelerators (c_{acc}). Using eq. 2.16, we can plot the required minimum ρ_{red} to achieve a given cost-saving G as shown in Fig. 2.15. Fig. 2.15 shows four different contours corresponding to four different G values. For example, to achieve an overall 15% savings ($G = 0.15$) on server nodes cost, the VNF-PC algorithm should achieve at least 18% reduction of total server nodes, when an additional cost of 18.5% is needed for the installation of hardware accelerators. As expected, one can observe that higher G values require high server node reduction ρ_{red} and low additional costs c_{acc}. Therefore, efficient accelerator-aware VNF-PC heuristics are required to gain the benefits of hardware accelerators even when their cost is expected to fall in the future. As stated earlier, about $\rho_{red} = 18 - 20\%$ reduction in total server nodes can be obtained using our VNF-AAPC heuristic. As a result, about 15% of overall cost-saving (G) is achievable by the operator.

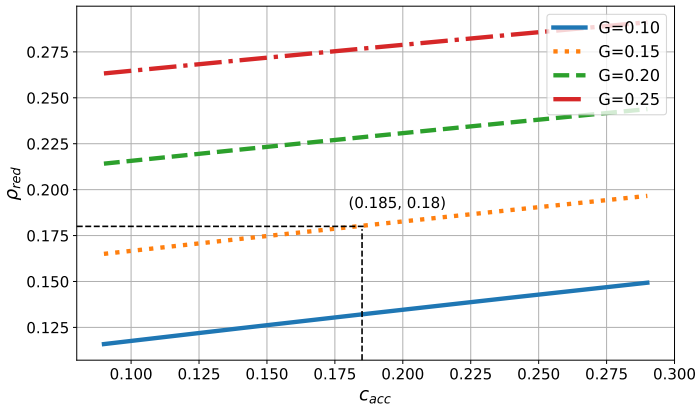


Figure 2.15: Variation of relative node reduction ρ_{red} with respect to additional hardware accelerator cost for VNF-PC heuristic's. Each line represent a locus of all points with fixed value of cost-saving G.

2.8 Conclusion

NFVi generally includes all hardware and software components required to build a virtualized environment for running VNFs. However, due to specific performance or energy goals, it becomes essential to provide some kind of acceleration to certain VNFs. However, the current NFVi resource allocation models do not consider hardware accelerator resources while performing placement and chaining of VNFs; therefore, resulting in inefficient utilization of NFVi resources.

In this paper, we modeled the VNF-AAPC problem for NFV environments containing hardware accelerators along with the usual NFVi resources. To tackle the VNF-AAPC problem, we proposed two approaches: (i) the ILP method and (ii) the heuristic algorithm. As opposed to the ILP-approach, the heuristic-based method is able to scale with the problem size at the cost of a small penalty. Both approaches aim at minimizing the cost incurred to the operator due to the utilization of resources for the deployment of VNF-chains. The heuristic-based approach performs tasks of VNF placement and chaining in two different phases: (i) the Placement of accelerate-able VNFs, (ii) the Placement and Chaining of remaining VNF-chain segments. The proposed methods were also evaluated using simulation experiments and then were compared in terms of their resulting cost and other performance metrics. The simulation results indicate that the accelerator-aware heuristic approach can achieve 12-14% cost savings as compared to the accelerator-agnostic heuristic. Finally, we also performed an overall cost analysis on the use of hardware accelerators in NFV environments. The analysis shows that the proposed accelerator-aware VNF-PC heuristic could be used to achieve significant cost savings when using hardware accelerators in NFVi.

Hardware accelerators are not only utilized in cloud DCs for performance enhancements of VNFs but also in other scenarios e.g. network edges, Centralized Radio Access Networks (CRANs). To reduce energy costs and meet the strict performance requirements in CRAN, various techniques to offload baseband processing functions, e.g. iFFT/FFT, turbo-coding, using hardware accelerators are being investigated. However, the problem of modeling resource dimensioning for virtual base stations in cloud RANs (C-RAN) architectures with hardware accelerators still remains to be investigated.

2.9 Acknowledgments

This work was funded through NGPaaS, under the grant number 761557, in the scope of the European Commission Horizon 2020 and 5G-PPP programs.

References

[1] B. Yi, X. Wang, K. Li, M. Huang, et al. *A comprehensive survey of network function virtualization.* Computer Networks, 133:212–262, 2018.

[2] *Architectural Framework*, 2013. Online; Release v1.1.1 2013-10. Available from: https://www.etsi.org/deliver/etsi_gs/nfv/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf.

[3] L. Linguaglossa, S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, and G. Bianchi. *Survey of Performance Acceleration Techniques for Network Function Virtualization.* Proceedings of the IEEE, 107(4):746–764, 2019.

[4] N. Nikaein. *Processing radio access network functions in the cloud: Critical issues and modeling.* In Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services, pages 36–43. ACM, 2015.

[5] *Acceleration Technologies; Report on Management Aspects Specification*, 2015. Online; Release v2.4.1 2018-02. Available from: https://www.etsi.org/deliver/etsi_gs/NFV-IFA/001_099/004/02.04.01_60/gs_NFV-IFA004v020401p.pdf.

[6] J. G. Herrera and J. F. Botero. *Resource allocation in NFV: A comprehensive survey.* IEEE Transactions on Network and Service Management, 13(3):518–532, 2016.

[7] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee. *Network function virtualization: Challenges and opportunities for innovations.* IEEE Communications Magazine, 53(2):90–97, 2015.

[8] S. Gebert, A. Müssig, S. Lange, T. Zinner, N. Gray, and P. Tran-Gia. *Processing time comparison of a hardware-based firewall and its virtualized counterpart.* In International Conference on Mobile Networks and Management, pages 220–228. Springer, 2016.

[9] *Addressing 5G Network Function Requirements.* White Paper, 2018. Online.

[10] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet. *Dynamic Hardware-Acceleration of VNFs in NFV Environments.* In 2019 Sixth International Conference on Software Defined Systems (SDS), pages 254–259, June 2019. doi:10.1109/SDS.2019.8768671.

[11] Z. Martinasek, J. Hajny, D. Smekal, L. Malina, D. Matousek, M. Kekely, and N. Mentens. *200 Gbps Hardware Accelerated Encryption System for FPGA Network Cards*. In Proceedings of the 2018 Workshop on Attacks and Solutions in Hardware Security, pages 11–17. ACM, 2018.

[12] X. Li, X. Wang, F. Liu, and H. Xu. *DHL: Enabling flexible software network functions with FPGA acceleration*. In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pages 1–11. IEEE, 2018.

[13] X. Ge, Y. Liu, D. H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu. *OpenANFV: Accelerating Network Function Virtualization with a Consolidated Framework in Openstack*. In ACM SIGCOMM Computer Communication Review, volume 44, pages 353–354. ACM, 2014.

[14] A. Albanese, P. S. Crosta, C. Meani, and P. Paglierani. *Gpu-accelerated video transcoding unit for multi-access edge computing scenarios*. In Proceeding of ICN, 2017.

[15] M. Masoudi, M. G. Khafagy, A. Conte, A. El-Amine, B. Françoise, C. Nadjahi, F. E. Salem, W. Labidi, A. Süral, A. Gati, et al. *Green Mobile Networks for 5G and Beyond*. IEEE Access, 7:107270–107299, 2019.

[16] S. Han, K. Jang, K. Park, and S. Moon. *PacketShader: a GPU-accelerated software router*. ACM SIGCOMM Computer Communication Review, 41(4):195–206, 2011.

[17] X. Yi, J. Duan, and C. Wu. *Gpunfv: a gpu-accelerated nfv system*. In Proceedings of the First Asia-Pacific Workshop on Networking, pages 85–91, 2017.

[18] B. Li, K. Tan, L. L. Luo, Y. Peng, R. Luo, N. Xu, Y. Xiong, P. Cheng, and E. Chen. *Clicknp: Highly flexible and high performance network processing with reconfigurable hardware*. In Proceedings of the 2016 ACM SIGCOMM Conference, pages 1–14. ACM, 2016.

[19] Z. Bronstein, E. Roch, J. Xia, and A. Molkho. *Uniform handling and abstraction of NFV hardware accelerators*. IEEE Network, 29(3):22–29, 2015.

[20] Y. Watanabe, Y. Kobayashi, T. Takenaka, T. Hosomi, and Y. Nakamura. *Accelerating NFV application using CPU-FPGA tightly coupled architecture*. In 2017 International Conference on Field Programmable Technology (ICFPT), pages 136–143. IEEE, 2017.

[21] *Acceleration Technologies; Report on Acceleration Technologies & Use Cases*, 2015. Online; Release v1.1.1 2015-12. Available from: https://www.etsi.org/deliver/etsi_gs/nfv-ifa/001_099/001/01.01.01_60/gs_nfv-ifa001v010101p.pdf.

[22] H. Moens and F. De Turck. *VNF-P: A model for efficient placement of virtualized network functions.* In 10th International Conference on Network and Service Management (CNSM) and Workshop, pages 418–423. IEEE, 2014.

[23] S. Mehraghdam, M. Keller, and H. Karl. *Specifying and placing chains of virtual network functions.* In 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), pages 7–13, Oct 2014. doi:10.1109/CloudNet.2014.6968961.

[24] B. Addis, D. Belabed, M. Bouet, and S. Secci. *Virtual network functions placement and routing optimization.* In 2015 IEEE 4th International Conference on Cloud Networking (CloudNet), pages 171–177. IEEE, 2015.

[25] M. Ghaznavi, A. Khan, N. Shahriar, K. Alsubhi, R. Ahmed, and R. Boutaba. *Elastic virtual network function placement.* In 2015 IEEE 4th International Conference on Cloud Networking (CloudNet), pages 255–260. IEEE, 2015.

[26] S. Sahhaf, W. Tavernier, M. Rost, S. Schmid, D. Colle, M. Pickavet, and P. Demeester. *Network service chaining with optimized network function embedding supporting service decompositions.* Computer Networks, 93:492–505, 2015.

[27] F. Carpio, S. Dhahri, and A. Jukan. *VNF placement with replication for load balancing in NFV networks.* In 2017 IEEE International Conference on Communications (ICC), pages 1–6. IEEE, 2017.

[28] H. Fan, Y. Hu, S. Zhang, and Q. Ren. *Hardware Acceleration Resource Allocation Mechanism for VNF.* Procedia computer science, 131:746–755, 2018.

[29] S. Dräxler and H. Karl. *SPRING: Scaling, Placement, and Routing of Heterogeneous Services with Flexible Structures.* In 2019 IEEE Conference on Network Softwarization (NetSoft), pages 115–123, June 2019. doi:10.1109/NETSOFT.2019.8806700.

[30] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet. *VNF-AAP: Accelerator-aware Virtual Network Function Placement.* 2019.

[31] N. Kodirov, S. Bayless, F. Ruffy, I. Beschastnikh, H. H. Hoos, and A. J. Hu. *VNF chain allocation and management at data center scale*. In Proceedings of the 2018 Symposium on Architectures for Networking and Communications Systems, pages 125–140. ACM, 2018.

3

On Decomposition and Deployment of Virtualized Media Services

In telecom industry, NFV aims to replace network middleboxes with low-cost COTS hardware that is used to host VNFs. Similarly, MFV proposes to transition from the architecture where media transport and processing is based on specialized hardware towards the architecture based on COTS hardware. MFV shall potentially help broadcasters to reduce their expenditures as it is happening with NFV. In Chapter 2, we addressed the resource allocation problem for network services in the context of NFV. However, due to some peculiarities of media services vis-a-vis network services, additional opportunities can be exploited that are not obvious with network services. The advantage of decomposing media streams and VMFs can be utilized for the optimization of a media service's VMF-FG.

In this chapter, a VMF-FG decomposition algorithm is proposed that optimizes a given media service prior to its deployment. For VMF-FG deployment, we present two VMF-PC heuristics: NFPC and k-cutPC. In comparison to NFPC, which is inspired from the heuristic presented in Chapter 2, the k-cutPC heuristic is suitable for the deployment of media services.

⋆⋆⋆

G.P. Sharma, D. Colle, W. Tavernier, and M. Pickavet

Published in the IEEE Transactions on Broadcasting, vol. 67, no. 3, pp. 761–775, 2021.

3.1 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.

3.2 Introduction

Today's TV broadcasters 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 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 are 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 the processing of several uncompressed media streams through a network of software-based media processing functions, which are referred here as Virtual Media Functions (VMFs). The 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 a mapping of the service's VMF Forwarding Graph (VMF-FG) to the underlying infrastructure. The mapping involves an assignment of Virtual Media Functions (VMFs) to server nodes, along 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. Evaluating the performance of the proposed VMF-PC algorithms with the varying decomposition of VMF-FGs.

The rest of the paper [1] is structured as follows. In Section 3.3, 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 3.4. Section 3.4 also describes the two VMF-PC algorithms– NFPC and k-cutPC. The evaluation of the two VMF-PC algorithms and VMF-FG decomposition is presented in Section 3.5. Finally, Section 3.6 draws the main conclusions along with the potential future research.

3.3 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 with 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 scenarios 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.

[1] The section in the paper on the MFV architecture is contained in section 1.4.2 of the introduction chapter.

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.

3.3.1 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 non-blocking switching operation at all times. In addition, SDI-based media transport in studios has 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 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. It is in contrast to 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 an IP-routed infrastructure. A multilayer switching architecture, as commonly used in data centers, can be thought of as a single switch interconnecting all the studio devices. Fig. 3.1 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 data centers owned by Google or Facebook, the same fat-tree topology can be used to fulfill the broadcast studio requirements. Moreover, data center topologies are 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 data center 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, 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

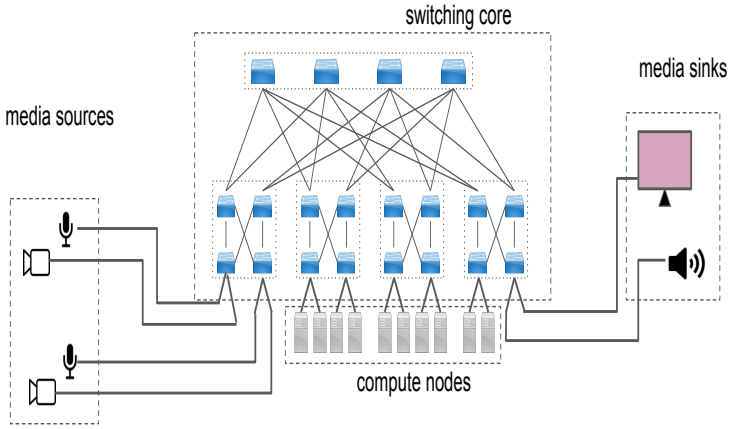


Figure 3.1: Illustration of studio architecture based on a data center topology.

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 3.1 lists different standards of the suite ST 2110 along with a short description.

For the independent transport of media essences, SMPTE ST 2110-10/-

Table 3.1: 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

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 a full-frame is generated for further processing or display.

3.3.1.1 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. 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. 3.2 illustrates 4-way decomposition of a 4K video stream. The original 4K (3840x2160p30) stream with the bitrate of 4.976 Gbps when 4-way decomposed results in four 2K (1920x1080p30) sub-streams each with the bitrate of 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 3.4.

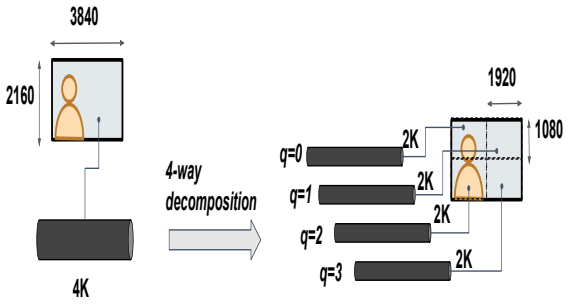


Figure 3.2: An example of 4-way SD decomposition of 4K video stream.

3.3.2 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) [2] 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 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 on the media service deployment problem, 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 (NFV). NFV is an architecture where packet processing functionality, which is conventionally realized by specialized hardware appliances called middleboxes,

[2]The speed-up in overall throughput in the last few decades is mostly because of the parallelism resulting from multiple CPU cores. The increase in single processor speed has been marginal.

is being realized using Virtual Network Functions (VNFs). These VNFs hosted 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 environment; we refer to these software MF implementations as Virtual Media Functions (VMFs). Tab. 3.2 lists some example VMFs along with a short description for each of them.

Table 3.2: Example VMFs with short description.

Name	Notation	Description
Chroma key	*chrm-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-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 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 broadcasters 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. A detailed discussion on the MFV architecture can be found Chapter 1. The CO layer in Fig. 1.13 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 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.

3.4 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 the 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 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, e.g., 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. 3.3. 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,

Table 3.3: 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 ports 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.

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 multi-viewer 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 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 the 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. 3.3 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 $\mathcal{L} = \{(src0, chrm\text{-}key), (src1, chrm\text{-}key), (src2, scl), (chrm\text{-}key, pip), (scl, pip), (chrm\text{-}key, vqa), (vqa, dst0), (pip, dst1)\}$. 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-key* applies the chroma-keying operation on 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-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 streams. The output of *chrm-key* is also multicasted 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,

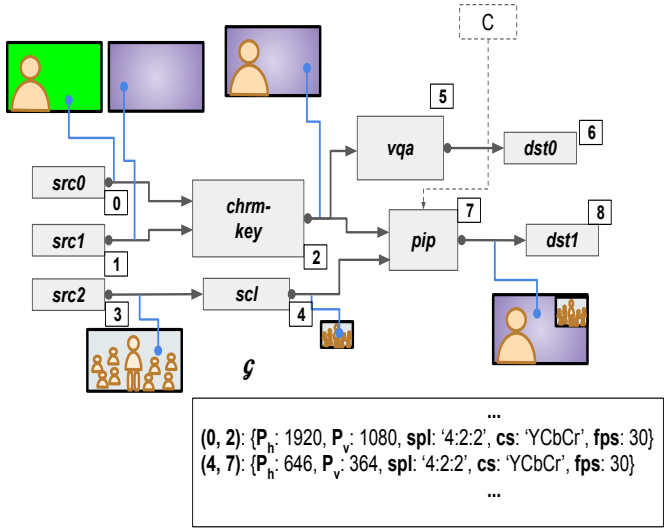


Figure 3.3: The VMF-FG $\mathcal{G}$ representation of an example media service.

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

3.4.1 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. Decomposing the virtual links of a VMF-FG further allows the decomposition of the VMFs of the given VMF-FG. With VMFs and virtual links being decomposed, several optimizations 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. 3.2. For the sake of simplicity, we assume that all VMFs are Multiple Input Single Output (MISO) types, similar to the one shown in Fig. 3.4. 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. 3.4, 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. 3.4, 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'.

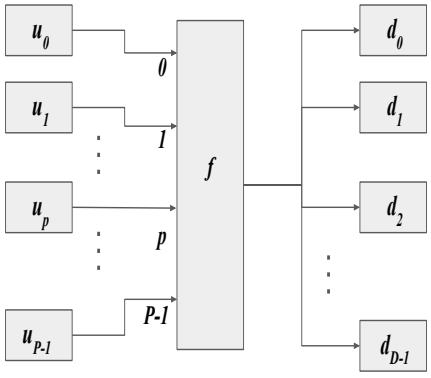


Figure 3.4: An illustration of a MISO VMF f.

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. 3.5 (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. 3.5 (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.

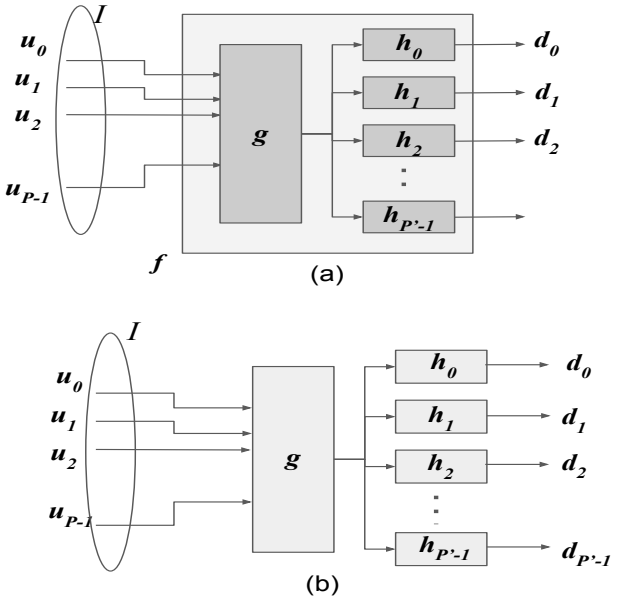


Figure 3.5: (a) Internals of the MIMO VMF f and (b) its decomposition into MISO VMFs g, h_0, h_1, ..., $h_{P'-1}$

The VMF-FG decomposition algorithm consists of four steps as shown in Fig. 3.6.

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 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,

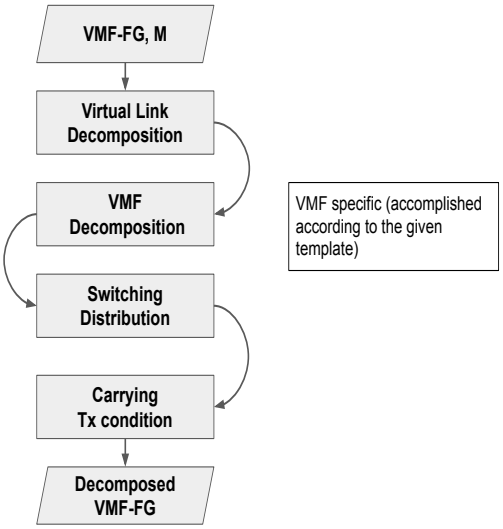


Figure 3.6: Flowchart showing the VMF-FG decomposition procedure.

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. 3.7 (a) whereas Fig. 3.7 (b) shows link decomposition of (f_i, f_j) into M new links.

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-key* and picture-in-picture (pip) VMF can be decomposed.

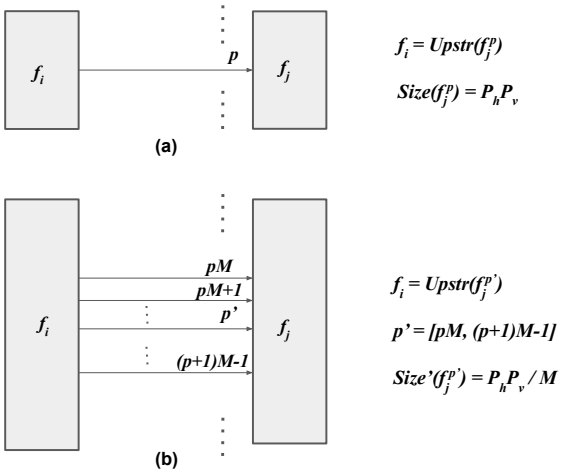


Figure 3.7: Illustration for the virtual link decomposition step. (a) The virtual link $(f_i, f_j) \in \mathcal{L}$ and (b) its decomposition by M.

The VMF decomposition step for *chrm-key* is quite straightforward as shown in Fig. 3.8 (c). Prior to the VMF decomposition, the link decomposition step is performed, as shown in Fig. 3.8 (b). Each link in Fig. 3.8 (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 quadrants of a frame. As each output pixel of *chrm-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-key* can be decomposed into *chrm-key*0, *chrm-key*1,*chrm-key*2 and *chrm-key*3, 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. 3.9 (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. 3.9 (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. 3.9 (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. 3.9 (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. 3.10 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. 3.11. 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 dis-

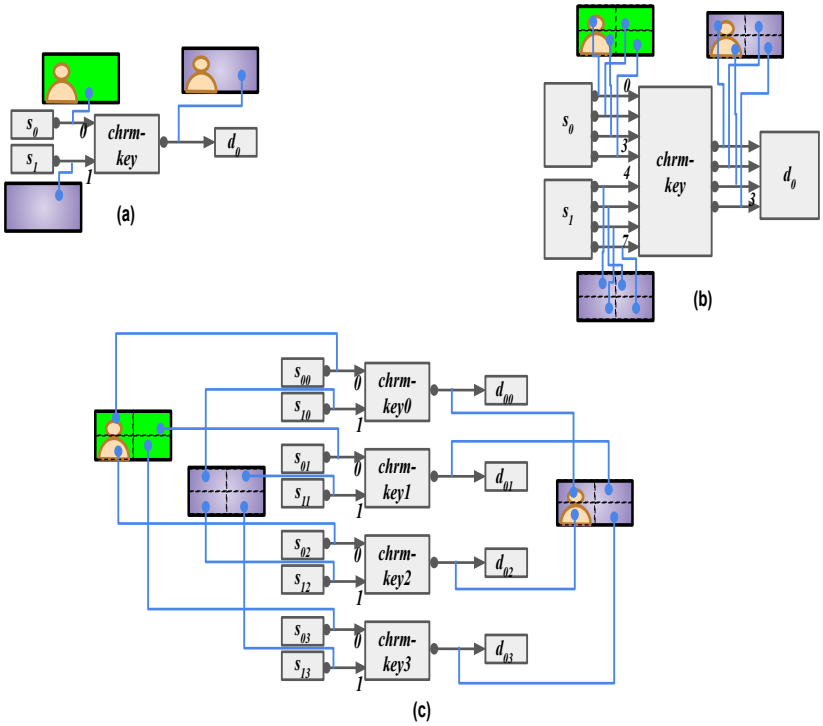


Figure 3.8: 4-way decomposition of the chrm-key VMF. (a) Illustration of operation of the chrm-key VMF. The chrm-key VMF after (b) the virtual link decomposition step and (c) the VMF decomposition step.

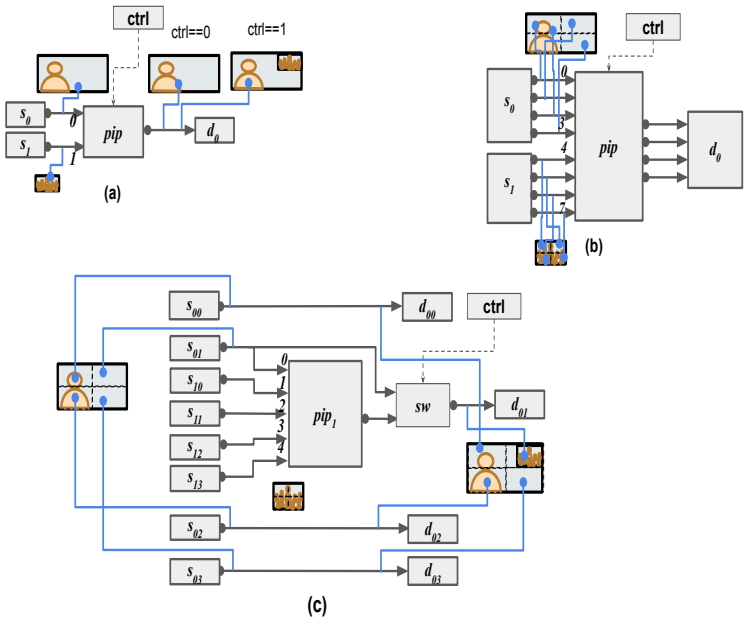


Figure 3.9: 4-way decomposition of the pip VMF. (a) Illustration of operation of the pip VMF. The pip VMF after (b) the virtual link decomposition step and (c) the VMF decomposition step.

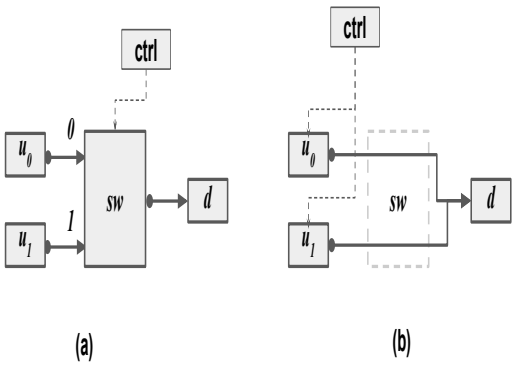


Figure 3.10: Illustration showing the distribution of switching functionality of a VMF to its upstream VMFs.

tributed 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 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. 3.11 (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. 3.11 (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 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 the computation of f and its upstream 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 consec-

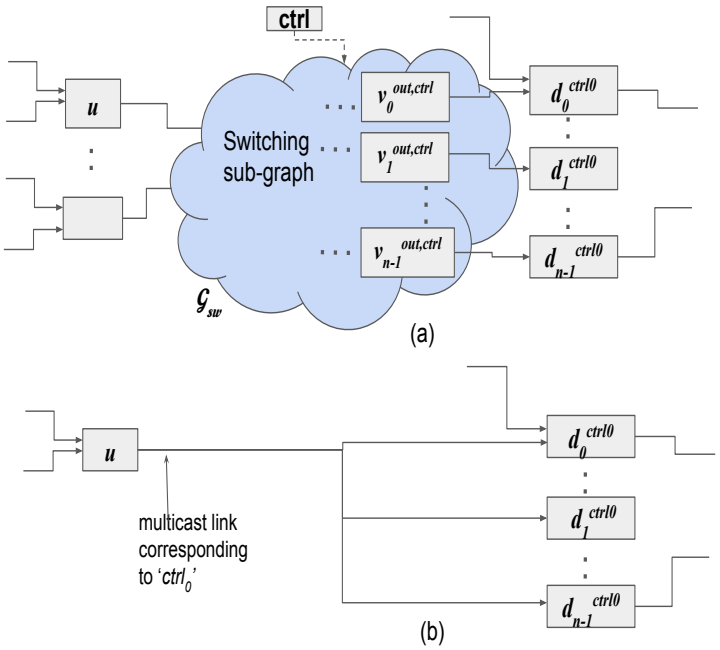


Figure 3.11: Distribution of switching functionality of a switching subgraph to its upstream VMFs. (a) VMF-FG region showing a switching subgraph and (b) its distribution to the input VMF u for a given control value $ctrl_0$.

utive VMFs u, v and w during a path traversal starting with snk through $\mathcal{G}^3$, as shown in Fig. 3.12. 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 $\mathcal{G}'$.

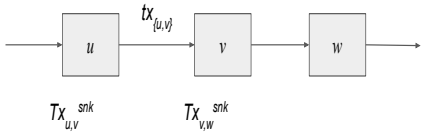


Figure 3.12: Propagation of transmit conditions from the downstream v VMF to the upstream VMF u.

3.4.2 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 the next-fit based VMF-PC algorithm we refer to as the NFPC algorithm and the second algorithm is based on the min k-cut algorithm we refer to as the *kcut*-PC algorithm.

3.4.2.1 Next-fit Approach

The pseudo-code for the next-fit based VMF placement and chaining algorithm is shown in Alg. 3.2. 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 denote 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 3.1: 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      return None, None
8  end
```

Algorithm 3.2: 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             else if f' ∉ qu_vmf then
13                 En-queue f' to qu_vmf
   /* All VMFs have been PC'ed                                    */
14 Update cap using used_res
```

Each sink node *snk* is considered as a starting node for the BFS traversal through the VMF-FG [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. 3.1, 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. 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-stream VMFs, $used\_res$ is updated according to the resource demands of f. 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; 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. 3.1 and Alg. 3.2 that the asymptotic time complexity of the NFPC heuristic is linear in the number of requests and total server nodes, i.e., $\mathcal{O}(|R| * |N_c|)$.

3.4.2.2 k-cut Approach

The pseudo-code for k-cutPC algorithm is shown in Alg. 3.5. The output of the algorithm is same as that of the NFPC algorithm, whereas the input additionally includes n_{bst} shown in Alg. 3.2. 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. 3.2. 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. 3.4. It uses the random contraction algorithm `RandContr` to get a k-cut of an undirected 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 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. 3.5.
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. 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 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 . 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 3.3: 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      wt ← Sum of weight of all edges crossing last k nodes
7      return G, wt
8  end
```

Algorithm 3.4: Procedure for generating n_{bst} best k cuts

```
1  Procedure NbstKcut(G = (F, L), k, n_bst):
2      G_cltrs, W_cltrs ← φ, φ ;            /* Set of generated cuts and
         corresponding weights */
3      N_runs ←| F |^(2k−2) * ln | F |
4      while N_runs > 0 do
5          G_cut, wt_cut ← RandContr(G, k) /* get a random k-cut     */
6          if | G_cltrs |≤ n_bst then
7              G_cltrs ← (G_cltrs ∪ G_cut) /* gather n_bst best cuts  */
           /* replace worst (max. weight) cut with G_cut             */
8          else if wt < max W_cltrs then
9              G_rep ← G ∈ G_cltrs with wt > wt_cut /* k-cut with min.
                   weight but > wt_cut                                */
10             Replace G_rep in G_cltrs with G_cut
11         N_runs ← N_runs − 1
12     return G_cltrs
13 end
```

Algorithm 3.5: Algorithm for the k-cut based VMF-PC.

```
   /* VMF-FG, MFVi network and # of best k-cuts                    */
   Input    : 𝒢 = (ℱ, ℒ), G^I = (N, E), cap, n_bst
   /* VMF placement and chaining mapping                           */
   Output   : α, γ
   Initialize: used_res, α, γ
 1 C_cpts ← ConccCompts(𝒢) ;      /* all connected components of 𝒢 */
 2 for 𝒞 in C_cpts do
 3 |   Calculate k_min, k_max values
 4 |   for k in [k_min, k_max] do
 5 |   |   (ℱ_cuts, ℒ_cuts) = NbstKcut(𝒞, k, n_bst) /* get n_bst best k-cuts
   |   |       */
 6 |   |   for (ℱ_cut, ℒ_cut) in (ℱ_cuts, ℒ_cuts) do
 7 |   |   |   PC of cluster (ℱ_cut, ℒ_cut) using Alg. 3.2
 8 |   |   |   if PC successful then
 9 |   |   |   |   Chain cluster (ℱ_cut, ℒ_cut) with other VMFs
10 |   |   |   |   If cluster chaining successful, update α, γ, used_res and
   |   |   |   |     break
11 |   |   |   break loop if 𝒞 is PC'd
12 |   |   If 𝒞 still not PC'd, stop the algorithm
   /* All connected components have been PC'ed                     */
13 Update cap using used_res
```

3.5 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 its 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 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. The 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.

3.5.1 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 is $\kappa^3/4$. Fig. 3.13 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 3.4. 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. The VMF-FG decomposition algorithm and both VMF-PC algorithms are written in Python and we have used the networkx

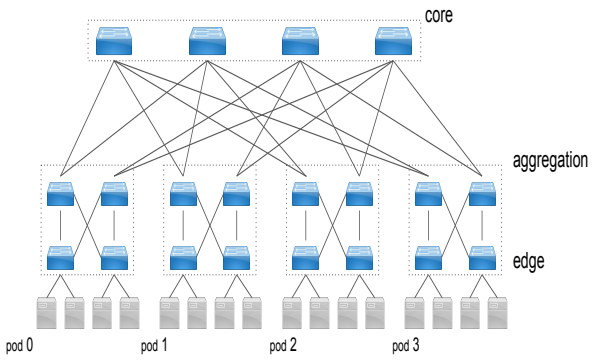


Figure 3.13: Fat-tree topology of a data center with $\kappa = 4$ pods.

package wherever necessary. We use an 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. 3.14.

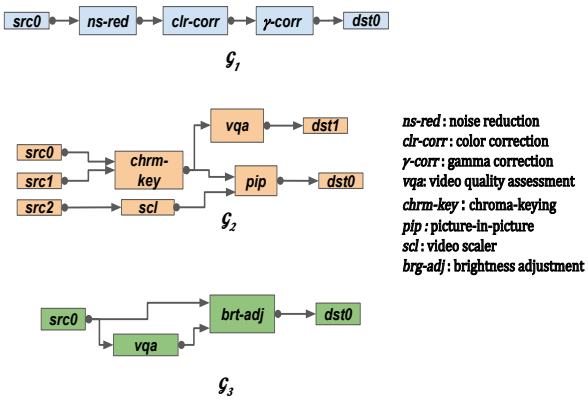


Figure 3.14: Set of VMF-FGs used in the evaluation.

3.5.2 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

Table 3.4: 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

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.
The acceptance ratio variation for NFPC and k-cutPC has been shown in Fig. 3.15 (a) and Fig. 3.15 (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. 3.15 (a) and Fig. 3.15 (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.

3.5.3 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. 3.16 (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 ineffi-

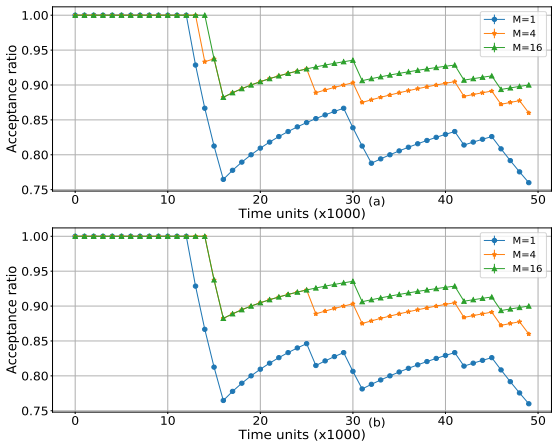


Figure 3.15: Acceptance ratio variation over time for the deployment of 50 requests on the MFVi with data center topology $\kappa = 4$ (16 server-nodes) for (a) NFPC and (b) k-cutPC algorithms.

ciency 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. 3.16 (b) shows the total number of physical links N_{lnks} reserved for both the VMF-PC algorithms. Similar to Fig. 3.16 (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.

3.5.4 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

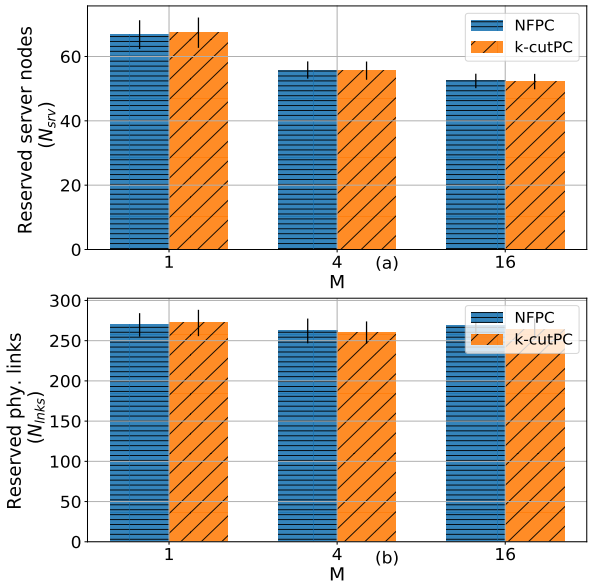


Figure 3.16: Variation of resource reservation with M for 50 requests on a data center topology with 128 server-nodes ($\kappa = 8$). (a) Total reserved server nodes and (b) total reserved physical links.

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. 3.17 (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. 3.17 (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.

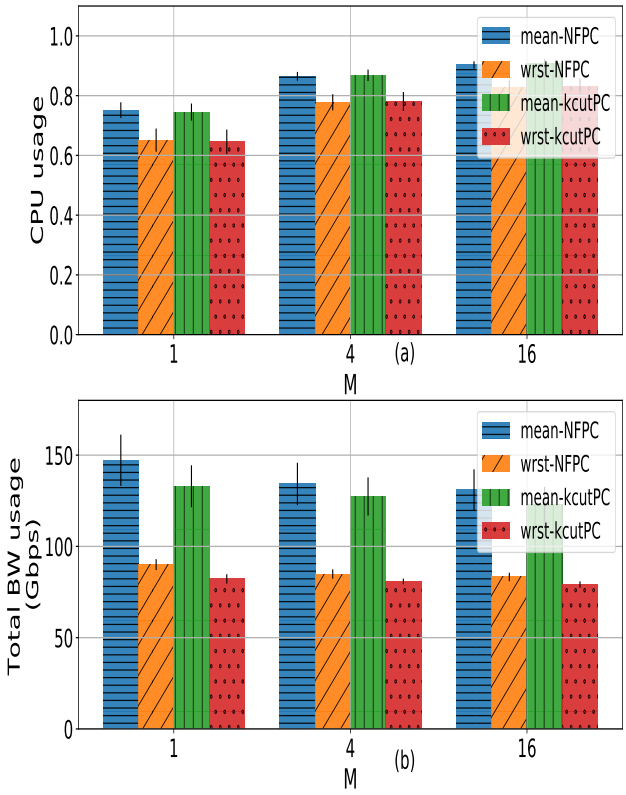


Figure 3.17: Variation of resource utilization with M for 50 requests on a data center topology with 128 server-nodes ($\kappa = 8$). (a) Normalized CPU utilization and (b) Total inter-node bandwidth utilization.

3.5.5 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. 3.18 (a), (b) and (c) shows average end-to-end hops for the VMF-FGs shown in Fig. 3.14. 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 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 do 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. 3.14).

3.6 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. Thus the adoption of COTS platforms is expected to rise 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 de-

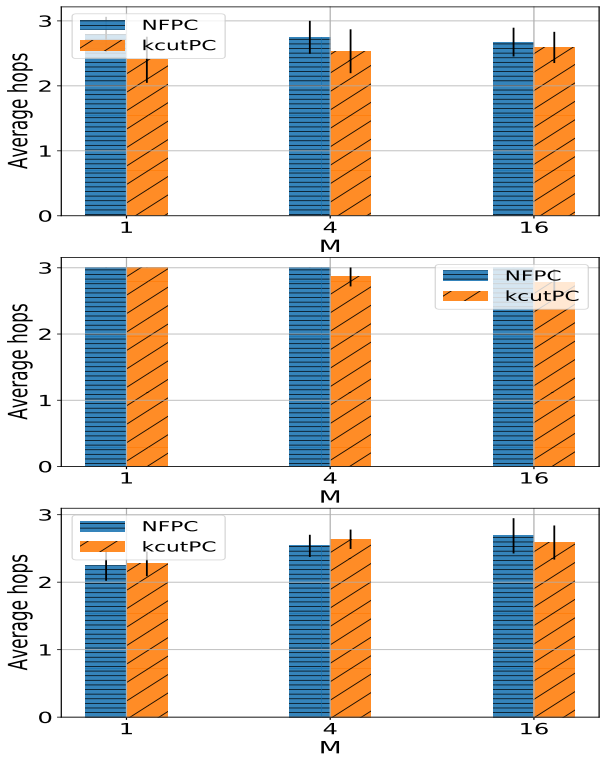


Figure 3.18: Variation of end-to-end hops for (a) $\mathcal{G}_1$, (b) $\mathcal{G}_2$ and (c) $\mathcal{G}_3$ on a data center topology with 128 server-nodes ($k = 8$).

lay 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.

3.7 Acknowledgments

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*. Technical report, RedHat, 02 2018.

[2] *The Road to COTS and the Cloud for real-time broadcast production*. Technical report, Nevion, 01 2018.

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

[4] Grass Valley. *Agile Media Processing Platform*.

[5] J. G. Herrera and J. F. Botero. *Resource allocation in NFV: A comprehensive survey*. IEEE Transactions on Network and Service Management, 13(3):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, pages 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), pages 31–37. IEEE, 2020.

[8] T. Fautier. *How OTT Services Can Match the Quality of Broadcast*. SMPTE Motion Imaging Journal, 129(3):16–25, 2020. doi:10.5594/JMI.2020.2969763.

[9] Y. Reznik, J. Cenzano, and B. Zhang. *Transitioning Broadcast to Cloud*. Applied Sciences, 11(2):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), pages 1–5. IEEE, 2016.

[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, pages 1–16, 2014.

[12] A. Kovalick. *Design elements for core ip media infrastructures*. SMPTE Motion Imaging Journal, 125(2):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/. 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, pages 1–17, 2017.

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

[16] *ST 2110-30:2017 - SMPTE Standard - Professional Media Over Managed IP Networks: PCM Digital Audio*. ST 2110-30:2017, pages 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, pages 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, pages 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, pages 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, 8:11456–11468, 2020.

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

[23] *Broadcast Video Infrastructure Implementation Using FPGAs*. Technical report, Altera, 03 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), pages 111–115. IEEE, 2018.

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

[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, STOC 2020, page 473–484, New York, NY, USA, 2020. Association for Computing Machinery. Available from: https://doi.org/10.1145/3357713.3384285, doi:10.1145/3357713.3384285.

4

Scheduling for Media Function Virtualization

In the previous chapter, we presented two VMF-PC algorithms for the deployment of media services. The VMF chaining process in these algorithms only takes in account the bandwidth of network links. Therefore, the traffic between any two neighbouring VMFs can be subjected to packet delays and losses due to congestion in the network. However, producing broadcast-quality content requires zero packet loss along with bounded latency and jitter between all neighbouring VMFs.
In order to guarantee zero packet loss and bounded delays, media service deployment needs to include packet scheduling between neighbouring VMFs. To this end, this chapter first formulates the problem of VMF-FG scheduling and attempts to solve it using a heuristic algorithm.

⋆ ⋆ ⋆

G.P. Sharma, W. Tavernier, D. Colle, and M. Pickavet

Published in Future Internet, vol. 13, no. 7, 2021.

Abstract Broadcasters are building studio architectures based on Commercial-off-the-shelf (COTS) IT hardware because of advantages such as cost-reduction, ease of management and upgradation. Media Function Virtualization (MFV)

leverages IP for media transport to processing media streams using Virtual Media Functions (VMFs) running on general-purpose compute (e.g., Intel Xeon servers) platforms. Media service deployment in an MFV environment entails solving the VMF-FG scheduling problem to ensure the required broadcast-quality guarantees. In this paper, we formulate the VMF-FG scheduling problem and propose a greedy-based algorithm to solve it. The evaluation of the algorithm is carried out in terms of the end-to-end delay and the VMF queuing delay. Moreover, the importance of VMF-FG decomposition in upgradation to higher-quality formats is also highlighted.

4.1 Introduction

The rising demand for new TV broadcast services, e.g., Over-the-Top (OTT) and high-quality content forces broadcasters to regularly upgrade studio infrastructure, thus resulting in high expenditures [1]. The cut-throat competition further accentuates the challenge of keeping the business viable. This challenge has forced broadcasters to seek alternative studio architectures for broadcast media production.

For several decades, Serial Digital Interface (SDI) has been the most preferred solution for media transport within studios because of its robust and reliable performance [2]. However, upgrading SDI-based infrastructure to support higher quality media, e.g., Full High Definition (FHD) and Ultra High Definition (UHD) videos, requires a high-speed SDI routing matrix-thus rendering the upgradation phase quite expensive and complicated [3].

As an alternative to SDI, packet-switched media transport architectures are gaining the attention of broadcasters. Internet Protocol (IP) is a versatile solution to interconnect devices on the Internet as well as private networks. Due to the decades of evolution in IP, the speed of packet forwarding devices has grown manifold. For instance, network cards and switches with Gbit/sec Ethernet (10GbE) interfaces are now commonly available. As a result of this evolution in IP, all-IP studio architectures are being explored to transport uncompressed media streams across studios [2].

A complementary transition is happening in the media processing domain to move from bespoke hardware towards Commercial-off-the-shelf IT hardware (e.g., Intel Xeon Servers) running software applications to process media streams [4]. COTS hardware being inexpensive and ubiquitous, the upgrade of media processing infrastructure is more economical compared to the specialized hardware boxes.

The progress made on these two fronts– COTS hardware exploitation for (i) media transport and (ii) media processing is crucial in accelerating the adoption of Media Function Virtualization (MFV). Analogous to Network

Function Virtualization (NFV), where network processing is accomplished via software deployed on COTS hardware, MFV aims to implement Media Functions (MFs) as software [4], [5]. Despite its advantages, MFV faces a unique set of challenges that are not relevant to NFV. Broadcast production quality standards are quite stricter than networking services, for instance, loss of a packet or even delay in the arrival of a packet can ruin the viewer experience. Deterministic Networking (DetNet) can be exploited to provide end-to-end timing guarantees for the transport of media streams [6]. Specifically, DetNet mechanism such as CSQF allows scheduling packet transmissions along the network path that can fulfill the real-time requirements needed from the underlying network [7].
In an MFV environment, media services are represented using a directed graph referred to as Virtual Media Function Forwarding Graph (VMF-FG). The deployment of a media service entails a mapping of the VMF-FG to the infrastructure, which is referred to as VMF Placement and Chaining (VMF-PC) problem. The naive VMF-PC algorithm assumes best-effort networking for media transport between VMFs thus does not provide any guarantee on metrics such as packet loss, end-to-end delay or jitter. Over-provisioning bandwidth for media streams could result in an improvement in these metrics, yet deterministic performance cannot be guaranteed. Therefore, the VMF-FG mapping process needs to incorporate scheduling of VMFs and virtual links to ensure broadcast-quality guarantees.
In this paper, we first formulate the VMF-FG scheduling problem and then propose a greedy heuristic to solve it. The performance of the heuristic is evaluated by conducting numerical experiments.
The rest of the paper [1] is structured as follows. The related works are presented in section 4.2. The problem statement and the proposed heuristic are described in section 4.3. The evaluation setup and results are described in section 4.4. Finally, our conclusions and final remarks are summarized in section 4.5.

4.2 Related Works

Broadcast media production is undergoing a massive transformation propelled by IP and virtualization technologies. The migration to IP-based media transport from SDI is foreseen to accelerate in the near future. The interest in IP among broadcasters is emphasized by the fact that the SMPTE has released a suite of ST 2110 standards [8–10]. These standards describe how different media essences, i.e., video, audio, and ancillary data, can be

[1] The sections in the paper on the MFV architecture and CSQF are contained in sections 1.4.2 and 1.5.1 of the introduction chapter, respectively.

transported independently using IP. Many all-IP broadcast studios are now being built based on these standards.
A live TV broadcast based on IP was produced by the British Broadcast Corporation (BBC) during the Glasgow 2014 Commonwealth Games [11]. UHD streams captured by multiple cameras in several competition venues were delivered to the software-defined production facility. The final program, after HEVC, was also delivered over IP. Likewise, a new broadcast facility in Wales, UK has been built based on IP.
The Canadian public broadcaster: CBC/Radio-Canada has constructed an all-IP facility responsible for over 100 TV, radio and online programming in Montreal, Canada [12]. The switching network is based on leaf-spine topology with 100Gbps links. The network can support real-time multicast traffic along with redundancy for media streams. The software-defined network ensures that media streams can be distributed uniformly over the topology. The flexibility offered by IP is crucial for COTS-based media processing, as discussed next.
D. Luzuriaga et al. have also demonstrated the PoC for a vision mixer based on open-source software tools (e.g., OBS studio and KX studio) running on COTS hardware platform [13]. It is presented a low-cost replacement for a specialized hardware production system. The evaluation shows that the vision mixing setup has an acceptable delay of about 1.4s, which is reasonable for professional (non-live) media production scenarios. Furthermore, live media mixing based on software has been demonstrated by the BBC partnered with Isotama [14]. To operate the setup in real-time and control the final live output, the software pipeline is interfaced with a browser-based operation tool.
Broadcast media services in an MFV environment can be represented using VMF-FGs. In our previous work, we proposed a VMF-FG decomposition algorithm [15]. The VMF-FG decomposition algorithm processes an input VMF-FG to produce an optimized VMF-FG that consumes fewer resources. Recently, some work has been done to address the joint routing and scheduling problem for DetNets [16]. The problem is formulated as an Integer Linear Program (ILP) and two methods– column generation and dynamic programming, are proposed to maximize traffic acceptance. An ultra-fast and scalable greedy heuristic is also proposed that is capable of solving the problem with a small penalty. In this paper, the DetNet requests are simply two endpoints (source and sink). The problem of VMF-FG scheduling assumes a timing relationship between different virtual links. To the best of our knowledge, there has not been any study on the scheduling of VMF-FGs in order to reliably deploy media services. This work aims to first formulate the VMF-FG scheduling problem and also proposes a heuristic to solve it.

4.3 System Model and Algorithm

An upper limit on timing metrics is required to ensure that broadcast-quality is maintained. To this end, media streams between VMFs can be scheduled using a queue scheduling mechanism such as CSQF. This shall ensure that each VMF receives its input frames within the required time window so that the output frame is produced and then timely received at the downstream VMFs.
Next, we describe the VMF-FG scheduling problem and propose a greedy-based heuristic to solve it.

Table 4.1: Description of the notations used for different parameters and variables involved in the system model.

Notation	Description
$\mathcal{G} = (\mathcal{F}, \mathcal{L})$	VMF-FG representation of a media service, where $\mathcal{F}$ and $\mathcal{L}$ are the set of VMF and virtual links, respectively.
$G^I = (N, E)$	Directed graph representation of the MFVi network infrastructure, where N is the set of nodes and E is the set of physical links between the nodes.
R_{frame}	Number of frames transmitted per second on a media stream.
T_c	Cycle time.
N_c	Set of all server nodes ($N_c \subset N$) , i.e., nodes with compute resources.
C	The set of all cycles in a hypercycle.
$bw_{n_i,n_j,c}$	Available bandwidth (in bytes) on physical link $(n_i, n_j) \in E$ during cycle $c \in C$.
α	The variable denotes the node and core assignment of VMFs.
γ	The variable denotes the physical path assignment of virtual links.
ω	The variable denotes the schedule assignment of virtual links.

4.3.1 System Model

A media service s is represented using a DAG denoted by $\mathcal{G} = (\mathcal{F}, \mathcal{L})$, where $\mathcal{F}$ is the set of VMFs and $\mathcal{L}$ is the set of virtual links. The CPU requirement of VMF f is denoted by cpu_f and the bandwidth requirement, in bytes per hypercycle, on a virtual link is denoted by bw_{f_i,f_j}. Here, the hypercycle time

is equal to the frame interval, i.e., $|C| * T_c = T_{frame} = 1/R_{frame}$. Thus, bandwidth requirement per hypercycle for (f_i, f_j) is equal to the frame size, e.g., an FHD (1920x1080 or 2K) @30fps video stream with YCbCr color space and 4:2:2 sub-sampling, the bandwidth requirement per frame is 5184000B (5MB).
The MFVi network is also denoted with a directed graph $G^I = (N, E)$, where N and E denote the set of physical nodes and links, respectively. The subset of physical nodes includes server nodes ($N_c \subset N$) with computational resources to host VMFs. The available CPU resources on a node $n \in N_c$ is denoted by cpu_n and bandwidth in cycle c on physical link (n_i, n_j) is denoted by $bw^c_{n_i,n_j}$.
Deploying a media service in an MFV environment involves mapping the service's VMF-FG onto the MFVi network. In the problem of VMF-FG scheduling, given a VMF-FG $\mathcal{G}$ and MFVi G^I, (i) the VMFs should be assigned to the server nodes ($\alpha : \mathcal{F} \rightarrow N_c$), (ii) the virtual links should be mapped to the physical paths ($\gamma : \mathcal{L} \rightarrow \texttt{Pths}$) and (iii) the virtual links should also be assigned a packet transmission schedule ($\omega : \mathcal{L} \rightarrow \mathbb{R}^{|C|}$). As shown in Fig. 4.1, the CO block has a view of the MFVi network (G^I); the scheduler in the CO block takes VMF-FG $\mathcal{G}$ and G^I as input and solves the VMF-FG scheduling to generate a solution. The solution (α, γ and ω) is then used to configure MFVi, e.g., assign VMFs to servers, segment routing configuration for CSQF schedule, etc. Table 4.1 lists the various parameters and variables involved in the system model.

4.3.2 VMF-FG Scheduling Algorithm

The VMF-FG scheduling algorithm is based on the Breadth-First Search (BFS) algorithm as shown in Alg. 4.1. The VMF-FG scheduling variables are initialized and the VMF-FG traversal starts from a sink node. For each (already placed) VMF $f_{dw} \in \texttt{DwstrNbrs}(\mathcal{F}, f)$ downstream to f, the scheduling of VMF f and the associated virtual link (f, f_{dw}) is iteratively attempted using procedure `VlinkSch`. The tentative node to place f is selected using the Next-fit algorithm; starting from the same node where f_{dw} is placed, i.e., $\alpha[f_{dw}]$, then the next node and so on ($\alpha[f_{dw}] + \delta n$). The neighbours of f are then added to the VMF queue. Next, we describe the procedure for virtual link scheduling in detail.
The procedure to schedule virtual links is presented in Alg. 4.2. The procedure `VlinkSch` takes as input the bandwidth requirement bw_{f_i,f_j} of the virtual link (f_i, f_j), the server nodes for the potential placement of f_i and f_j: n_1 and n_2, the current placement (α), chaining (γ) and link schedule (ω). As illustrated in Fig. 4.2, the input schedule at VMF f_j must end $d_{proc}(f_j)$ before f_j starts transmitting the processed (output) frame. Next, procedure

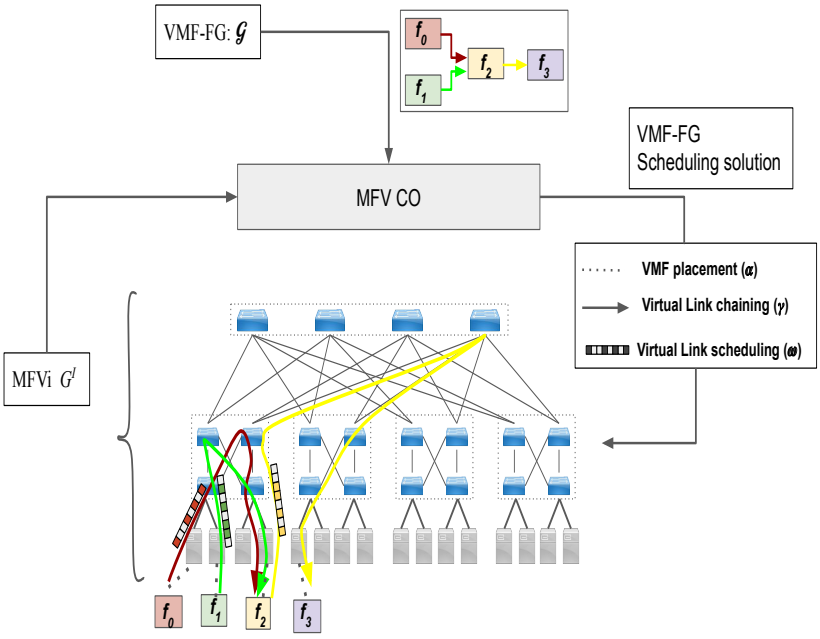


Figure 4.1: An overview of the VMF-FG scheduling problem.

`PthSchGrd` (discussed next) is called to greedily allocate bw_{f_i,f_j} among $|C|$ cycles between t_{end} and $t_{end} - T_c + 1$. The feasible path p, the associated schedule s along with the timings of the start and end cycles of s are returned by `PthSchGrd`. In case a feasible path/schedule does not exist, the procedure is terminated by returning ϕ. Else, VMF timings are updated and a schedule for f_i is generated using procedure `CpuSch`. As shown in Fig. 4.2, the f_i's output timings are obtained by delaying f_j's input timings by the path delay (`PthDel(`p`)`). `CpuSch` returns an available CPU core on n_1 where f_i can be scheduled between time interval $(T^{out}_{start}[f_i] - d_{proc}(f_i), T^{out}_{start}[f_i])$. Before returning, the placement, chaining and scheduling variables are updated.

Procedure `PthSchGrd` iterates over all the paths between n_1 and n_2 and

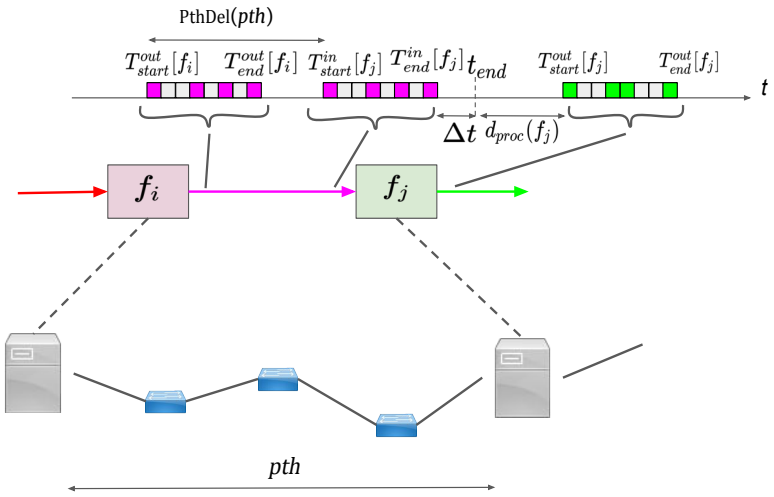


Figure 4.2: Illustration for the relationship between different VMF timing schedules.

returns with a feasible schedule, if it exists. More precisely, for path p, first a greedy schedule is allocated for the destination link $((p[-2], p[-1]))$ and it is checked whether it is compatible with the other links in p using procedure `PthSch`. If not, ϕ is returned; otherwise, `PthSch` returns the packet schedule for the whole path p. The current path p, the associated schedule s, and the start and end timings: t_{start}, t_{end} of s are returned.

Algorithm 4.1: Simplified pseudo-code for VMF-FG scheduling algorithm; a given VMF-FG $\mathcal{G} = (\mathcal{F}, \mathcal{L})$ is to be scheduled on the MFVi network $G^I = (N, E)$

```
1  α, γ, ω ← φ, φ, φ;
2  for vmf_snk ∈ F_snk do
3      Q_vmf ← {vmf_snk} ;   /* initializing the VMF queue for BFS */
4      while |Q_vmf| > 0 do
5          f ← Q_vmf.dequeue();
           /* schedule f and vlink (f, f_dw)                          */
6          for f_dw in DwstrNbrs(G, f) do
7              δn ← 0;
8              while VlinkSch(f,f_dw,bw_{f,f_dw}, α[f_dw] + δn, α[f_dw], α, γ, ω)
                do
9                  δn ← δn + 1;
           /* enqueue the upstream neighbours                         */
10         for nbr in UpstrNbrs(G, f) do
11             Q_vmf.enqueue(nbr)
```

Algorithm 4.2: Procedure for allocating a schedule for a virtual link

```
   /* VMF input output timings                                        */
   Global  : T^out_start, T^out_end, T^in_start, T^in_end
1  Procedure VlinkSch(f_i, f_j, bw_{f_i,f_j}, n_1, n_2, α, γ, ω):
2      t_end ← T^out_start[f_j] − d_proc(f_j);
3      p, s, t^in_start, t^in_end ← PthSchGrd(bw_{f_i,f_j}, n_1, n_2, ω, t_end);
4      if p == φ then
5          return False;
6      else
7          T^in_start[f_j], T^in_end[f_j] ← t^in_start, t^in_end;
8          T^out_start[f_i], T^end_in[f_i] ← t^in_start− PthDel(p), t^in_end− PthDel(p);
           /* get a core on n_1 for f_i scheduling                    */
9          core ← CpuSch(f_i, n_1, T^out_start[f_i] − d_proc(f_i), T^out_start[f_i]);
10         if core == φ then
11             return False
12         α[f_i], γ[(f_i, f_j)], ω[(f_i, f_j)] ← (n_1, core), p, s;
13         return True;
14 end
```

Algorithm 4.3: Procedure for chaining and scheduling a path.

```
   Global : |C|
   /* Hypercycle length                                          */
 1 Procedure PthSchGrd(bw, n1, n2, ω, t_end):
 2     t̄_end ← t_end%|C|;
 3     for p in Pths(n1, n2) do
 4         for Δt in [0, |C|] do
 5             t̄^new_end ← (t̄_end − Δt)%|C|;
 6             s_lnk, t_start, t_end ← GrdAlloc(bw, ω, p[−2], p[−1], t̄^new_end);
 7             if s_lnk ≠ φ then
 8                 s ← PthSch(s_lnk, p);
 9                 if s ≠ φ then
10                     return p, s, t_start, t_end;
11     return φ, φ, φ, φ;
12 end
```

4.4 Evaluations

In this section, we perform a few numerical experiments to evaluate the performance of our VMF-FG scheduling algorithm. First, we describe the setup used for evaluation. Next, the performance of our algorithm is evaluated in terms of the VMF-FG's delays. We also present the impact of media quality formats on resource utilization.

4.4.1 Evaluation setup

The MFVi physical network considered for the evaluation is the fat-tree data center topology because of its ease of management and flexibility to scale for high-quality media formats [2]. 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 is $\kappa^3/4$. We assume $\kappa = 8$, i.e, there are a total of 128 servers, with each server node containing 20 cores. All the devices (switches and server nodes) in the MFVi physical network are equipped with 10GbE interfaces.

We assume the media service request is represented using a VMF-FG as the one shown in Fig. 4.3 with 11 VMFs (and 6 endpoints) and 16 virtual links. The VMF-FG is considered in three media formats– (i) HD @ 30 fps, (ii) HD @ 60fps and (iii) FHD @ 30fps. The sub-sampling and sample encoding of the VMF-FG are assumed to be 4:2:2 and 10 bits, respectively.

The other parameters and their corresponding values (range) for media service requests, MFVi, and VMFs are listed in Table 4.2.

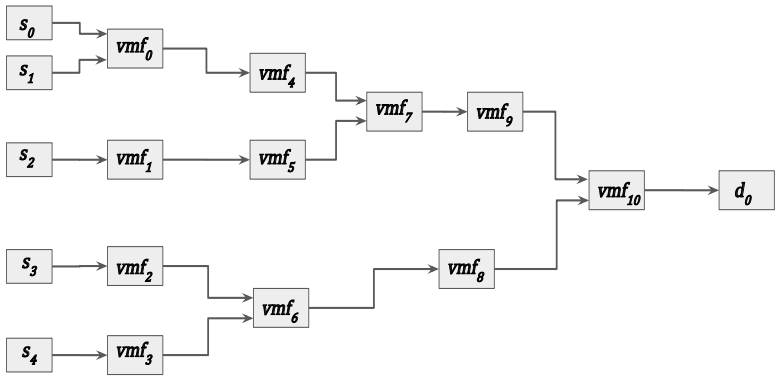


Figure 4.3: Illustration of the VMF-FG used for the evaluation.

Table 4.2: Default values/range of various parameters involved in the evaluation.

Parameter	Value or range	Units
Topology	Fat-tree ($\kappa = 8$)	-
Formats	HD@30fps, HD@60fps, FHD@30fps	-
CPU cores/node	20	-
Link bandwidth	10	Gbps
Cycle time (T_c)	{100, 200, 300, 400}	μs
VMF processing delay	5	ms
Node (switch) processing delay (d_{n_i,n_j})	40	μs
Links delay (d_n)	80	μs

4.4.2 End-to-end Delay

End-to-end delay is important for broadcast media services, especially for live production scenarios. Here, we report the end-to-end (E2E) delay in a VMF-FG, i.e., maximum delay among all the possible paths between the VMF-FG's sources and sinks. In Fig. 4.4 (a), we compare the average E2E delay for different cycle time (T_c) values. It can be observed that the E2E

delay increases with T_c. This results from the fact that the delay along the physical paths corresponding to the virtual links increases with T_c as the queuing delay in switches nodes increases.
We also report the E2E delay variation with M. The decomposition of a VMF-FG leads to VMF decomposition which results in the reduction of VMF processing delay. This results in the decreasing E2E VM-FG delay with M.
The are various components of E2E delay. The contribution due to video frame queuing is an important metric because it highlights the efficiency of VMF-FG scheduling. Here, we define the average VMF Queuing delay as the mean of VMF queuing delay in all the VMFs of the VMF-FG. The smaller the value of the average VMF queuing delay is, the better is the VMF-FG scheduling. The average queuing delay slightly decreases with T_c as shown in Fig 4.4 (b). However, VMF-FG decomposition results in increasing the average VMF queuing delay due to higher consolidation of VMFs per node.

4.4.3 Impact of formats

Here, we report the impact of media formats on the performance of scheduling algorithm. The format of media streams affects resource utilization efficiency. Fig. 4.5 compares the average numbers of cores used per node for three different video formats. With no VMF-FG decomposition ($M = 1$), the average core utilization decreases with higher-quality media format. The network bandwidth increasingly becomes the bottleneck with an increase in media quality thus fewer VMFs are placed on a node even if free CPU cores are available on the node.
It can also be observed that VMF-FG decomposition improves the average core utilization per node. By decomposing VMF-FG, bandwidth requirement per virtual link decreases thus increasing VMF consolidation per node.

4.5 Conclusion

The transition from specialized hardware to COTS IT infrastructure is taking place in broadcast studios, i.e., IP networking for media transport and general-purpose compute for media processing. MFV proposes to utilize COTS compute hardware to run media processing as VMFs that can be chained together in the form of VMF-FGs. To ensure that broadcast-quality guarantees in an MFV environment are met, it is important to not only map VMFs and virtual links to the underlying MFVi but also to timely schedule

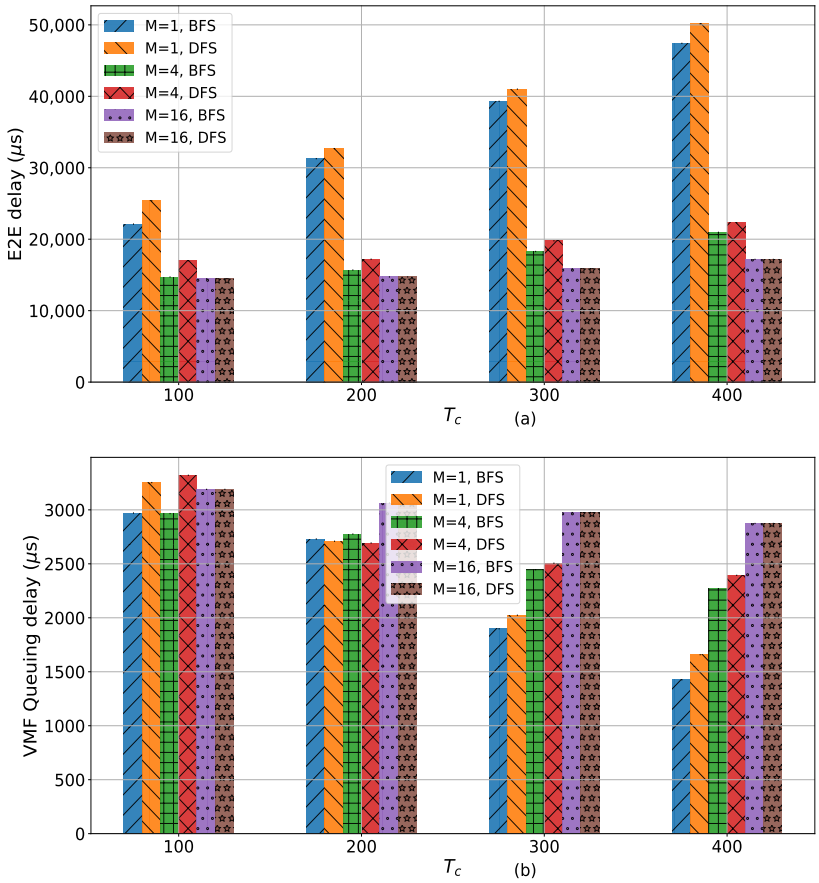


Figure 4.4: (a) The average E2E delay and (b) the average queuing delay versus cycle time (T_c).

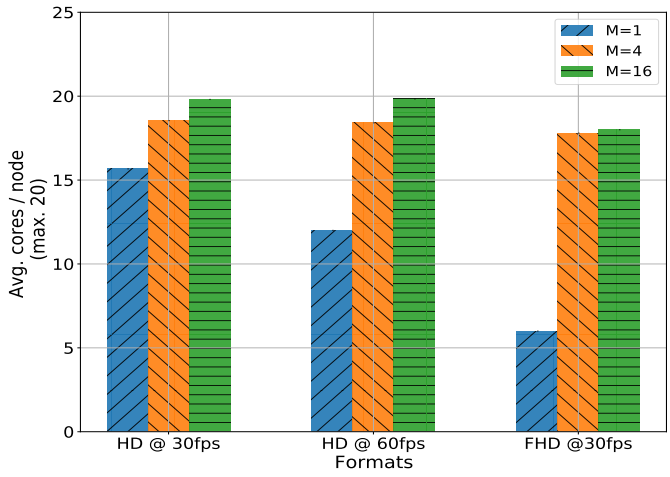


Figure 4.5: Impact of media formats on resource utilization for different M.

them. To this end, we first formulated the VMF-FG scheduling problem. We have then proposed a greedy-algorithm to find its solution. The evaluation of the algorithm shows an increase in the end-to-end delay with increasing cycle time. We also highlighted the improvement in the end-to-end delay with VMF-FG decomposition, particularly for high-quality formats. The implementation of media services in an MFV environment with VMF-FG scheduling can be a part of future work.

4.6 Acknowledgments

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.* Technical report, RedHat, 02 2018.

[2] 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, pages 1–16, 2014.

[3] K. Paulsen. *Prepping for the IP Transition.* Technical report, Dell EMC, 01 2017.

[4] *The Road to COTS and the Cloud for real-time broadcast production.* Technical report, Nevion, 01 2018.

[5] J. G. Herrera and J. F. Botero. *Resource allocation in NFV: A comprehensive survey.* IEEE Transactions on Network and Service Management, 13(3):518–532, 2016.

[6] N. Finn, P. Thubert, B. Varga, and J. Farkas. *Deterministic networking architecture.* draft-ietf-detnet-architecture-03 (work in progress), 2017.

[7] M. Chen, X. Geng, and Z. Li. *Segment Routing (SR) Based Bounded Latency.* Internet Engineering Task Force, Internet-Draft draft-chendetnet-sr-based-bounded-latency-00, 2018.

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

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

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

[11] B. Research and Development. *The IP network behind the R&D Commonwealth Games 2014 Showcase*, 2014. Available from: https://www.bbc.co.uk/rd/blog/2014-07-commonwealth-games-showcase-network.

[12] F. Poulin, P. Keroulas, S. Nyamweno, W. Vermost, P. Ferreira, and I. Kostiukevych. *How CBC/Radio-Canada Tested Media-over-IP Devices to Build its New Facility.* SMPTE Motion Imaging Journal, 129(4):35–44, 2020.

[13] D. Luzuriaga, C.-H. Lung, and M. Funmilayo. *Software-Based Video–Audio Production Mixer via an IP Network*. IEEE Access, 8:11456–11468, 2020.

[14] B. Research and Development. *Compositing and Mixing Video in the Browser*, 2018. Available from: https://www.bbc.co.uk/rd/blog/2017-07-compositing-mixing-video-browser.

[15] 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), pages 31–37. IEEE, 2020.

[16] *Joint routing and scheduling for large-scale deterministic IP networks*. Computer Communications, 165:33 – 42, 2021. Available from: http://www.sciencedirect.com/science/article/pii/S0140366420319642, doi:https://doi.org/10.1016/j.comcom.2020.10.016.

5

Routing and Scheduling for 1+1 Protected DetNet flows

The VMF-FG scheduling algorithm in the previous chapter exploits CSQF, a packet scheduling mechanism, to bound delays as well to prevent packet losses due to congestion in network nodes. In addition to the congestion in network, failures in the network can also cause packet losses. For many mission critical applications, disruption in the flow for even a few milliseconds can be detrimental to the performance of the application. This can be prevented by adding redundant paths between the communication endpoints of the application. Therefore, a dedicated protection mechanism like 1+1 *can ensure the communication between the endpoints continues in case of a failure in one of the network paths.*
This chapter addresses the 1+1 *routing and scheduling problem that combines* 1+1 *protection with packet scheduling to ensure high reliability along with worst-case end-to-end guarantees.*

⋆ ⋆ ⋆

G.P. Sharma, W. Tavernier, D. Colle, and M. Pickavet

Accepted in Computer Networks, 2022.

Abstract Deterministic Networking (DetNet) is attracting a lot of attention lately due to its ability to provide bounded latency and zero packet loss

for time-sensitive applications. In this paper, we formulate the routing and scheduling problem for *1+1* protected DetNet flows based on Cycle Specified Queuing and Forwarding (CSQF). The solution to this problem selects two paths between the two endpoints of each service request and schedules packet transmission on these paths meanwhile maximizing the accepted traffic. We have modeled the problem using Integer Linear Programming (ILP). We also propose two heuristic approaches: greedy and Tabu-search (TS) that can perform *1+1* routing and scheduling for a large number of requests in a reasonable time. Eventually, the performance of the ILP approach and heuristics is evaluated by performing simulation experiments. The results highlight the scalability of the two heuristics as compared to the ILP and superior performance of TS over greedy. The trade-off of cycle time on traffic acceptance and end-to-end delay is also presented.

5.1 Introduction

A wide range of real-time and safety-critical applications, such as connected cars, industrial control and media broadcasting, require deterministic performance from the network [1]. Proprietary bus-based technologies have been employed to build such networks that are currently hard to scale and manage. Although, packet-based networks can offer best-effort service they fail to provide any Quality of Service (QoS) guarantees. Non-deterministic queuing delay in Ethernet switches can result in unbounded jitter and occasional packet losses thus rendering traditional Ethernet networks incapable of handling the above-mentioned applications. For instance, even a single packet loss or excessive delay is not tolerable in TV broadcast production, as it would be detrimental to the viewing experience.

To be able to support time-sensitive applications, the network is required to provide a strict guarantee on latency, packet loss and jitter. The IEEE Time-sensitive Network (TSN) Task Group (TG) has developed a collection of standards to standardize the support of time-sensitive applications in Local Area Networks (LANs) by leveraging mechanisms such as traffic-shaping, frame preemption, priority scheduling, etc. As the work done by the TSN WG has focused on LANs, these mechanisms are not suitable for a network spanning multiple LANs. The IETF Deterministic Networking (DetNet) working group has been working on mechanisms that exploit L2 functionality in *Time Sensitive Networking* (TSN) so that zero packet loss along with deterministic latency and jitter can be achieved in L3. Particularly, the working group has described Cycle Specified Queuing and Forwarding (CSQF) based on Segment Routing to achieve deterministic performance [2] [3].

The end-to-end latency on a path between two points of a network can be bounded by capping the latency of each node (switch or router) in the path, as the latency component due to transmission delay and propagation delay is usually constant on a wired link. Cyclical queuing and dequeuing of packets in CSQF can be leveraged to limit the node latency. We have discussed the operation of CSQF in section 1.5.1. But it is worth mentioning here that by specifying packet transmission the end-to-end latency along any path can be bounded. Moreover, this also ensures that the queue occupancy in any node does not exceed an upper limit. Packet losses due to congestion thus can be prevented by using queues of appropriate lengths that are not overflowed.

Avoiding packet overflows in queues guarantees zero congestion losses; however, this is not the only cause for a packet loss. To provide protection against equipment (random media and/or memory) failures, dedicated protection for services is required. *1+1* protection has been a commonly exploited method to provide high reliability. The idea of *1+1* protection is to use two paths to route packets between the source and destination as opposed to a single path; the destination selects packets from one path only. In case of a node/link failure on one of the paths, the destination can just start receiving packets from the other path. Due to this simplistic architecture, this scheme has been popular in optical networks (SONET/SDH), where the destination switches to one of the paths based on the switching criteria, e.g., signal strength [4]. However, *1+1* protection cannot be naively used for DetNet flows; packet scheduling needs to be performed along the two paths such that packets can be reliably recovered in case of a failure, as discussed in the next section.

Deploying *1+1* protected DetNet flows entails two components– (1) routing or selecting two paths between the source and destination and (2) generating reliable packet schedules along these paths. The problem of joint routing and scheduling in DetNets using CSQF has been investigated in [5]. To the best of our knowledge, modeling of protection schemes in DetNets has not been addressed yet. Therefore, we address the Routing and Scheduling (RTSCH) problem for *1+1* protected DetNet flows by formulating an Integer Linear Program (ILP) for it. Additionally, we propose two heuristic algorithms to solve the above problem instances of reasonable size.

The rest of the paper [1] is organized as follows. The technical background regarding *1+1* protection for DetNet flows is provided in section 5.2. Section 5.3 details the related works for this paper. The system model and ILP formulation for the *1+1* DetNet protection problem are described in

[1] The section in the paper on CSQF is contained in section 1.5.1 of the introduction chapter.

section 5.4. In section 5.5, we have discussed (a) greedy based heuristic and (b) Tabu-search based heuristic. Section 5.6 presents the performance evaluation of the ilp and two heuristics in terms of various metrics. Finally, the paper is concluded in section 5.7.

5.2 CSQF and *1+1* protection

DetNet services avoid packet losses resulting from contention between different flows by traffic regulation combined with careful packet scheduling. For packet scheduling, a queuing mechanism like CSQF can be exploited to provide guarantee zero packet loss and end-to-end delay.
Packet scheduling is not enough for DetNet services as these services can still be disrupted due to failures of a network node (e.g., memory error) or a link (e.g., fiber break). Dedicated protection schemes, e.g., *1+1*, have been widely used in packet-based networks [4] to protect against such failures. In *1+1* protection, an extra path, in addition to the original path, is used to route packets between the source and destination. At the destination end, packets received on the two paths are de-duplicated.
Due to different end-to-end delays along the two paths, the reliability of *1+1* protection can be impacted. This phenomenon is demonstrated in Fig. 5.1. The Packet Replication Function (PRF) present on the source is responsible for duplicating the packets, that are received on its input interface onto its two output interfaces that are subsequently routed through two different paths p_1 and p_2. At the destination, the Packet Elimination Function (PEF) performs packet de-duplication/elimination on the packets received on the last links $(p_1[-2], p_1[-1])$ and $(p_2[-2], p_2[-1])$ of paths p_1 and p_2, respectively. Packet de-duplication entails the presence of sequencing information with packets that can be done by adding a sequence number or timestamp to the packet while doing duplication at the PRF.
The above mentioned functions for *1+1* protection can be implemented via IP encapsulation/decapsulation of the DetNet flows. For instance, [7] proposed to use P4 match-action tables to implement the PRF and the PRE at the source and destination nodes, respectively. As this paper is not concerned with the actual implementation of these functions, we will next explain only their high-level logic. The PRF simply adds the packet sequence number to the packet header and then duplicates the packet to send them over the two interfaces of the source node. With packet sequence information, packet elimination becomes quite straightforward. The overview of PEF logic is shown in Fig. 5.1 (b). The PEF remembers the highest sequence number seq_{lst} of the last output cycle and only outputs ($output(pkts)$) the packets of this cycle if its highest sequence number

seq_{pkts} is higher than seq_{lst}. Next, the PEF updates seq_{lst} to seq_{pkts}. In case, if $seq_{pkts} < seq_{lst}$, the packets of this cycle are discarded. The simple logic of a PEF allows it to perform packet elimination without contributing a significant latency and excessive buffering.

With such a PEF at the destination, a reliable packet elimination is only

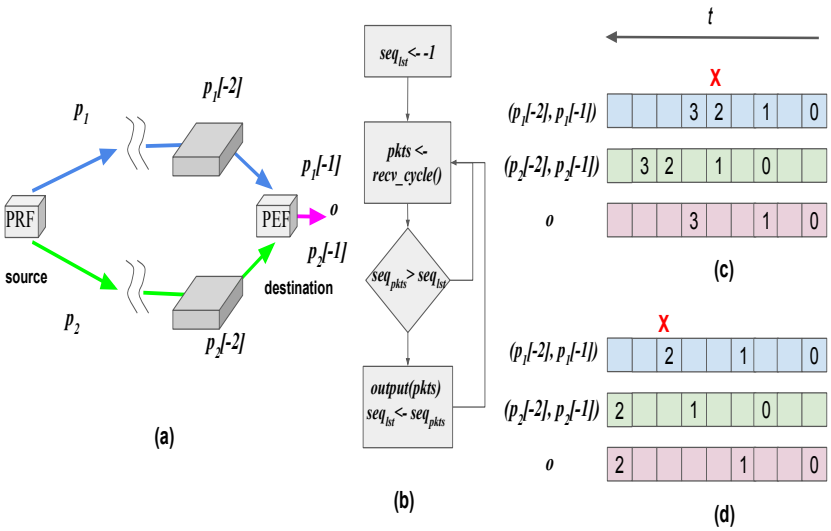


Figure 5.1: Reliable recovery with 1+1 CSQF. (a) 1+1 protection, (b) PEF logic, (c) Naive scheduling and (d) End-to-end delay aware scheduling.

possible if the packet transmission schedules on the two paths follow the *spacing constraint*. Let's assume that each cycle has bandwidth to send one packet; the PRF sends four packets in four cycles over the two disjoint paths, where the end-to-end latency of p_2 is higher than p_1 as shown in Fig. 5.1(c). The packet schedules on links $(p_1[-2], p_1[-1])$ and $(p_2[-2], p_2[-1])$ and the output o of the PEF is shown. The PEF receives $pkt_0, pkt_1, pkt_2, pkt_3$ on p_1 during cycles $0, 2, 4, 5$ and on p_2 during cycles $2, 4, 6, 7$, respectively. Supposing pkt_2 is lost somewhere on p_1, the PEF is not able to recover it from p_2, as the next packet received by the PEF with $seq > 1$ (after recovering pkt_1) is pkt_3 on p_1, though pkt_2 is later received on p_2.

The above situation can be avoided by carefully scheduling packet transmissions on the *1+1* paths. The *spacing constraint* on packet schedules ensures that any two cycles in the packet schedule with non-zero bandwidth allocation must be separated by at least the absolute value of the difference between the end-to-end delays of the two paths. This constraint ensures

reliable recovery as shown in Fig. 5.1 (d) where the new schedules adheres to the spacing constraint.
It is worth noting that reliable recovery is possible without satisfying the spacing constraint, though that requires an additional component: the Packet Re-ordering Function (PROF). The PROF would require additional buffering to store the received packets on the two paths and then perform de-duplication. Though, de-duplication can be performed efficiently by maintaining a heap data structure for the received packets, where each heap insertion operation has $O(log(n))$ complexity; nonetheless, the buffering and additional computation could contribute a significant latency to the packet recovery process. In this paper, we therefore assume that the destination does not perform packet re-ordering.

5.3 Related Works

The attempts to make networks more deterministic have been going on for several years. The standardization work by the TSN task group in IEEE 802.1 started in 2015. These standards have sought to guarantee bounded latency and high reliability in Layer 2. For reliability, the group proposed IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER) as a new Ethernet sub-protocol [8]. To extend these guarantees to Layer 3, IETF DetNet TG has proposed various mechanisms [9].
With the coming up of TSN standards, researchers have shown keen interest in developing algorithms for the deployment of time-sensitive applications. Numerous studies have focused on generating schedules for time-triggered traffic in IEEE 802.1Qbv/TSN such that a given objective function is optimized. Craciunas et al. have used Satisfiability Modulo Theory (SMT) to optimize the network as well as application schedules in a switched multispeed time-triggered network [10]. In [11], the authors have formulated the problem of routing and scheduling of time-triggered traffic in TSNs as an ILP to provide real-time guarantees by exploiting the logical centralization paradigm of software-defined networking. Approaches based on SMT or ILP generate exact solutions by expressing the scheduling problem as a constrained optimization or constrained satisfaction problem that is solved by ILP or SMT solvers. The advantage of using exact approaches is that they generate provable optimal solutions. On account of scheduling problems being NP-hard, exact approaches do not scale well to flow numbers and network sizes. Alternatively, heuristic and metaheuristic approaches solve this problem by compromising on the quality of the solution for execution speed. Tabu search in [12] and incremental backtracking in [13] and [14]

have been proposed to speed up the joint RTSCH problem with close to an optimal solution.
Recently, the RTSCH problem in CSQF-based networks has also been addressed in various studies. In [5], the joint routing and scheduling problem for DetNets is formulated using ILP and two methods–column generation and dynamic programming, are presented to maximize traffic acceptance while solving the problem. Moreover, a scalable greedy algorithm is also proposed that is capable of solving the problem with an small gap. Similarly, the authors in [15] have modeled load balancing in DetNets where time-sensitive flows are split over multiple paths in order to improve network utilization.
Reactive mechanisms such as Dynamic Multipath Optimization (DMPO), through the use of continuous network monitoring and flow prioritization, can provide a level of reliability by re-routing the critical traffic during an interruption [16]. However, the restoration time in DPMO spans hundreds of milliseconds to seconds; thus, an interruption can still be detrimental for Detnet flows. Dedicated path protection schemes like *1+1* have been studied in relation to Elastic Optical Networks (EONs). Walkowiak et al. have formulated the offline problem of routing and spectrum allocation (RSA) to dedicated path protection (DPP) in EONs as an ILP. Tabu-search-based metaheuristics were proposed to solve the problem instances of reasonable size.
These works are limited largely to the optical networks and hence do not address the problem of dedicated protection in DetNets. In contrast to these studies, our work focuses on the problem of *1+1* protection for DetNet flows. We assume the nodes in the given DetNet are CSQF capable and the source and the destination are capable of performing the necessary operations to provide *1+1* protection.

5.4 System Model and ILP Formulation

Next, we describe the system model and then formulate the *1+1* RTSCH problem as an ILP. As discussed earlier, the time at each node's output port is divided into intervals of equal duration T_c, referred to as *cycles*. The number of cycles after which the traffic pattern of the application repeats is referred to as *hypercycle*; therefore, it is sufficient to consider the problem for one hypercycle only. Moreover, it can be assumed without loss of generality that cycles on each node are aligned, i.e, start at the same time.
Physical infrastructure: The deterministic network infrastructure is represented using a directed graph $G = (N, E)$, where N is the set of CSQF-capable switches or routers and E is the set of physical links connecting

the nodes in N. For each link $e = (n_i, n_j) \in E$ the cycles in a hypercycle are denoted by C and the available bandwidth for each cycle $c \in C$ denoted by $bw_{n_i,n_j,c}$. For instance, on an unused 10G link during each cycle of T_c=20μs, a total of 25000B or 16x1500B packets are available. The total delay introduced on the link (n_i, n_j) is represented by d_{n_i,n_j}, that includes the propagation delay, transmission delay and processing delay due to n_j.
Requests: A set of requests R needs to be RTSCH'ed with *1+1* protection on G. Each request $r \in R$ has a corresponding five-valued tuple $\mathbb{T}^r = (n_s^r, n_d^r, bw^r, D_r^{e2e}, \delta D_r^{e2e})$ that contains the metadata required for RTSCH'ing r. Here, n_s^r and n_d^r are the r's source and destination between which two DetNet flows are required; bw^r denotes the r's bandwidth requirement during one hypercycle; D_r^{e2e} and δD_r^{e2e} are the maximum end-to-end delay and jitter permissible for request r. Fig. 5.2 shows $\mathbb{T}^r$ for request r, $n_s^r = n_0$, $n_d^r = n_9$ and $bw^r = 64$ packets(=92kB); for $T_c = 20\mu$s and $|C| = 50$, r requires full four cycles out of fifty cycles along each link of p_1 and p_2.
Variables: Binary variable γ_p^r indicates the routing of $r \in R$ using $p \in P^{n_s^r,n_d^r}$; $\gamma_p^r = 1$, if path p is used for the routing of r; otherwise $\gamma_p^r = 0$. Continuous decision variable $\omega_{p,n_i,n_j,c}^r$ denotes the amount of bandwidth allocated to request r on physical link $(n_i, n_j) \in p, p \in P^{n_s^r,n_d^r}$ during cycle $c \in C$. In addition to binary variables γ_p^r and $\omega_{p,n_i,n_j,c}^r$, we next introduce a few helpers or dependent variables that are required in the ILP formulation. Binary variable λ^r indicates if exactly two disjoint paths are allocated for the routing of r. $\rho_{p,n_i,n_j,c}^r$ is a binary variable that indicates if non-zero bandwidth is allocated to request $r \in R$ on link $(n_i, n_j) \in p, p \in P^{n_s^r,n_d^r}$. Binary variable $\eta_{p,n_i,n_j,c_1,c_2}^r = 1$, if at least one of ρ_{p,n_i,n_j,c_1}^r and ρ_{p,n_i,n_j,c_2}^r is 0; otherwise $\eta_{p,n_i,n_j,c_1,c_2}^r = 0$.
Table 5.1 lists the parameters and variables involved in the system model along with a short description.
In the *1+1* RTSCH problem, given the set of requests R and DetNet topology G, requests must be routed and scheduled through two disjoint paths. Fig. 5.2 shows an illustration where the controller takes as input DetNet topology G along with request set R and solves the *1+1* RTSCH problem. The controller itself can be distributed / replicated across multiple instances for robustness and scalability reasons [17, 18] . Nevertheless, we can assume a single logical controller that is responsible for routing and scheduling Detnet flows. The solution of the *1+1* RTSCH problem, i.e., routes and schedules for requests can then be used to configure the source nodes with SID label stacks, as discussed earlier.

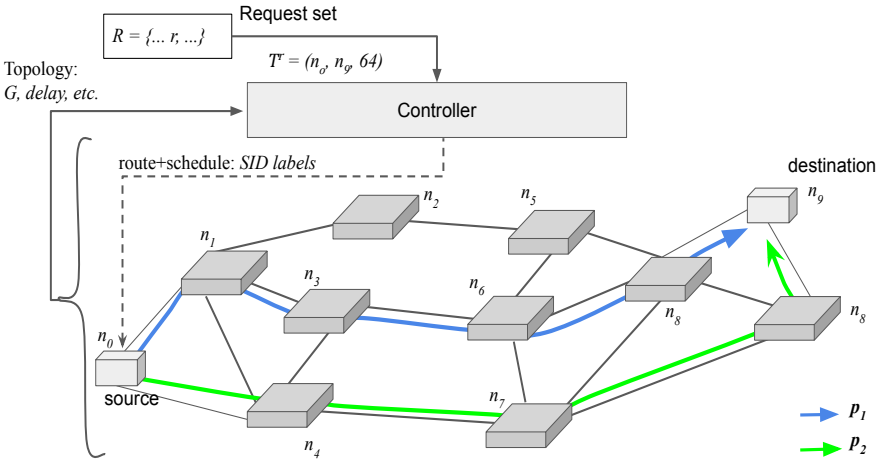


Figure 5.2: Illustration of the 1+1 *RTSCH problem.*

5.4.1 ILP Formulation

In this section, the *1+1* RTSCH problem is formulated as an optimization problem. The goal is to find two disjoint paths between the sender and receiver of each request as well as schedule packet transmission along the two paths meanwhile optimizing an objective function, e.g., maximizing the total accepted traffic, minimizing the jitter or end-to-end delay on the two paths.

The DetNet *1+1* protection problem is formulated as a Integer Linear Program (ILP). The objective of the formulation is to maximize the total accepted traffic for *1+1* RTSCH'ing.

$$obj : \max \sum_{r \in R} \lambda^r bw^r \tag{5.1}$$

The constraints for this ILP formulation are listed from (5.2) to (5.13). Constraint pair (5.2-5.3) ensures exactly two paths are selected from $P^{n_s^r, n_d^r}$ for the request mapping, where λ^r indicates *1+1* routing. With this constraint pair, λ^r is 0 if $\sum_{p \in P^{n_s^r, n_d^r}} \gamma_p^r = 0$ and is 1 if exactly two paths are selected.

$$\sum_{p \in P^{n_s^r, n_d^r}} \gamma_p^r \geq 2\lambda^r, \forall r \in R. \tag{5.2}$$

$$\lambda^r \geq \gamma_p^r, \forall r \in R, \forall p \in P^{n_s^r, n_d^r}. \tag{5.3}$$

The constraint (5.4) enforces disjointedness on the selected paths. In this constraint, if selected paths p_1 and p_2 are disjoint, i.e, $disj(p_1, p_2) = 1$,

Table 5.1: Description of the notations used for different parameters and variables involved in the system model.

Notation	Description
$G = (N, E)$	Directed graph representation of the DetNet infrastructure, where N is the set of nodes and E is the set of physical links between the nodes.
N_e	$N_e \subset N$ is the set of all endpoints, i.e, sources and destinations.
C	The set of all cycles in a hypercycle.
$bw_{n_i,n_j,c}$	Available bandwidth (in bytes) on physical link $(n_i, n_j) \in E$ during cycle $c \in C$.
$d^{tx}_{n_i,n_j}$	Transmission delay on link (n_i, n_j).
d^{proc}_{n}	Maximum processing delay on node n.
d_{n_i,n_j}	Total delay on link (n_i, n_j); $d_{n_i,n_j} = d^{tx}_{n_i,n_j} + d^{proc}_{n_j}$.
R	Set of all requests to be RTSCH'ed on G.
$\mathbb{T}^r$	Request tuple $\mathbb{T}^r = (n^r_s, n^r_d, bw^r, D^{e2e}_r, \delta D^{e2e}_r)$ corresponding to $r \in R$, where n^r_s, n^r_d, bw^r, D^{e2e}_r and δD^{e2e}_r are the source, destination, bandwidth requirement (in bytes) of r, maximum permissible end-to-end delay and jitter, respectively.
$P^{n^r_s,n^r_d}$	The set of all paths between the source n^r_s and the destination n^r_d of $r \in R$.
γ^r_p	Binary variable indicates the routing of $r \in R$ on $p \in P^{n^r_s,n^r_d}$.
λ^r	Binary (helper) variable indicates the *1+1* routing of $r \in R$.
$\omega^r_{p,n_i,n_j,c}$	Bandwidth (in bytes) allocated to request $r \in R$ on link $(n_i, n_j) \in p$, $p \in P^{n^r_s,n^r_d}$, during cycle interval $c \in C$.
$\rho^r_{p,n_i,n_j,c}$	Binary (helper) variable indicates if non-zero bandwidth is allocated to request $r \in R$ on link $(n_i, n_j) \in p$, $p \in P^{n^r_s,n^r_d}$, during cycle interval $c \in C$.
$\eta^r_{p,n_i,n_j,c_1,c_2}$	Binary (helper) variable $\eta^r_{p,n_i,n_j,c_1,c_2}$ equals 1, if at least one of ρ^r_{p,n_i,n_j,c_1} and ρ^r_{p,n_i,n_j,c_2} is 0; otherwise $\eta^r_{p,n_i,n_j,c_1,c_2} = 0$.

then the LHS (=1) equals the RHS (=1); however, if they are non-disjoint, i.e, $disj(p_1, p_2) = 0$, then the LHS (=1) is greater than the RHS (=0), which shall be prevented by the constraint.

$$\gamma^r_{p_1} + \gamma^r_{p_2} - 1 <= disj(p_1, p_2), \forall r \in R, \forall p_1, p_2 \in P^{n^r_s,n^r_d}. \tag{5.4}$$

The number of packets (bandwidth) allocated to a request during a

cycle must be conserved, i.e, the total packets transmitted in a cycle at a link should be equal to the total packets transmitted on its downstream link in the cycle shifted by the link delay. Mathematically, the amount of bandwidth allocated on a link (n_i, n_j) of path p during cycle c is equal to the bandwidth allocated on downstream link (n_j, n_k) on p during cycle $(c + d_{n_j,n_k})\% \mid C \mid$ as expressed in constraint (5.5). Fig. 5.3 illustrates packet scheduling along the network path p. The packet schedule on link (n_0, n_1) is delayed by the link delay $d_{n_0,n_1} = 5T_c$ to get the schedule for (n_1, n_2) which is delayed by $d_{n_1,n_2} = 4T_c$ to get the schedule for (n_2, n_3). Cycles 5 and 7 on link (n_1, n_2) and cycles $9 \equiv 1(\%8)$ and $11 \equiv 3(\%8)$ on link (n_2, n_3), have bandwidth equal to cycles 0 and 2 on link $(n0, n1)$.

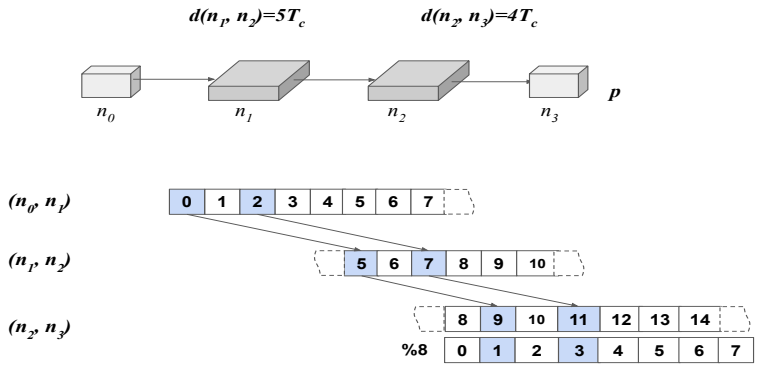


Figure 5.3: Relationship of packet schedules on various links of a path.

$$\omega^r_{p,n_i,n_j,c} = \omega^r_{p,n_j,n_k,(c+d_{n_j,n_k})\%|C|} \\ ,\forall r \in R, \forall p \in P^{n^r_s,n^r_d}, \forall (n_i, n_j), (n_j, n_k) \in p, \forall c \in C. \tag{5.5}$$

The sum of bandwidth allocated to all requests during a cycle cannot exceed the available bandwidth in the cycle; this condition is ensured by constraint (5.6).

$$\sum_{r \in R} \omega^r_{p,n_i,n_j,c} \leq bw_{n_i,n_j,c},\ (n_i, n_j) \in E, \forall c \in C. \tag{5.6}$$

The total bandwidth allocated in hypercycle C should be equal to the bandwidth requirement of r; this is guaranteed by constraint (5.7).

$$\sum_{c \in C} \omega^r_{p,n_i,n_j,c} = bw^r \gamma^r_p,\ \forall r \in R, \forall p \in P^{n^r_s,n^r_d}, \forall (n_i, n_j) \in p. \tag{5.7}$$

The pair of constraints in (5.8a-5.8b) ensures that the same schedule is used on the first link ($(p_1[0], p_1[1])$ and $(p_2[0], p_2[1])$) of the two selected paths.

$$\omega^r_{p_1,p_1[0],p_1[1],c} \geq \omega^r_{p_2,p_2[0],p_2[1],c} + (\gamma^r_{p_1} + \gamma^r_{p_2} - 2)\omega_{max}, \quad \forall r \in R, \forall p_1, p_2 \in P^{n^r_s,n^r_d}, \forall c \in C. \tag{5.8a}$$

$$\omega^r_{p_1,p_1[0],p_1[1],c} \leq \omega^r_{p_2,p_2[0],p_2[1],c} - (\gamma^r_{p_1} + \gamma^r_{p_2} - 2)\omega_{max}, \quad \forall r \in R, \forall p_1, p_2 \in P^{n^r_s,n^r_d}, \forall c \in C. \tag{5.8b}$$

As discussed in section 5.2, at the receiver, any two cycles in the schedule with non-zero bandwidth should be spaced by at least the difference of the end-to-end delays along the two selected paths. The pair of constraints in (5.9a-5.9b) enforces the minimum spacing ($|D_{p_2} - D_{p_1}|$) on the schedules of p_1 and p_2 near the receiver end ($(p_1[-2], p_1[-1]), (p_2[-2], p_2[-1])$). For path p_1, when two cycles c_2 and c_1 have minimum bandwidth allocated, i.e, $n^r_{p_1,p_1[-2],p_1[-1],c_1,c_2} = 0$, then the LHS of the constraint pair ($c_2 - c_1$) is at least $|D_{p_2} - D_{p_1}|$. In other words, if $D_{p_2} \geq D_{p_1}$, then (5.9a) is active while (5.9b) is inactive and if $D_{p_2} \leq D_{p_1}$, then (5.9b) is active while (5.9a) is inactive. If at least one of $\rho^r_{p_1,n',n^r_d,c_1}$ and $\rho^r_{p_1,n',n^r_d,c_2}$ is 0 ($n^r_{p_1,p_1[-2],p_1[-1],c_1,c_2} = 1$), the pair of constraints is inactive.

$$c_2\rho^r_{p_1,p_1[-2],p_1[-1],c_2} - c_1\rho^r_{p_1,p_1[-2],p_1[-1],c_1} \geq (D_{p_2} - D_{p_1}) - C_{max}\eta^r_{p_1,p_1[-2],p_1[-1],c_1,c_2}, \quad \forall r \in R, \forall p_1, p_2(\neq p_1) \in P^{n^r_s,n^r_d}, (p_1[-2], p_1[-1]) \in p_1, \forall c_1, c_2(> c_1) \in C. \tag{5.9a}$$

$$c_2\rho^r_{p_1,p_1[-2],p_1[-1],c_2} - c_1\rho^r_{p_1,p_1[-2],p_1[-1],c_1} \geq (D_{p_1} - D_{p_2}) - C_{max}\eta^r_{p_1,p_1[-2],p_1[-1],c_1,c_2}, \quad \forall r \in R, \forall p_1, p_2(\neq p_1) \in P^{n^r_s,n^r_d}, (p_1[-2], p_1[-1]) \in p_1, \forall c_1, c_2(> c_1) \in C. \tag{5.9b}$$

The binary variable $\rho^r_{p,n_i,n_j,c}$ in constraint (5.10) is 0 when $\omega^r_{p,n_i,n_j,c} = 0$ and is 1, if at least ω^r_{min} is allocated.

$$\omega^r_{min}\rho^r_{p,n_i,n_j,c} \leq \omega^r_{p,n_i,n_j,c} \leq \omega^r_{max}\rho^r_{p,n_i,n_j,c}, \quad \forall r \in R, \forall p \in P^{n^r_s,n^r_d}, (n_i, n_j) \in p, \forall c \in C. \tag{5.10}$$

For Detnet flows, it is required that the packets transmitted by n^r_s are received by n^r_d within a permissible delay D^{e2e}_r. With CSQF, the worst-case

delay along a path p can be expressed as $\sum_{(n_i,n_j)\in p} d_{n_i,n_j}$ [19]. The constraint in (5.11) imposes this condition on the two selected paths for each request. As far as the jitter with CSQF is concerned, its value is dependent on the spacing between any two cycles with non-zero bandwidth. The maximum (intra-cycle) jitter is independent of the chosen path and is equal to $2T_c$. Therefore, the cycle time should be chosen such that $T_c \leq D_r^{e2e}$.

$$\gamma_p^r \sum_{(n_i,n_j)\in p} (d_{n_i,n_j}) \leq D_r^{e2e}, \forall r \in R, \forall p \in P^{n_s^r,n_d^r}. \tag{5.11}$$

The constraint trio in (5.12a, 5.12b, 5.12c) ensures $\eta^r_{p,n_i,n_j,c_1,c_2}$ is 0 only if both ρ^r_{p,n_i,n_j,c_1} and ρ^r_{p,n_i,n_j,c_2} are 0 and is 1, otherwise. Variables γ_p^r, λ^r, $\rho^c_{p,n_i,n_j,c}$ and $\eta^r_{p,n_i,n_j,c_1,c_2}$ can only take binary values (0 or 1) while $\omega^r_{p,n_i,n_j,c}$ is bounded; these limits are expressed in constraint (5.13).

$$\eta^r_{p,n_i,n_j,c_1,c_2} \geq (1 - \rho^r_{p,n_i,n_j,c_1}), \forall r \in R, \forall p \in P^{n_s^r,n_d^r}, (n_i,n_j) \in p, \forall c \in C. \tag{5.12a}$$

$$\eta^r_{p,n_i,n_j,c_1,c_2} \geq (1 - \rho^r_{p,n_i,n_j,c_2}), \forall r \in R, \forall p \in P^{n_s^r,n_d^r}, (n_i,n_j) \in p, \forall c \in C. \tag{5.12b}$$

$$\eta^r_{p,n_i,n_j,c_1,c_2} \leq (2 - \rho^r_{p,n_i,n_j,c_1} - \rho^r_{p,n_i,n_j,c_2}), \forall r \in R, \forall p \in P^{n_s^r,n_d^r}, (n_i,n_j) \in p, \forall c \in C. \tag{5.12c}$$

$$\gamma_p^r \in \{0,1\}, \lambda^r \in \{0,1\}, \eta^r_{p,n_i,n_j,c_1,c_2} \in \{0,1\}, \omega^r_{p,n_i,n_j,c} \in [0, \omega^r_{max}], \rho^r_{p,n_i,n_j,c} \in \{0,1\}\, \forall r \in R, \forall p \in P^{n_s^r,n_d^r}, \forall (n_i,n_j) \in p, \forall c, c_1, c_2 \in C. \tag{5.13}$$

5.5 Heuristics

The ILP approach, presented in the previous section, solves the *1+1* RTSCH problem by navigating through the solution space consisting of all combinations of variable values. As the total number of variables in the formulation increases drastically with the size of the problem the resulting solution space is huge. Therefore, we next present two heuristics that have relatively low execution times and provide sub-optimal but decent solutions.

5.5.1 Greedy Heuristic

The pseudocode for the greedy heuristic to solve the *1+1* RTSCH problem is shown in Alg. 5.1. The procedure `batch_rt_sch` is called with the set of

requests R arranged in the decreasing order of bw^r values. Thus, `greedy_rt_sch` is called first for r with the highest bw^r value and last for r with the lowest bw^r value (l. 36).

In `greedy_rt_sch`, first, all paths between the two endpoints n_s^r and n_d^r are generated and stored in $P^{n_s^r,n_d^r}$ (l. 17). We make use of Networkx's all_simple_paths procedure to generate the paths between the given endpoints [20]. The algorithm utilizes a modified depth-first search for path computation and runs in $O(N+E)$ time for each path.

Next, a pair of *for* loops is used to select the first two paths on which routing and scheduling is feasible.

For a path-pair $(p_1, p_2) \in P^{n_s^r,n_d^r} P^{n_s^r,n_d^r}$ pair, it is checked whether p_1 and p_2 are disjoint and if enough cycles are available to generate a feasible schedule; otherwise, the next (p_1, p_2) pair is considered (l. 18). Consider for example a path pair p_1 and p_2, a hypercycle with eight cycles ($|C| = 8$) and each cycle with a bandwidth of 16 (1500B) packets, if the difference in the end-to-end delay along p_1 and p_2 is 4 cycles then the requests with $bw^r > 16\lceil 8/2 \rceil = 32$ cannot be routed on p_1 and p_2, irrespective of the schedule. However, if the difference in the end-to-end delay is 3 cycles, then only the requests with $bw^r > 48$ are rejected. This step ensures that next step (schedule computation) is not executed if scheduling is not possible for the selected path pair p_1 and p_2.

Using `gen_sch`, a schedule is generated at the destination end ($(p_1[-2], p_1[-1])$) starting with cycle c_{strt} (l. 24). The procedure `gen_sch` greedily attempts to allocate bw^r among C cycles of $(p_1[-2], p_1[-1])$, with start cycle c_{strt} and a minimum cycle spacing of δc (l. 1-14). If such schedule exists ($sch_{p_1,d} \neq \phi$) at the destination end, then using procedure `sch_pth` with direction argument $dir = -1$ (from destination to source), the schedule for all the links in p_1 is generated by (negatively) delaying the destination schedule $sch_{p_1,d}$. This is required because of the constraint (5.5), as discussed in section 5.4. If such a schedule can be generated, i.e, bandwidth required in the corresponding cycles along all the links on p_1 is available, the schedule for p_1 is stored in ω_{p_1}. Again, for the second path, by calling procedure `sch_pth` with the schedule at the source end of p_2, i.e., $\omega_{p_1}[(p_1[0], p_1[1])]$, and the direction argument $dir = +1$ (from source to destination), the schedule for all the links of p_2 is generated and stored in ω_{p_2}. In case a schedule is found, the schedule $\omega_{p_1} \cup \omega_{p_2}$ along the two paths is returned; otherwise the next (p_1, p_2) pair is checked.

If a request is successfully RTSCH'ed, i.e., $\omega^r \neq \phi$ (l. 37), the link bandwidth in the corresponding cycles along the two paths is updated using procedure `upd_bw`. Moreover, the total accepted traffic ($objval$) is updated and the request schedule is stored in sch.

The path computation step of the heuristic has the time complexity of $O(|E||P|)$ whereas the schedule generation step runs in $O(|E||P||C|^2)$ time. Thus, for a network topology of realistic size the schedule generation step dominates the overall execution time of the greedy heuristic.

5.5.2 Tabu-search Heuristic

On the one hand, the greedy heuristic is fast and returns a suboptimal solution, but on the other hand, though the ILP solution is optimal it takes a lot of time to compute. There exists a third approach to generate a close-to-optimal solution in a reasonable time. By employing a metaheuristic such as Tabu-search (TS) the substantial solution space can be navigated much more efficiently.

The flowchart for TS-based heuristic to solve *1+1* RTSCH problem is shown in Fig. 5.4. Similar to the greedy heuristic, the TS heuristic is called with the set of requests R, request tuples $\mathbb{T}$ and infrastructure graph G. First, `rt_sch` is called to RTSCH with `srtd`(R) and the resulting greedy solution is captured in *init_soln* object and the total accepted traffic for this solution is stored in *bst_objval*. Before proceeding further, the variable initialization is done.

The core of the TS heuristic is a *while* loop that runs for $ITRN_{max}$ iterations and tries to improve the current solution *init_soln* taking into account the recent moves, as well as the *tabu* moves. A move to one of the neighbour of *init_soln* is made using procedure `nbr_soln` (explained in the next paragraph). After generating the neighbouring solution *new_soln*, it is checked whether *new_soln* is better than *bst_objval*, i.e, the total accepted traffic for the neighbouring solution *new_soln.objval* is more than *bst_objval*. If yes, the move is added to the collection of tabu moves along with the tenure TT. This results in the prohibition of undertaking *move* for the next TT iterations. If *new_soln* is worse than the best score, the move is stored in *moves_used* so that it is not performed again while the search through the current neighbourhood is carried out. In case no better solution is found for $NOIMP\_ITRN_{max}$ iterations, the best score is relaxed by a factor of WF so that the search in another neighbourhood can be performed from the next iteration.

The pseudocode for the procedure `nbr_soln` is shown in Alg. 5.2. The two kinds of moves that can be undertaken in `nbr_soln` are: i) the request swap move and ii) the path change move, chosen with probability pr_{req} and $1 - pr_{req}$, respectively. If the request swap move is chosen, the order of requests $soln.R$ is modified by swapping two randomly selected requests $soln.R[i]$ and $soln.R[j]$. Otherwise, if the path change move is chosen, a request is randomly selected from $soln.R$ it is re-routed and re-scheduled us-

Algorithm 5.1: Procedure for the greedy-based heuristic to solve the *1+1* RTSCH.

Procedure `gen_sch`(c_1, bw^r, p, δc, $G = (N, E)$)**:**
  $bw_{alc}, sch, c_{nxt} \leftarrow 0, \phi, c_1$;
  for c **in** $[c_1, C-1]$ **do**
    if $c == c_{nxt}$ **then**
      $bw_c \leftarrow \min(bw^r - bw_{alc}, bw^c_{p[-2],p[-1]})$;
      if $bw_c > 0$ **then**
        $bw_{alc} \leftarrow bw_{alc} + bw_c$;
        $sch[p, p[-2], p[-1], c] \leftarrow bw_c$;
        $c_{nxt} \leftarrow c_{nxt} + \delta c$;
      else
        $c_{nxt} \leftarrow c_{nxt} + 1$;
    if $bw_{alc} >= bw^r$ **then**
      return sch;
  return ϕ;
Procedure `greedy_rt_sch`($\mathbb{T}^r = (n^r_s, n^r_d, bw^r)$, $G = (N, E)$)**:**
  $P^{n^r_s, n^r_d} \leftarrow$ `all_paths`(n^r_s, n^r_d);
  for p_1 **in** $P^{n^r_s, n^r_d}$ **do**
    for p_2 **in** $P^{n^r_s, n^r_d}$ **do**
      $\delta c \leftarrow \max(1, \mid D^r_{p_2} - D^r_{p_1} \mid)$;
      if `!disj`(p_1, p_2) **or** $bw^r > bw_c \lceil |C|/\delta c \rceil$ **then**
        continue;
      for c_{strt} **in** $[0, C-1]$ **do**
        $sch_{p_1,d} \leftarrow$ `gen_sch`(c_{strt}, bw^r, p_1, δc, G);
        if $sch_{p_1,d} \neq \phi$ **then**
          $\omega_{p_1} \leftarrow$ `sch_pth`($sch_{p_1,d}$, p_1, *dir =-1*);
          if $\omega_{p_1} \neq \phi$ **then**
            $\omega_{p_2} \leftarrow$ `sch_pth`($\omega_{p_1}[(p_1[0], p_1[1])]$, p_2, *dir =+1*);
            if $\omega_{p_2} \neq \phi$ **then**
              return $\omega_{p_1} \cup \omega_{p_2}$
  return ϕ;
Procedure `rt_sch`(R, $\mathbb{T}$, $G = (N, E)$)**:**
  $objval$, $sch \leftarrow 0$, ϕ;
  for r **in** R **do**
    $\omega^r =$ `greedy_rt_sch`($\mathbb{T}^r$, G);
    if $\omega^r \neq \phi$ **then**
      $objval$, $sch \leftarrow objval + bw^r$, $sch \cup \omega^r$;
      `upd_bw`(ω^r, G);
  return sch,$objval$;

ing procedure `pth_chng`. After completing a move, the procedure `nbr_soln` returns the selected move along with the new solution. For both kinds, a *move* is skipped and another one is considered if it was performed recently or it is tabued in the current iteration.

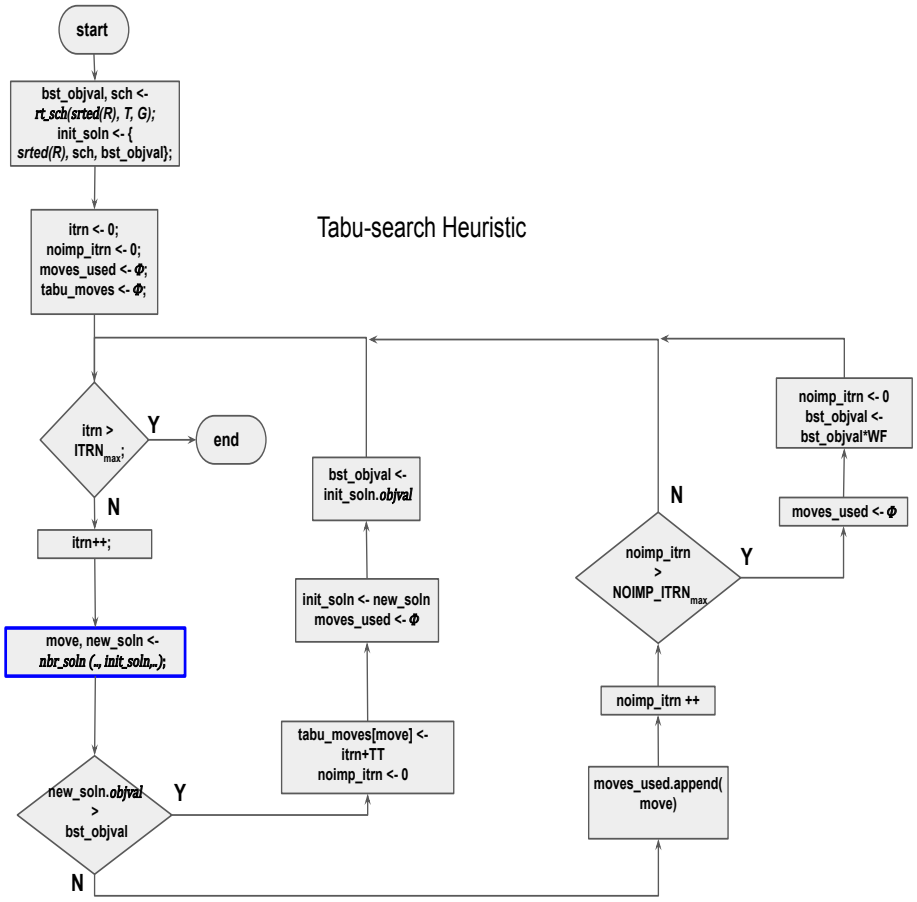


Figure 5.4: Flowchart depicting TS heuristic to solve the 1+1 *RTSCH problem.*

5.6 Evaluation and Results

In this section, we evaluate the performance of the ILP approach and the two proposed heuristics and then compare them. First, we describe the

Algorithm 5.2: Procedure to generate a neighbouring solution in the Tabu-search heuristic.

```
1  Procedure nbr_soln(itrn, soln, moves_used, moves_tabu, G):
2      if rand() <= pr_req then
3          i, j ← randsel(soln.R), randsel(soln.R) ;
4          while i ≠ j and ((i, j) ∉ moves_used ∪ moves_tabu or
             itrn > moves_tabu[(i, j)]) do
5              i, j ← randsel(soln.R), randsel(soln.R) ;
6          swap_req(soln.R, i, j);
7          return (i, j), {soln.R, rt_sch(soln.R, G)};
8      else
9          i ← randsel(soln.R) ;
10         while i ∉ moves_used ∪ moves_tabu or itrn > moves_tabu[(i, j)]
             do
11             i ← randsel(soln.R) ;
12         sch, objval ← pth_chng(soln.R[i], G);
13         return (i), {soln.R, sch, objval};
```

simulation setup and list the associated parameters. Then, the results comparing the different approaches in terms of various metrics are presented.

5.6.1 Setup and Parameters

All the simulation experiments are performed on an Intel Xeon server machine with quad-core CPU @ 2.40GHz with 16GB of RAM memory running Ubuntu-16.04 OS. The ILP model for the *1+1* RTSCH problem has been built using the Python API of IBM's ILOG CPLEx called DOcplex (Decision Optimization CPLEX Modeling). Both the heuristics are written in Python programming language and the Networkx package is used to generate paths between two nodes in the given graph.
The cycle time T_c by default is assumed to be 20μs and the number of cycles per hypercycle is 50.
For the evaluation of the ILP model and two heuristics, we have considered six topologies. Fig. 5.5 shows the topologies hexagon: `hex1`-`hex6`, that we have chosen to represent the DetNet infrastructure. Here, `hex1` is a ring topology where two disjoint paths exist between any pair of nodes. `hex2` and `hex3` are ring topologies with one and two diagonals, respectively. `hex4` also contains two diagonals; though, the average difference between disjoint points is small as compared to `hex3`. `hex5` contains three diagonals and `hex6` is a full mesh where each node is directly connected to the rest of the nodes. We have chosen the above topologies to highlight the impact of

connectivity on the average accepted traffic.
Each link in these topologies is a 10G link, thus the bandwidth per cycle $bw_{n_i,n_j,c}$ is 16x1500B packets per 20μs; the node processing delay and link delay (propagation and transmission) d_{n_i,n_j} are 40μs and 80μs, respectively.

For each request $r \in R$, the endpoints: source and destination are ran-

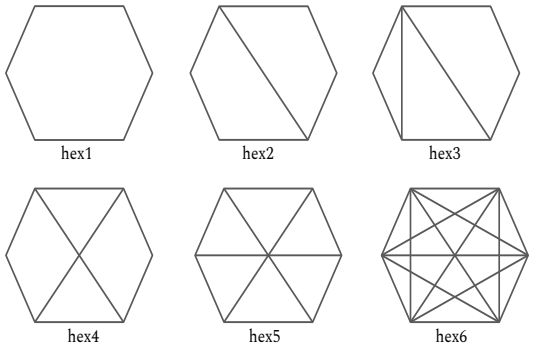


Figure 5.5: DetNet topologies used for the evaluation.

domly chosen from the set of topology vertices and its required bandwidth bw^r is randomly chosen with uniform probability in the range $[50, 120]$ packets (1500B).
Table 5.2 list the parameters involved in the simulation experiment and their corresponding values/ranges.

Table 5.2: Default values/range of various parameters involved in the evaluation.

Parameter	Value or range	Units
Topology	`hex1`, `hex2`, `hex3`, `hex4`, `hex5`, `hex6`	-
Request batch size ($\mid R \mid$)	{30, 50, 70}	numbers
Request bandwidth size (bw^r)	$[50, 120]$	packets
Total cycles per hypercycle ($\mid C \mid$)	$\{9, 18, 27, 54, 108, 216\}$	numbers
Cycle time (T_c)	$\{20, 40, 80, 160, 240, 480\}$	μs
Cycle max. bandwidth ($bw^{max}_{n_i,n_j}$)	$\{8, 16, 32, 64, 128, 256\}$	packets
Node processing delay (d_{n_i,n_j})	40	μs
Links delay (d_n)	80	μs

5.6.2 Performance and Calculation time Trade-off

The *1+1* RTSCH problem is solved using the CPLEX solver and the two heuristics; greedy (`GRD`) and tabu-search (`TS`). Fig. 5.6 shows the percentage of total for $|R| = 30$ with `hex1` topology. The low traffic acceptance for the three approaches results from the rejection of many requests due to non-adherence of the *spacing constraint* in `hex1`. On one hand, `GRD`'s performance is the lowest as it just returns the first feasible solution with greedy bandwidth allocation for each request. On the other hand, the ILP approach has the highest performance because it returns the optimal solution by searching through the entire solution space. Although, the performance of the `TS` approach is inferior to the `ILP` approach it still outperforms the `GRD` approach, especially for large problem sizes as will be highlighted in the next section.

The `ILP` approach, though is the most optimum approach, may not be suitable for large size instances of the *1+1* RTSCH problem. Even for a reasonable problem size, RTSCH'ing $|R| = 30$ on `hex1` topology, the CPLEX calculation exceeds 30 minutes. The calculation time increases rapidly when the topology is changed from `hex1` to `hex2`; in fact, it runs into several hours for $|R| \geq 30$ with the `hex2` topology. This is because as the connectivity of the topology increases, the solution space explodes due to the increase in the number of paths between two nodes thus resulting in very high calculation time. Fig. 5.7 depicts the evolution of best bound (Bst-Bnd), objective value (Obj) and integer (Bst-Int) solution with time for $|R| = 30$ and the `hex2` topology. Here Bst-bnd and Obj are the CPLEX's best objective function value achievable and current node's objective values, whereas Bst-int is the objective value of the best integer solution. It can be observed that most improvements in the integer solution's objective value are found in the early few minutes and there is a marginal improvement afterward; nonetheless, the CPLEX calculation time spans several hours. As an alternative to the ILP approach, the two heuristics scale well with the problem size that returns a solution within a few seconds.

Table 5.3 lists the CPLEX calculation times for the *1+1* RTSCH problem for $|R| = \{30, 50, 70\}$ with the `hex1` topology. The calculation time for `TS` is higher as compared to `GRD`; though, the maximum calculation time (for $|R| = 70$) is still within a few seconds. We also observed that the calculation time for `GRD` and `TS` does not vary with the topology. The ILP approach, on the other hand, is infeasible for all the problem sizes, except for $|R| = 30$ and the `hex1` topology, thus the calculation times are not indicated here. Consequently, it is unfit to solve the reasonable size instances of the *1+1* RTSCH problem and the heuristics should be employed for such instances.

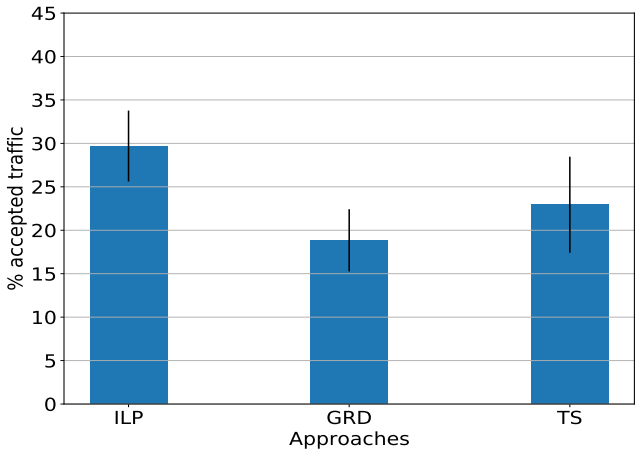


Figure 5.6: Comparison of the ILP and heuristics in terms of the percentage of the total accepted traffic.

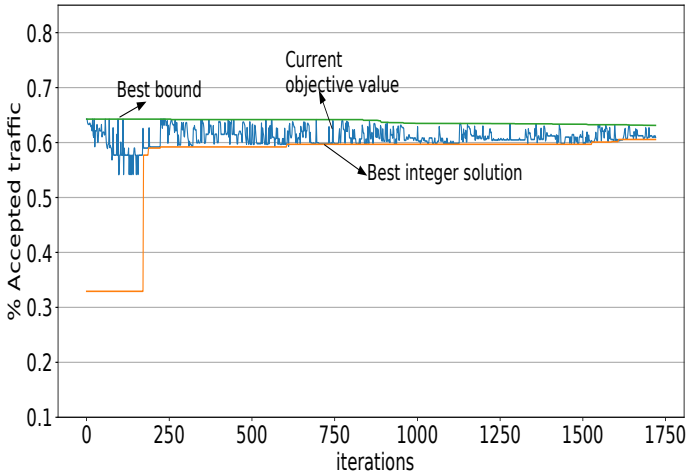


Figure 5.7: Evolution of the various ILP parameters with time.

Table 5.3: Approximate calculation time for GRD and TS with the hex1 topology.

$\|R\|$	GRD time(s)	TS time(s)
30	0.004	2.4
50	0.008	4.0
70	0.01	7.0

5.6.3 Heuristic Performance

Before presenting the performance results for the heuristics, we present the results showing the percentage of the total acceptable traffic for the topologies hex1-hex6. Fig. 5.8 shows the percentage of total accepted traffic satisfying only the spacing constraint assuming each cycle has the maximum bandwidth $bw^{max}_{n_i,n_j}$. These results indicate that with the increase in the connectivity of a topology, from hex1 to hex3, the average number of paths between any two nodes increases. As the number of paths increases, the number of requests that adhere to the spacing constraint increases resulting in an increase in the total accepted traffic.

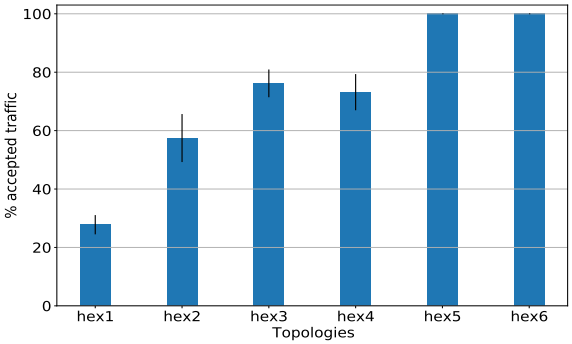


Figure 5.8: The percentage of the total accepted traffic for different hex topologies, considering only the spacing constraint.

Next, we report the percentage of the total accepted traffic taking into account the actual cycle bandwidth $bw_{n_i,n_j,c}$ instead of maximum cycle bandwidth $bw^{max}_{n_i,n_j}$. The total accepted traffic is the sum of the bandwidth requirements of all requests that can be routed and scheduled. Fig. 5.9 shows the percentage of the total accepted traffic for the sets of requests with different sizes. The total accepted traffic for the two heuristics decreases with increasing $\mid R \mid$ because increasingly cycles are consumed for the accepted traffic and lesser bandwidth per cycle is left for the other requests. The total accepted traffic also increases as the connectivity of the

graph increases which in turn increases the average number of possible paths for each request. It can also be observed that `TS` outperforms `GRD` in all the cases. For instance, in `hex4` and $|R| = 50$, the % accepted traffic for `TS` is better as compared to `GRD` by 8%.

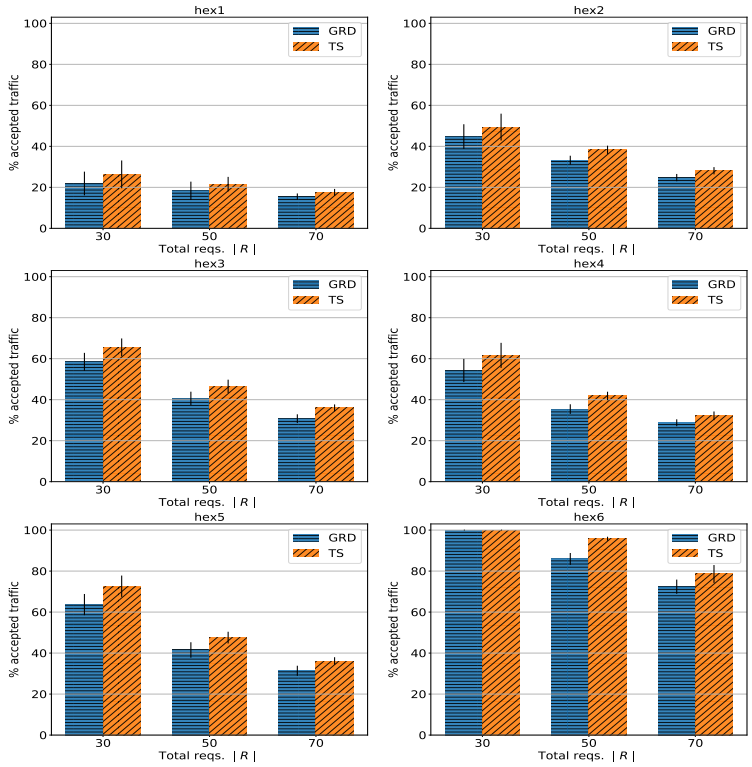


Figure 5.9: The percentage of the total accepted traffic with $|R|$ for different hex topologies.

5.6.4 Impact of T_c

Next, the influence of the cycle time on the performance of the two heuristics is reported. The COST239 network is selected for this evaluation. We selected this network as it has sufficient connectivity as shown in Fig. 5.10. Each node is connected to at least four neighbours, thus providing sufficient redundancy for $1 + 1$ protection.
Fig. 5.11 shows the percentage of the total accepted traffic for $\mid R \mid = 50$ on the COST239 network for cycle times between 20-480μs. It can be observed that as the cycle time is increased, the total accepted traffic increases for

all topologies. As the cycle time increases, the total bandwidth of each cycle increases (though the total number of cycles in a hypercycle decreases), more requests can be mapped to the same cycle thus improving the total accepted traffic.

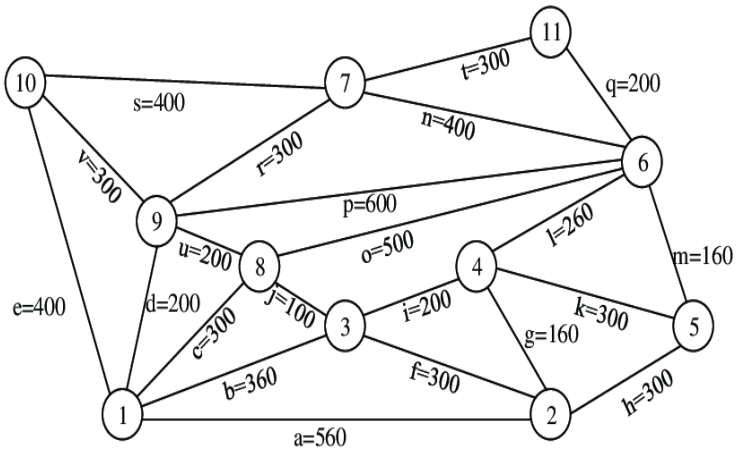


Figure 5.10: Topology for the pan-European COST239 network. The label corresponding to each link is its length (in kms).

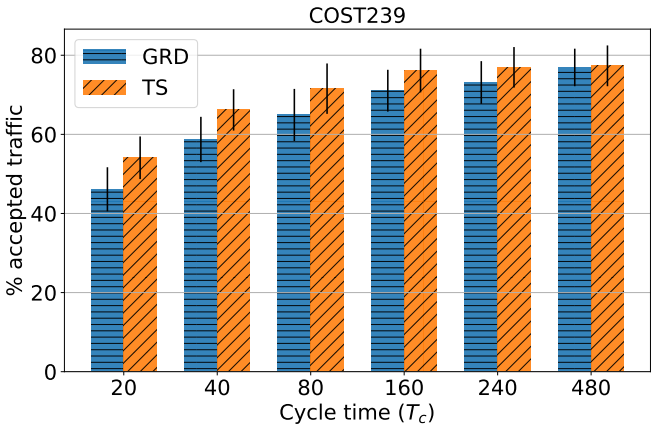


Figure 5.11: The influence of T_c on the percentage of the total accepted traffic for $|R| = 50$ for the COST239 network.

The cycle time also impacts the end-to-end delay between the source and the destination. As described in [3], the end-to-end delay d_{e2e} on path

p with N nodes (including source and destination) and cycle time T_c are related as : $d_{e2e} = (N-1)d^{tx}_{n_i,n_j} + (N-2)(d^{proc}_{n_j} + 2T_c)$. The average end-to-end delay for $\mid R \mid = 50$ between the endpoints of the accepted requests on the six topologies is depicted in Fig. 5.12. As expected, the end-to-end delay increases with increasing cycle time as packets have to be queued for a longer time.

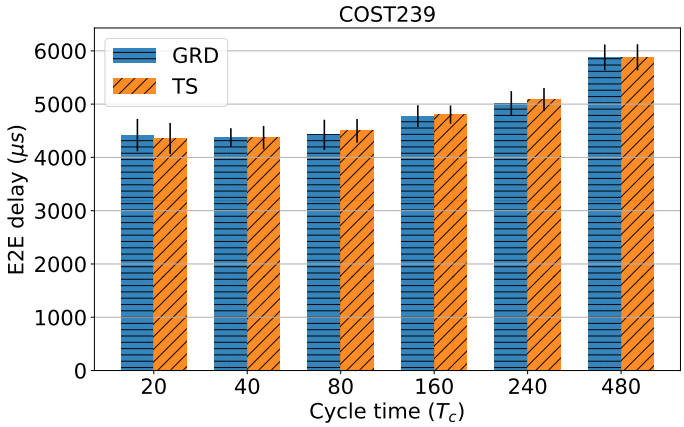


Figure 5.12: The influence of T_c on the average end-to-end delay for $|R| = 50$ for the COST239 network.

5.7 Conclusion

Packet losses due to network congestion can be addressed using DetNet mechanisms such as CSQF. To guarantee protection against failures in the network, a scheme like *1+1* is required. The problem of *1+1* RTSCH for DetNets was investigated in this paper. To allow reliable recovery, the packet schedules on the two selected paths should take into account the end-to-end delays. To this end, we formulated the *1+1* RTSCH problem as an ILP. We proposed two additional approaches: greedy and Tabu-search heuristics, that are scalable for relatively large problem sizes. Performance evaluation of these approaches highlights the influence of the topology and request set size on the average accepted traffic. Although the ILP approach is exact, it is infeasible beyond small-sized problem instances. The `GRD` and `TS` heuristic trade-off on the solution quality ($< 10\%$) for its speed such that reasonable size problem instances can be solved in a few seconds which the ILP approach would require several hours to solve.
Moreover, the trade-off of choosing T_c on total accepted traffic and the end-

to-end delay was also discussed.
Professional media flows can be reliably transported using CSQF in a DetNet. The future work will include the modeling of RTSCH for media services that are represented by Virtual Media Function Forwarding Graphs (VMFFG) in contrast to simple services consisting of just source-destination pairs.

5.8 Acknowledgements

This research was funded by the Flemish FWO SBO S003921N VERIEND.com (Verifiable and elastic end-to-end communication infrastructures for private professional environments) project, by the FWO project under grant agreement #G055619N and the Flemish Government under the "Onderzoeksprogramma Artificiele Intelligentie (AI) Vlaanderen".

References

[1] E. Grossman, C. Gunther, P. Thubert, P. Wetterwald, J. Raymond, J. Korhonen, Y. Kaneko, S. Das, Y. Zha, B. Varga, et al. *Deterministic networking use cases*. IETF draft, 2018.

[2] N. Finn, P. Thubert, B. Varga, and J. Farkas. *Deterministic networking architecture*. draft-ietf-detnet-architecture-03 (work in progress), 2017.

[3] M. Chen, X. Geng, and Z. Li. *Segment Routing (SR) Based Bounded Latency*. Internet Engineering Task Force, Internet-Draft draft-chendetnet-sr-based-bounded-latency-00, 2018.

[4] N. Sprecher and A. Farrel. *Rfc 6372-mpls transport profile (mpls-tp) survivability framework*. IETF MPLS Working Group (MWP), Internet-Draft, 2011.

[5] J. Krolikowski, S. Martin, P. Medagliani, J. Leguay, S. Chen, X. Chang, and X. Geng. *Joint routing and scheduling for large-scale deterministic IP networks*. Computer Communications, 165:33 – 42, 2021. Available from: http://www.sciencedirect.com/science/article/pii/S0140366420319642, doi:https://doi.org/10.1016/j.comcom.2020.10.016.

[6] *IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications*. IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011), pages 1–421, 2020. doi:10.1109/IEEESTD.2020.9121845.

[7] S. Lindner, D. Merling, M. Häberle, and M. Menth. *P4-Protect: 1+ 1 Path Protection for P4*. arXiv preprint arXiv:2001.11370, 2020.

[8] *IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability*. 2017. doi:10.1109/ieeestd.2017.8091139.

[9] A. Nasrallah, A. S. Thyagaturu, Z. Alharbi, C. Wang, X. Shao, M. Reisslein, and H. ElBakoury. *Ultra-low latency (ULL) networks: The IEEE TSN and IETF DetNet standards and related 5G ULL research*. IEEE Communications Surveys & Tutorials, 21(1):88–145, 2018.

[10] S. S. Craciunas and R. S. Oliver. *SMT-based task-and network-level static schedule generation for time-triggered networked systems*. In Proceedings of the 22nd international conference on real-time networks and systems, pages 45–54, 2014.

[11] N. G. Nayak, F. Dürr, and K. Rothermel. *Time-sensitive software-defined network (TSSDN) for real-time applications.* In Proceedings of the 24th International Conference on Real-Time Networks and Systems, pages 193–202, 2016.

[12] F. Dürr and N. G. Nayak. *No-Wait Packet Scheduling for IEEE Time-Sensitive Networks (TSN).* In Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS '16, page 203–212, New York, NY, USA, 2016. Association for Computing Machinery. Available from: https://doi.org/10.1145/2997465.2997494, doi:10.1145/2997465.2997494.

[13] W. Steiner. *An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-Hop Networks.* In Proceedings of the 2010 31st IEEE Real-Time Systems Symposium, RTSS '10, page 375–384, USA, 2010. IEEE Computer Society. Available from: https://doi.org/10.1109/RTSS.2010.25, doi:10.1109/RTSS.2010.25.

[14] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner. *Scheduling Real-Time Communication in IEEE 802.1Qbv Time Sensitive Networks.* In Proceedings of the 24th International Conference on Real-Time Networks and Systems, RTNS '16, page 183–192, New York, NY, USA, 2016. Association for Computing Machinery. Available from: https://doi.org/10.1145/2997465.2997470, doi:10.1145/2997465.2997470.

[15] S. Chen, J. Leguay, S. Martin, and P. Medagliani. *Load Balancing for Deterministic Networks.* In 2020 IFIP Networking Conference (Networking), pages 785–790. IEEE, 2020.

[16] *Dynamic Multipath Optimization.* Technical report, VMware, 2020. Available from: http://wan.velocloud.com/rs/098-RBR-178/images/sdwan-678-dynamic-multipath-optimization-wp-1119.pdf.

[17] Y. E. Oktian, S. Lee, H. Lee, and J. Lam. *Distributed SDN controller system: A survey on design choice.* computer networks, 121:100–111, 2017.

[18] F. P.-C. Lin and Z. Tsai. *Hierarchical edge-cloud SDN controller system with optimal adaptive resource allocation for load-balancing.* IEEE Systems Journal, 14(1):265–276, 2019.

[19] E. L. Qiang, X. Geng, B. Liu, E. T. Eckert, and L. Geng. *Large-Scale Deterministic IP Network.* Internet Engineering Task Force, Internet-Draft draft-qiang-detnet-large-scale-detnet-05, 2020.

[20] A. A. Hagberg, D. A. Schult, and P. J. Swart. *Exploring Network Structure, Dynamics, and Function using NetworkX*. In G. Varoquaux, T. Vaught, and J. Millman, editors, Proceedings of the 7th Python in Science Conference, pages 11 – 15, Pasadena, CA USA, 2008.

6

End-to-end Scheduling for Wired-wireless Mixed Networks

The previous two chapters presented scheduling mechanisms that provide guarantees on end-to-end delay and jitter in wired networks. While such mechanisms have been around in the wired domain for many years, the feasibility of such mechanisms has been demonstrated in the wireless domain recently. Supporting time-sensitive applications in a mixed network that consists of both wired and wireless links necessitates packet scheduling on all the links on the routed path.
This chapter deals with the problem of end-to-end scheduling for time-sensitive flows on wired-wireless mixed networks. The problem is formulated as an ILP and a greedy heuristic is presented to solve the large-scale problem instances.

⋆⋆⋆

G.P. Sharma, W. Tavernier, D. Colle, M. Pickavet, J. Haxhibeqiri, J. Hoebeke, and I. Moerman

Submitted to Computer Communication, 2022.

Abstract Proprietary communication technologies for time-critical communication in industrial environments are being replaced with Time-Sensitive

Networking (TSN) -enabled Ethernet. Furthermore, attempts have been made to bring TSN features in wireless networks so that wireless flexibility can be exploited and at the same time, the end-to-end timings for Time-Triggered (TT) flows can be guaranteed. Given a wired-wireless mixed network, the scheduling problem should be solved for a given set of TT flow requests. In this paper, we formulate the scheduling problem for wired-wireless mixed networks as an Integer Linear Programming (ILP) model. Next, a greedy-based heuristic is proposed to solve the realistic-sized instances of the problem. Evaluation results show that the greedy heuristic is suitable for realistic problem sizes where the ILP approach is found to be infeasible. Furthermore, the impact of wireless requests on the performance of the greedy heuristic is reported.

6.1 Introduction

Owing to its cost-effectiveness and maturity, Ethernet has been a defacto link-layer technology in standard communication networks. However, industrial environments are traditionally dominated by proprietary communication technologies such as Time-Triggered Ethernet (TTEthernet) and EtherCAT that are inflexible and costly but guarantee timely delivery of Time Sensitive (TS) traffic. In order to bring time sensitivity in Ethernet, the IEEE 802.1 Time-Sensitive Networking (TSN) Task Group (TG) released a suite of standards in 2012 [1]. This suite of standards specifies various mechanisms (e.g., time-synchronization, time-aware shaping) aimed at enabling the co-existence of Time-triggered (TT) traffic that requires end-to-end timing guarantees along with standard Best-Effort (BE) traffic flows that do not need any timing assurances.

Time-Aware Shaping (TAS) or IEEE 802.1Qbv is one of the TSN components that emulates Time Division Multiple Access (TDMA) on each egress port [2]. The transmission time on links is divided into time slices that are assigned to each TT flow, thus ensuring no interference from other TT or BE flows.

Wireless networks offer several advantages in comparison to wired networks. A wireless network interconnecting several devices can be deployed quickly without requiring complex cable installation thus resulting in cost savings. It also enables connectivity of endpoints that are not accessible for cables, especially in dynamic industry environments (e.g., for mobile robots or moving machine parts). Other advantages include easier reconfigurability of factory settings and scalability. Owing to the evolution of high-speed wireless networking and the above advantages, networks consisting of both wired and wireless segments are prevalent in all kinds of industry

environments [3]. End-to-end guarantees in such *mixed wired-wireless networks* can be provided by implementing TSN mechanisms in wireless, in addition to wired links. The implementation of such a TSN mechanism in wireless has been gaining some attention, recently. In [4], a Proof-of-Concept (PoC) for wireless TSN has been demonstrated, which is based on WiFi-based Software Defined Radio (SDR) called Openwifi [5]. The PoC implements time-synchronization and time-aware shaping in the wireless domain. Considering the possibility of wireless TSN, the problem of scheduling TT flows on a given wired-wireless mixed network still remains a challenge. The end-to-end scheduling problem has been addressed for wired TSN widely [6], [7], [8]. However, modeling of the end-to-end scheduling problem for a wired-wireless mixed network is not straightforward because of these reasons: (i) difference in transmission speeds of wired and wireless links, (ii) half-duplex communication in wireless links as opposed to that of wired links and (iii) sharing of wireless links among Wireless Access point (WAP) and multiple endpoints. To this end, this paper addresses the scheduling problem for wired-wireless mixed networks. More concretely, the contributions of this paper are as follows:

1. We propose a transformation of the wireless link as a graph consisting of only simplex links. This transformation simplifies the modeling of wireless links in the mixed network and simplifies the scheduling algorithm.

2. An ILP model representing the scheduling problem for a wired-wireless mixed network is presented. A greedy-based heuristic method is proposed, as the ILP approach is not feasible beyond small-size problem instances.

3. Finally, we extensively evaluate the proposed ILP and greedy heuristic on two different topologies via simulation experiments.

The remainder of the article is organized as follows: Section 6.2 provides a survey of the key related works. The system model along with the problem statement is presented in section 6.3. The ILP formulation for the mixed network wired-wireless scheduling problem is discussed in 6.4. Next, a greedy-based heuristic algorithm is described in section 6.5. The ILP approach and heuristic are evaluated for scalability and performance in section 6.6. The paper is concluded in section 6.7.

6.2 Related Works

Efforts to bring time sensitivity into Ethernet have been going on for several years [9], [10]. The TSN-TG, a task group of IEEE 802.1 Working Group (WG), has published a suite of standards targeting extensions to IEEE 802 networks that enable end-to-end deterministic connectivity, i.e, bounded delays and jitter along with high availability [1]. The most relevant standards in the suite are listed in Table 6.1.

Table 6.1: Overview of various TSN standards.

Standard	Description
IEEE 802.1AS-Rev	Timing and synchronization for time-sensitive Applications [11].
IEEE 802.1Qav	Prioritization of time-sensitive streams over best-effort using Credit Based Shaper (CBS) [12].
IEEE 802.1Qbv	Scheduling of time-sensitive traffic using time-aware shaping [2].
IEEE 802.1Qca	Path control and reservation [13].
IEEE 802.1Qci	Per-stream Filtering and Policing [14].
IEEE Std 802.1CB-2017	Frame Replication and Elimination for Reliability [15].

The IEEE 802.1Qbv aka Time-aware Shaping (TAS) specifies the mechanisms to implement TDMA in IEEE 802 networks [2]. The use of the TAS entails the computation of routes and schedules based on the given network and TT flow specifications. Earlier works on the TSN scheduling problem were mostly concerned with computing schedules for proprietary networks (e.g., TTEthernet, PROFINET), whereas recent works have focused on computing schedules for TAS-enabled bridged networks. A Satisfiability Modulo Theory (SMT) model was proposed by Tămaş–Selicean et al. to solve the scheduling problem in TTEthernet [9]. Schweissguth et al. in [8] have proposed an ILP to jointly solve the Routing and Scheduling problem for TTEthernet networks. Hanzaelk et al. have formulated the scheduling problem for Profinet in terms of the Resource-Constrained Project Scheduling with Temporal Constraints and present an algorithm to solve it [16].
The No-wait Packet Scheduling Problem (NW-PSP) for the TAS-enabled bridged network has been mapped to the No-wait Job-shop Scheduling Problem (NW-JSP) in [6] and is formulated as an ILP. Hellmanns et al. have proposed and evaluated various optimizations for ILP-based TSN scheduling in [17]. However, despite these optimizations, the ILP-based approach to solve the TSN scheduling problem does not scale well with respect to

the network topology size and the total number of TT flows. This stems from the fact that the TSN scheduling problem is an NP-hard problem and thus the exact approaches such as ILP and SMT become infeasible for the problem instances with large sizes.
To this end, various heuristics and meta-heuristics are proposed to return a suboptimal solution for the problem quite faster than the exact approaches. By compromising on the solution quality, the problem can still be solved for the problem instances with large sizes. Durr et al. also presented a tabu-search-based heuristic, as a faster alternative to the ILP approach, to solve the NW-PSP problem [6]. Pahlevan et al. solved the joint routing and scheduling algorithm using a Genetic Algorithm (GA) and demonstrated that the proposed algorithm results in significant improvement in schedulability, as well as flow-span over the List Scheduling (LS) approach [18]. An improved Ant-Colony Optimization (ACO) was proposed by Yang et al. to ensure deterministic end-to-end delay and jitter for TT traffic [19].
The aforementioned works are mainly focused on scheduling TT flows in a TSN-capable wired network despite the fact that various wireless technologies such as 5G and WiFi have been extended to incorporate TSN features. Haxhibeqiri et al. and Cavalcanti et al. have discussed, in detail, the prerequisites, i.e., time-synchronization and traffic shaping, to extend TSN capabilities in wireless networks [4], [20]. They also demonstrated a WiFi-based TSN solution that can be synchronized within a few μs along with an end-to-end latency of 3ms. The authors in [21] have discussed how the key features in WiFi7 (IEEE802.11be) can be leveraged to implement TSN functionalities to support low-latency communication. TSN-capable wired-wireless networks are feasible, although the scheduling problem for such mixed networks has not received serious attention. Ginthör et al. solved the scheduling problem for a mixed network consisting of TSN and 5G bridges using constraint programming while optimizing resource utilization [22]. As per our knowledge, the scheduling problem for wired-WiFi mixed networks has not been modeled or addressed yet. The SMT/ILP formulations and heuristics existing in the literature to represent the scheduling problem in wired TSN are not readily applicable to wired-WiFi mixed networks for the reasons explained in section 6.3. Therefore, an ILP formulation and a greedy-based heuristic are presented in this paper to address the problem.

6.3 Problem Statement

To achieve end-to-end timing guarantees for TT flows with TAS in a wired-wireless mixed network, a gating mechanism for all the classes of traffic is required in every node, as shown in Fig. 6.1. Each egress port of the TAS-

capable switch has multiple queues, each corresponding to a traffic class. A frame is assigned to a queue based on the Priority Code Point (PCP) in the VLAN tag of the frame and a configurable mapping of the PCP values to the hardware queues. The logic for queue selection for transmission at an egress port is based on the Gate Control List (GCL) for that egress port. A GCL contains a sequence of entries, each containing the duration of the entry and a bit-mask indicating the queues that are allowed transmission until the start of the next entry. Strict priority queuing is used for transmission selection, in case two queues have packets to transmit at the same time. The total duration of GCL entries is called the cycle time, denoted by T^{cyc}, i.e., the GCL sequence repeats itself after T^{cyc} interval.
We assume one queue is reserved at each egress port for TT traffic and its frames are assigned to this queue after selecting the egress port. We also apply the no-wait constraint to TT flows. In other words, the TT traffic's frame traverses through the routed path from the source to the destination without queuing at any intermediate node. With the no-wait constraint, the network offers the minimum possible delay on the routed path as the queuing delay in each node is zero.
It can be safely assumed that all nodes (switches, WAPs and endpoints), of the network, are synchronized (e.g., using Precision Time Protocol) within μs accuracy and are capable of adhering to the schedules they are configured with. This entails that the wireless nodes (WAP and endpoints) sharing the wireless medium can only transmit during their allocated time slots thus avoiding packet collision even within the shared medium. Given such guarantees, it is thus possible to determine the packet arrival time at any point on the routed path in a wired-wireless mixed network thus guaranteeing end-to-end delay. Next, we describe the system model and introduce the scheduling problem for wired-wireless mixed networks.

6.3.1 Model Description

The set of requests corresponding to the TT flows that need to be scheduled is denoted by R. For each request $r \in R$, a six-valued tuple $(n_r^{src}, n_r^{dst}, F_r, fsize_r, T_r, D_r^{e2e})$ is used to represent the attributes of each flow request. Here, n_r^{src} and n_r^{dst} denote the requested flow's source and destination nodes, respectively. The flow request r produces a total of F_r frames, each equal to $fsize_r$ bytes every T_r seconds. A request source, e.g., camera, might produce multiple $fsize_r$ sized frames in a cycle duration thus $F_r \geq 1$. T_r denotes the time period corresponding only to request r. The period of all requests in R need not be equal. We denote the common period (hyper-period) of all requests as T^{cyc}, i.e., the minimum time that is a period of all the requests; $T^{cyc} = LCM(\{T_r : \forall r \in R\})$. This is equal to the cycle time

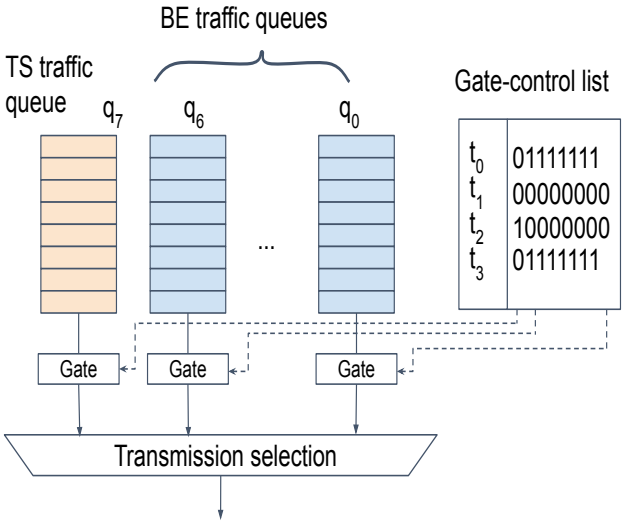


Figure 6.1: Typical architecture of an egress port supporting TAS (IEEE 802.1Qbv).

of the GCL sequence in each node of the network. At the start of every request cycle (T_r), F_r frames are produced by n_r^{src} that need to be scheduled in the next T_r seconds. Therefore, the number of frames that needs to be scheduled in cycle time T^{cyc} is $F_r^{cyc} = (T^{cyc}/T_r)F_r$. The maximum allowable end-to-end delay for request $r \in R$ is denoted by D_r^{e2e}.
The topology of the wired-wireless mixed network is represented by a directed graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$. The endpoints and switching nodes (TSN switches and WiFi APs) form the set of nodes $\mathcal{N}$; whereas, $\mathcal{E}$ denotes the set of physical (wired and wireless) links between the nodes. The forwarding nodes are capable of implementing TAS for TT flows, i.e., they comply with IEEE 802.1Qbv. In our model, we consider two types of delays– transmission delay and processing delay. The processing delay of node is denoted by d_n^{proc} and transmission delay for r's frame on link (n_i, n_j) is denoted by d_{r,n_i,n_j}^{trans} $(= fsize_r/B_{n_i,n_j})$, where B_{n_i,n_j} is the link speed in Bps. d_{r,n_i,n_j}^{trans} determines the duration for which the gate for the TT queue on node n_i (on output port corresponding to (n_i, n_j)) should be opened to let the frame to be transmitted. We have ignored the contribution of propagation delay ($< 1\%$ of T_r) as it is not significant in a LAN environment (e.g., factory network). p denotes a path on the network consisting of ordered list of nodes; whereas ϵ_p is the ordered list of edges in path p. D_{p,n_i} is the delay

along path p starting from the source node ($p[0]$) until the last node (n_i); this includes the transmission delay by n_i. D_p is the delay along path p until the last node ($p[-1]$) of p.
$\forall r \in R$, a route must be determined from n_r^{src} to n_r^{dst}. We assume that a maximum of K shortest paths between n_r^{src} and n_r^{dst} are predetermined using off-the-shelf routing algorithms, and passed as an input to the model to compute transmission schedules on one of these routes. This pre-processing vastly reduces the number of variables in the ILP model and thus reduces the solve time significantly [17].

6.3.2 Wireless link Modeling

Despite wireless nodes being capable of TAS, the computation of end-to-end scheduling of TT flows in wired-wireless mixed networks is generally more complex. Firstly, the transmission speed of the wireless link is sensitive to variable channel conditions. The interference can come from other wireless networks as WiFi operates in unlicensed frequency bands. The throughput value of the wireless link assumed in our model is 9 Mbps. Though WiFi supports higher transmission rates, choosing a relatively lower value results in higher reliability. Even though it can be ensured that the wireless nodes in the given network are allowed to transmit only during their allocated time slots, external interference is still possible. Thus it might be required to perform multiple re-transmissions to realize a successful end-to-end frame delivery of a TT flow. Here, we consider the case where the first transmission attempt is successful. However, the model can be further extended to allocate time slots for multiple re-transmissions to ensure a given reliability threshold is achieved. As explained next, an additional challenge is to model the given wired-wireless mixed topology such that a given scheduling algorithm can take it as an input.
The half-duplex wireless links are shared by multiple endpoints and the WAP whereas a wired link is always point-to-point and full-duplex. Therefore, computing the transmission schedule on the wireless link becomes complicated. To simplify the scheduling problem, the original wireless link (Fig. 6.2 (a)) can be modeled as a graph (Fig. 6.2 (b)) consisting of the WAP, additional dummy nodes and dummy links. n^{ap} is connected to two dummy nodes– n^{in-ap} and n^{out-ap} via simplex links and the two dummy nodes are connected to each via another simplex link (n^{in-ap}, n^{out-ap}). The dummy nodes n^{in-ap} and n^{out-ap} have 0 processing delay, whereas the processing delay of n^{ap} is the same as that of the original AP. The links (n_i, n^{in-ap}), (n^{out-ap}, n_i), (n^{ap}, n^{in-ap}) and (n^{out-ap}, n^{ap}) all have 0 transmission delay (or ∞ bandwidth). The link (n^{in-ap}, n^{out-ap}) represents the medium

shared between the WAP and its connected endpoints and has a bandwidth equal to the (minimum) wireless bandwidth. It can be seen that the half-duplex property of the wireless link is maintained in Fig. 6.2 (b), as any communication between the WAP and an endpoint or among endpoints themselves is traversed over (n^{in-ap}, n^{out-ap}) thus ensuring only one node is transmitting at a particular time.
Using the above transformation, each half-duplex wireless link can be broken into a subgraph consisting only of simplex links. Therefore, the given wired-wireless mixed network denoted by $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ can be transformed into a network, denoted by $G = (N, E)$, consisting only of simplex or full-duplex links. Tab. 6.2 lists the notations used for various parameters and sets along with their short description.

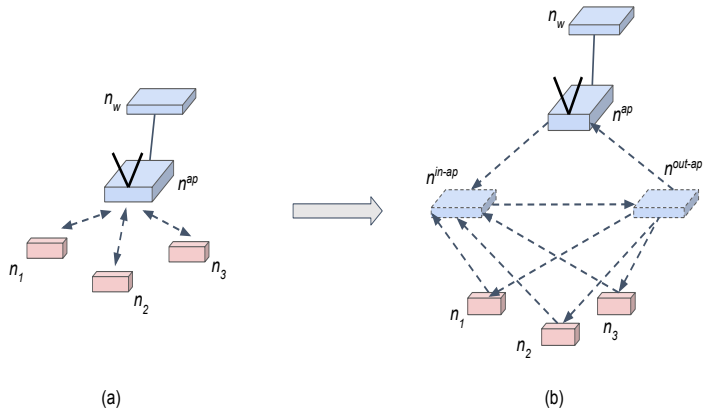


Figure 6.2: (a) Half-duplex wireless link modeled as (b) a graph consisting of dummy nodes and simplex links. The half-duplex property of the wireless link is retained by ensuring all communication traverses the dummy link (n^{in-ap}, n^{out-ap}).

6.3.3 Problem Definition

In IEEE 802.1Qcc, the centralized configuration model for TSN is described, wherein a central entity aka Centralized Network Controller (CNC) has a global view of the complete network and manages all nodes; this includes wireless APs as well as wireless endpoints. The architectural model for a centrally controlled wired-wireless mixed TSN is depicted in Fig. 6.4. The CNC is mainly responsible for (i) discovering the network topology that includes capturing the capabilities of various nodes and (ii) configuring nodes with appropriate GCLs. The GCLs are computed by the scheduler, a com-

ponent of the CNC, based on the specifications of TT traffic (R) and the given network topology G (with delay parameters). As topology discovery for wired-wireless mixed networks is not the subject of this article, we will focus only on the scheduling aspect. We assume the scheduler is provided with the global view of the network along with delay parameters and the request set.

The end-to-end scheduling problem can be defined as the problem of computing the end-to-end optimized schedules for all TT flow requests given the topology graph of a wired-wireless mixed network and requests' specifications. The optimization criterion considered in our model is the flow-span time. The flow-span time is defined as the maximum injection time among all TT flows with respect to the start of the request cycle (T_r). Fig. 6.3 shows the frame injection times (length of arrows) of a request r; where $F_r = 2$ and $T^{cyc} = 4T_r$ thus $F_r^{cyc} = 8$. Here, the request flow-span corresponds to the relative injection time of frame 7 as it has the longest relative injection time.

By choosing flow-span as our optimization, we make sure that the TT flow frames are not queued at the source node for too long, thereby reducing the end-to-end delay.

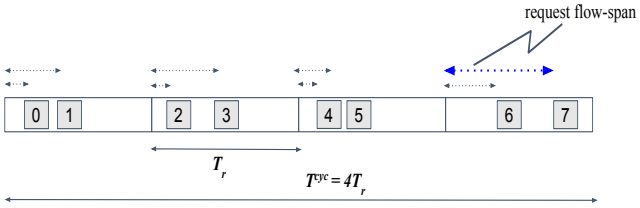


Figure 6.3: Illustration for the maximum flow-span of request r. The length of each dotted arrow indicates its injection time relative to the start of the request's cycle. The flow-span of this request corresponds to the relative injection time (in blue) of frame 7.

With the no-wait constraint, the scheduling problem boils down to the computation of the solution consisting of (i) the routed path and ii) the injection times for all TT flow requests. Using this solution and the delay along the routed path, the gate opening and closing times at each intermediate node are determined, along with the guard times to prevent the BE traffic from interfering with the scheduled TT traffic [6]. The gate opening and closing times are then used for the generation of the GCL for network

nodes. The CNC configures each node in the network with the GCL so that all TT flow requests are supported.

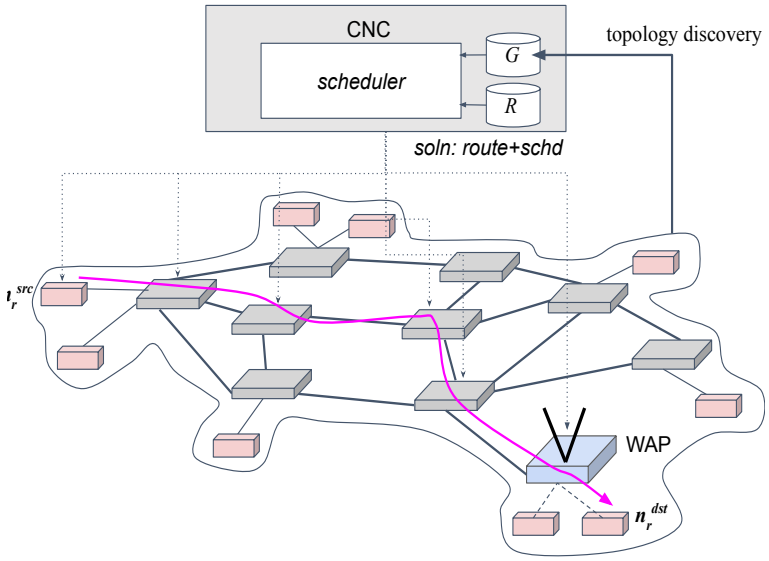


Figure 6.4: Overview of the end-to-end scheduling problem in a wired-wireless mixed network. The red, blue and grey boxes represent the endpoints, WAP and switching nodes in the network, respectively.

6.4 ILP formulation

In this section, we present the ILP formulation for the end-to-end scheduling problem for wired-wireless mixed networks. Here, we describe the ILP objective function along with various constraints that ensure end-to-end guarantees are given to TT flows. Tab. 6.2 lists the notations used for all variables involved in the ILP formulation along with their short description.

6.4.1 Constraints

Each request should be mapped to exactly one physical path in G. The constraint in (6.1) ensures that only one path is selected for routing request $r \in R$ out of K shortest paths between n_r^{src} and n_r^{dst}.

$$\sum_{p \in P_{r,K}} \gamma_{r,p} = 1, \quad \forall r \in R. \tag{6.1}$$

Table 6.2: Description of the notations used for the parameters and variables involved in the system model.

Notation	Description
R	The set of requests that need to be routed and scheduled on a given Time-Sensitive Network (TSN). Each request $r \in R$ is associated with a tuple $\mathcal{T} = (n_r^{src}, n_r^{dst}, F_r, T_r, D_r^{e2e})$, where n_r^{src}, n_r^{dst} and T_r are request's source node, destination node, total frames and time period, respectively. The set of all frames that needs to be transmitted by request r during request period T_r is denoted by $\mathcal{F}_r = \{f : 0 \leq f \leq F_r - 1\}$
T^{cyc}	The lowest time interval that is a period of all the requests, i.e., $T^{cyc} = LCM(\{T_r : \forall r \in R\})$. D_r^{e2e} is the maximum allowable end-to-end delay for all frames of request $r \in R$; this is relative to the start of the last common cycle (T^{cyc}).
F_r^{cyc}	The total number frames of request $r \in R$ during T^{cyc}, i.e., $F_r^{cyc} = (T^{cyc}/T^r)F$. The set of frames during T^{cyc} is denoted by $\mathcal{F}_{r_1}^{cyc} = \{f : 0 \leq f \leq F_r^{cyc}\}$
$G = (N, E)$	Directed graph representation of the network, where N is the set of nodes (switches, APs and endpoints) and E is the set of physical links between the nodes.
d_{n_i,n_j}^{prop}	The propagation delay on physical link $(n_i, n_j) \in E$.
d_{n_i,n_j}^{trans}	The transmission delay on physical link $(n_i, n_j) \in E$.
d_n^{proc}	The frame processing time of node $n \in N$.
$P_{r,K}$	K shortest paths between nodes n_r^{src} and n_r^{dst} for request $r \in R$; $p \in P_{r,K}$ is an ordered list of nodes from n_r^{src} to n_r^{dst}.
ϵ_p	Ordered list of edges in path p.
D_{p,n_i}	The delay along path p starting from $p[0]$ up to node $n_i \in p$.
D_p	The delay along path p starting from $p[0]$ up to $p[-1]$.
$\gamma_{r,p}$	The decision variable indicates if $r \in R$ is mapped to a physical path p.
$\rho_{r_1,p_1,f_1,r_2,p_2,f_2}^{n_i,n_j}$	The decision variable indicates that frame $f_1 \in \mathcal{F}_{r_1}^{cyc}$ of request $r_1 \in R$'s frame is scheduled before frame $f_2 \in \mathcal{F}_{r_2}^{cyc}$ of request $r_2 \in R(\neq r_1)$'s on the common link (n_i, n_j) of the routed paths p_1 and p_2 of r_1 and r_2, respectively.
$t_{r,f}^{in}$	The continuous decision variable is the injection time of frame $f \in \mathcal{F}_r^{cyc}$ of request $r \in R$ at the source node; $t_{r,f}^{in}$ is relative to the start of the last common cycle (T^{cyc}).

No-wait scheduling disallows frames to be queued on the routed path. Therefore, any two frames from two requests that are routed on a common physical link cannot be transmitted such that they overlap at any time. To avoid the overlap of two frames: $f_1 \in \mathcal{F}_{r_1}^{cyc}$ and $f_2 \in \mathcal{F}_{r_2}^{cyc}$ on physical link (n_i, n_j), either the transmission of f_1 should end before the start of f_2's transmission or the transmission of f_2 should end before the start of f_1's transmission. The pair of constraints in (6.2 and 6.3) ensures this condition.

$$t_{r_1,f_1}^{in} - t_{r_2,f_2}^{in} + M_1(\rho_{r_1,p_1,f_1,r_2,p_2,f_2}^{n_i,n_j}) + M_2(2 - \gamma_{r_1,p_1} - \gamma_{r_2,p_2}) \geq D_{p_2,n_i} - D_{p_1,n_i-1} - d_{n_i}^{proc}, \quad \forall r_1, r_2 \in R(r_1 \neq r_2), \forall f_1 \in \mathcal{F}_{r_1}^{cyc}, \forall f_2 \in \mathcal{F}_{r_2}^{cyc}, \forall p_1, p_2(\neq p_1) \in P_{r,K}, \forall (n_i, n_j) \in \epsilon_p. \tag{6.2}$$

$$t_{r_2,f_2}^{in} - t_{r_1,f_1}^{in} + M_1(1 - \rho_{r_1,p_1,f_1,r_2,p_2,f_2}^{n_i,n_j}) + M_2(2 - \gamma_{r_1,p_1} - \gamma_{r_2,p_2}) \geq D_{p_1,n_i} - D_{p_2,n_i-1} - d_{n_i}^{proc}, \quad \forall r_1, r_2 \in R(r_1 \neq r_2), \forall f_1 \in \mathcal{F}_{r_1}^{cyc}, \forall f_2 \in \mathcal{F}_{r_2}^{cyc}, \forall p_1, p_2(\neq p_1) \in P_{r,K}, \forall (n_i, n_j) \in \epsilon_p. \tag{6.3}$$

Here, $D(p_1, n_i)$ is the delay up to node n_i on path p_1 starting from the source node (n_r^{src}). M_1 and M_2 are arbitrary constants greater than the maximum value of the RHS in (6.2) and (6.3).
The indicator variable $\rho_{r_1,p_1,f_1,r_2,p_2,f_2}^{n_i,n_j}$ indicates the order of transmission of frames f_1 and f_2 on (n_i, n_j), when f_1 and f_2 are routed on the same physical link (n_i, n_j). If paths p_1 and p_2 are selected for routing requests r_1 and r_2, respectively, then $\gamma_{r_1,p_1} = \gamma_{r_2,p_2} = 1$. Moreover, if $\rho_{r_1,p_1,f_1,r_2,p_2,f_2}^{n_i,n_j} = 0$, the constraint in (6.3) becomes inactive, whereas the constraint in (6.2) is simplified to $t_{r_1,f_1}^{in} \geq t_{r_2,f_2}^{in} + constant$ thus f_1's transmission precedes f_2's. Conversely, if $\rho_{r_1,p_1,f_1,r_2,p_2,f_2}^{n_i,n_j} = 1$, the constraint in (6.2) becomes inactive, whereas the constraint in (6.3) is simplified to $t_{r_2,f_2}^{in} \geq t_{r_1,f_1}^{in} + constant$ thus f_2's transmission precedes f_1's. The second constraint in the pair is depicted in Fig. 6.5.

The transmission of any two frames $f_1, f_2 \in \mathcal{F}_r^{cyc}, f_1 \neq f_2$ of the same request $r \in R$ should not overlap with each other in the time domain. This condition is enforced by the constraint in (6.4); $\sum_{p \in P_{r,K}} \max_{(n_i,n_j) \in \epsilon_p}$ ensures that temporal overlap does not happen on any physical link of the routed

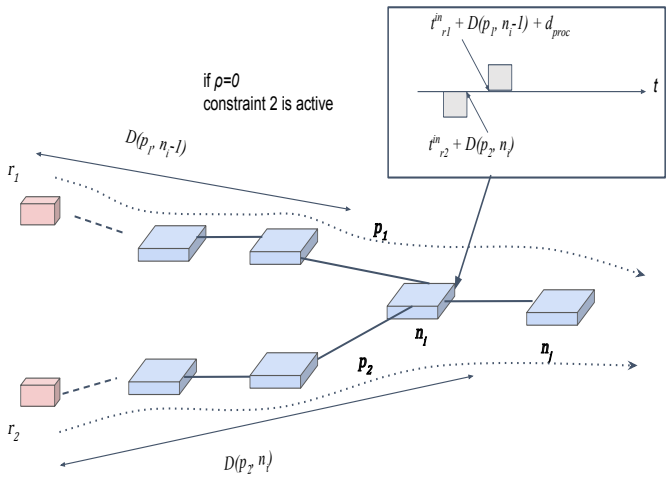


Figure 6.5: Overview of the frame overlap constraint. f_2 of r_2 is transmitted before f_1 of r_1 on common link (n_i, n_j).

path.

$$t^{in}_{r,f_1} + \sum_{p \in P_{r,K}} \max_{(n_i,n_j) \in \epsilon_p} \gamma_{r,p} d^{trans}_{r,n_i,n_j} \leq t_{r,f_2}, \quad \forall r \in R, f_1 \in \mathcal{F}^{cyc}_r, f_2 = f_1 + 1. \tag{6.4}$$

As $r \in R$ can have different time-periods, we schedule all requests for a common time period or cycle time T^{cyc}. However, for all requests, F_r frames should be scheduled during *their* own period T_r, i.e., the injection time of f should occur in the $\lfloor f/F_r \rfloor$'th time period. The constraint in (6.5) ensures this.

$$\lfloor f/F_r \rfloor T_r \leq t^{in}_{r,f} \leq (\lfloor f/F_r \rfloor + 1)T_r - \sum_{p \in P_{r,K}} \max_{(n_i,n_j) \in p} \gamma_{r,p} d^{trans}_{r,n_i,n_j}, \quad \forall r \in R, f \in \mathcal{F}^{cyc}_r. \tag{6.5}$$

The summation on the RHS of (6.5) ensures that the frame transmission finishes before the end of the period. The end-to-end delay of a request cannot exceed the corresponding application requirement. The constraint below (6.6) ensures that the end-to-end delays are bounded by the given limit.

$$(t^{in}_{r,f} - \lfloor f/F_r \rfloor T_r) + \sum_{p \in P_{r,K}} \gamma_{r,p} D_p \leq D^{e2e}_r, \quad \forall r \in R, \forall f \in \mathcal{F}^{cyc}_r. \tag{6.6}$$

The constraint in (6.7) ensures that the binary decision variables can only

take values in $\{0,1\}$ and the injection time is non-negative.

$$\begin{aligned} \gamma_{r,p}, \rho^{n_i,n_j}_{r_1,p_1,f_1,r_2,p_2,f_2} \in \{0,1\}, t^{in}_{r,f} \geq 0; \\ \forall r, r_1, r_2(\neq r_1) \in R, \forall p, p_1, p_2 \in P_{r,K}, \forall f, f_1, f_2(\neq f_1) \in \mathcal{F}^{cyc}_r \end{aligned} \tag{6.7}$$

6.4.2 Objective function

The objective function in (6.8) minimizes the maximum flow-span among all frames of all requests. For a given frame $f \in F^{cyc}_r$ of request $r \in R$, the flow-span is calculated with respect to the start of the period of f, i.e, $\lfloor f/F_r \rfloor T_r$).

$$obj : \min \max_{\forall r \in R, \forall f \in \mathcal{F}^{cyc}_r} (t^{in}_{r,f} - \lfloor f/F_r \rfloor T_r). \tag{6.8}$$

6.5 Heuristic

In this section, we describe the greedy-based heuristic, which is a scalable alternative to the ILP approach to solving the wired-wireless mixed network scheduling problem.

The greedy heuristic iteratively considers each TT flow request and attempts to schedule it on the given network. While scheduling a frame of a given request, the smallest injection time, for which there is no interference (i.e, no-wait constraint) from the frames of the already scheduled requests, is chosen. The pseudocode of the proposed greedy algorithm is discussed next in detail.

The procedure in Alg. 6.3 is the main procedure for the greedy heuristic. The given set of requests are first sorted based on *crit* criterion using procedure `reqs_order` (l. 3). Next, scheduling of each request in the sorted set is attempted via `sch_req` (l. 5). For example, a given set of requests can be sorted in the ascending order of the request period (T_r). After the request ordering step, requests with smaller T_r are scheduled first and the ones with longer T_r are scheduled last.

In Alg. 6.2, the frame injection times of the considered request are initialized with $\lceil f/F_r \rceil T_r$ (minimum possible injection time) (l. 2). The pair of `for` loops (l. 3-8) iteratively goes through the paths in $\forall p \in P_{r,K}$ and terminates if $\forall f \in \mathcal{F}^{cyc}_r$ can be scheduled on it and the injection time meets the end-to-end delay requirements. If a path can schedule all frames (l. 9), the path (p) and corresponding injection times (t^{in}_r) are returned; otherwise, ϕ, ϕ are returned.

In order to satisfy the no-wait constraint, no two frames should overlap in time at any of the common edges of their routed paths. To this end, we identify the time windows at the source of r that are forbidden, i.e, that

cause time overlap with other scheduled requests. Procedure `sch_fr_pth` (in Alg. 6.1) is responsible for computing injection time ($\geq t^{in}$) of a frame on path p, given frame injection times of other scheduled requests ($soln\_t^{in}$). t^{frb}, a list of forbidden injection time windows, is first initialized with [ ] (l. 2). For each frame (f^o) of the already scheduled request (r^o), the arrival time ($t^{arr}_{r^o,f^o}$) on an overlapping edge ($(n_i, n_j) \in$ `ovrlp_edges`(p, p^o)) is computed (l. 8). Using $t^{arr}_{r^o,f^o}$, forbidden time window(s) are computed by subtracting the path delay on p until the overlapping edge (l. 9) and added to list t^{frb} (l. 10-11). If the forbidden window continues beyond T_{cyc}, two windows per cycle are added to t^{frb} (l. 12-14). A feasible injection time ($\geq t^{in}$, $< t^{in} + T_r - d^{trans}$) that is not forbidden in t^{frb} is returned by `tin_frm_twnds` (l. 15).

Algorithm 6.1: Pseudo-code for the procedure responsible for determining an injection time for a frame.

```
1  Procedure sch_fr_pth(r, p, soln_pth, soln_t^in, t^in):
2      t^frb ← [ ] ; // sorted list of forbidden time windows at the
         source
3      for r^o ∈ soln_pth do
4          p^o ← soln_pth[r^o];
5          for (n_i, n_j) ∈ ovrlp_edges(p, p^o) do
6              d^trans_{r^o,n_i,n_j} ← (r^o.fsize)/B_{n_i,n_j};
7              for f^o ∈ [0, |F^cyc_{r^o}| − 1] do
8                  t^arr_{r^o,f^o} ← soln_t^in[r^o, f^o] + D_{r^o,p^o,n_i−1} + d^proc_{n_i} ;
                     // arrival time of f_o at node n_i
9                  t^src ≡ t^arr_{r^o,f^o} − D_{r,p,n_i−1} − d^proc_{n_i}(  mod T^cyc);
10                 if t^src + d^trans_{r^o,n_i,n_j} ≤ T^cyc then
11                     t^frb.append((t^src, t^src + d^trans_{r^o,n_i,n_j}));
12                 else
13                     t^frb.append((t^src, T^cyc));
14                     t^frb.append((0, d^trans_{r^o,n_i,n_j} − (T^cyc − t^src)));
15     return tin_frm_twnds(t^in, t^in + T_r, t^frb, D^e2e_r) ; // return
         smallest feasible injection time between t^in and
         t^in + T_r − d^trans
16 end
```

6.6 Evaluations

Both approaches– ILP and greedy, return a feasible solution for the mixed network wired-wireless scheduling problem that would guarantee that the

Algorithm 6.2: Pseudo-code for the procedure responsible for determining the routing path and frame injection times for a request.

```
1  Procedure sch_req(r, soln_pth, soln_t^in):
2      t_r^in ← {f : ⌊f/F_r⌋T_r | ∀f ∈ F_r^cyc} ;
3      for p ∈ P_{r,K} do
4          for f ∈ F_r^cyc do
5              t^in ← sch_fr_pth(r, p, soln_pth, soln_t^in, t_r^in[f]) ; // get
                 f's injection
6              if t^in ≠ φ and t^in + D_p ≤ D_r^e2e then t_r^in[f] ← t_in;
7              else break;
8          if t^in ≠ φ then return (p, t_r^in);
9      return φ, φ ; // no feasible schedule found on any path
10 end
```

Algorithm 6.3: Pseudo-code for the main procedure responsible for scheduling a given set of TT flow requests.

```
1  Procedure grd_sch_reqs(R, crit):
2      soln_pth, soln_t^in ← {}, {};
3      R_ord ← reqs_order(R, crit) ; // sort requests in R with crit
         criterion
4      for r ∈ R_ord do
5          p, t_r^in ← sch_req(r, soln_pth, soln_t^in);
6          if p ≠ φ then
7              soln_pth[r] ← p ; // store routed path
8              soln_t^in[r] ← t_r^in ; // store injection time of all frames
9      return soln_pth, soln_t^in ; // return greedy solution
10 end
```

timing requirement of all TT flow requests is met. However, an extensive evaluation of the two is needed to compare their performance and identify the scenarios where one is more beneficial than the other.
In this section, we discuss first the evaluation setup and then present the results of our evaluations for the two approaches in terms of scalability, scheduling performance and resource utilization.

6.6.1 Evaluation setup

The greedy heuristic was written in Python; whereas the ILP formulation is done in DOcplex, a native Python modeling library for optimization, and solved using the IBM's ILOG CPLEX solver [23]. For both approaches, NetworkX's Python API is used for the (pre-)computation of K shortest paths [24]. The experiments were performed on a PC with an Intel Core i5-8265U processor running @ 2.40GHz with 16GB of RAM memory running Ubuntu-18.04 OS.
Two kinds of topologies used for the evaluation– (i) ring topology (`RING`) and (ii) mesh topology (`MESH`) as shown in Fig. 6.6. The ring and mesh both consist of nine switch nodes, one WAP and twenty endpoint hosts. While the number of nodes in the ring and the mesh is the same, the different amount of connectivity is expected to impact the performance of the scheduling approach. To generate a request (r) in the set of TT flows (R), we choose two randomly chosen (different) nodes in the given topology as the endpoints (n_r^{src} and n_r^{dst}). The request's period (T_r) can be chosen randomly from set $\{1024, 2048, 4096\}(\mu s)$; thus the overall cycle time (T^{cyc}) is 4096μs. The number of frames per period is F_r is one and the frame size is randomly chosen from the set $\{50, 200, 500\}$B. The end-to-end delay requirement for the request is chosen as $2Tr$ for all requests.
The values/ranges of relevant evaluation parameters are listed in Tab. 6.3. In what follows, the labels– `ILP` and `GRD` refer to the ILP and greedy approach, respectively.

6.6.2 Results

Different evaluations were carried out to evaluate our approaches. We describe each evaluation and present the corresponding results separately. For each evaluation, fifty experiment iterations were performed and the observations are reported. In each experimental iteration, a total of $|R|$ TT flow requests are generated as explained previously.

Table 6.3: Default values/range of various parameters involved in the evaluation.

Parameter	Value or range	Units
Topology	`RING`, `MESH`	-
Topology size	9, 3x3	-
Request time period (T_r)	$\{2048, 4096, 8192\}$	μs
Request frame size ($fsize$)	$\{50, 200, 500\}$	B
Request e2e delay (D_r^{e2e})	$2T_r$	μs
Wired Tx speed (B^{wrd})	1000	Mbps
Wireless Tx speed (B^{wl})	9	Mbps
Node processing time (d_n^{proc})	50	μs

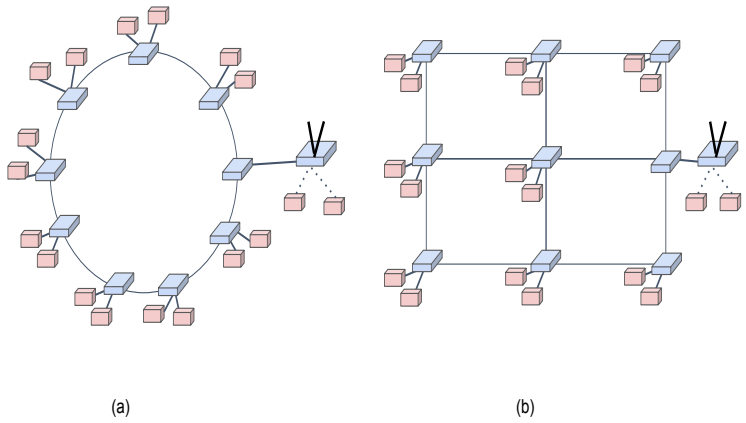


Figure 6.6: Topologies used for the evaluation: (a) `RING` and (b) `MESH`.

6.6.2.1 ILP and GRD Comparison

First, we compare the solve time, which is defined as the time required by the algorithm to find the solution, of the two approaches for the same problem instances. The solve time (in log scale) ILP and GRD when scheduling $|R| = 25$ requests are shown in Fig. 6.7. The solve time with MESH is more as compared to RING because $|P_{r,K}| \leq 2, \forall r \in R$ in RING whereas $|P_{r,K}| \geq 2$ for many requests in MESH. The greater number of paths between any two endpoints in MESH as compared to RING results in an extremely large solution space due to the explosion in the number of decision variables (γ and ρ).
Also, there are three orders of the magnitude speed difference between the two approaches– ILP and GRD. This can be attributed to the fact that ILP looks for an optimum solution whereas GRD returns a feasible greedy solution. Moreover, ILP appears to become impractical to solve for $|R| \geq 30$ on MESH where the CPLEX solver takes several hours to return the solution. Fig. 6.8 plots the evolution of three main CPLEX solver's parameters with time: (i) the objective value (solution to the relaxed ILP at the particular time), (ii) the integer solution and (iii) the lower bound (best achievable integer solution) with time. Clearly, the CPLEX solver has to traverse through the huge solution space of the problem though the returned (integer) solution was found quite early.
Next, we observe the maximum flow-span (time) percentage among all requests. Here, we define a request's flow-span time as the maximum delay in the injection time of the frame relative to the start of the cycle as a percentage of the request's cycle duration (T_r). The maximum flow-span indicates schedulability; the lower values of flow-span imply that the network has resources to schedule more requests and vice-versa. In Fig. 6.9, the y-axis is the total number of experiments (%-age) with the maximum flow-span time less than or equal to the corresponding x-axis value. It can be observed that the flow-span time for ILP is only slightly better (<5%) than GRD. As mentioned, this comes at the cost of an extremely long solve time of ILP.

6.6.2.2 GRD criteria

Next, we evaluate the performance of the GRD heuristic in detail. To this end, we consider the mixed network in MESH topology of size 10x10 with all alternate nodes of MESH connected with a WAP. The total number of TT flow requests ($|R|$) is 250.
We evaluate the impact of four request sorting criteria ($crit$)– RAND, ENDPOINT_-BW, BW and PERIOD_FSIZE on the GRD performance. The criteria are sum-

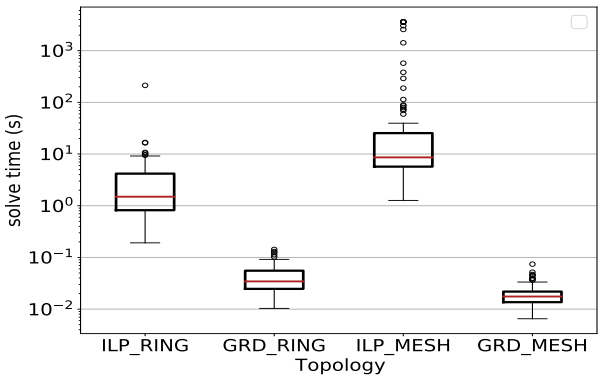


Figure 6.7: Box plot of the solve time (in log scale) for `ILP` and `GRD` on two topologies.

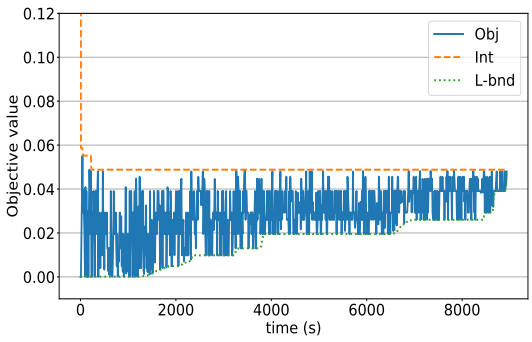


Figure 6.8: Plot showing the evolution of (i) the current objective value (Obj), (ii) the best integer solution value (int) and (iii) integer lower bound (L-bnd) with time for $|R| = 30$ on `MESH` of size 9.

marized in Tab. 6.4. Fig. 6.10 shows that the `PERIOD_FSIZE` criterion has the best performance while `ENDPOINT_BW` has the worst performance after `RAND` among all criteria. Scheduling the requests with smaller periods (T_r) is challenging as the frame injection times (l. 20, Alg. 6.1) for such requests are more constrained than the requests with larger T_r. Among the requests that have the same T_r, the requests with larger $fsize_r$ are selected first to be scheduled. In `ENDPOINT_BW`, requests are sorted on the basis of the transmission speed of their endpoints; the requests having wireless endpoints (lower speeds) are selected first for scheduling.

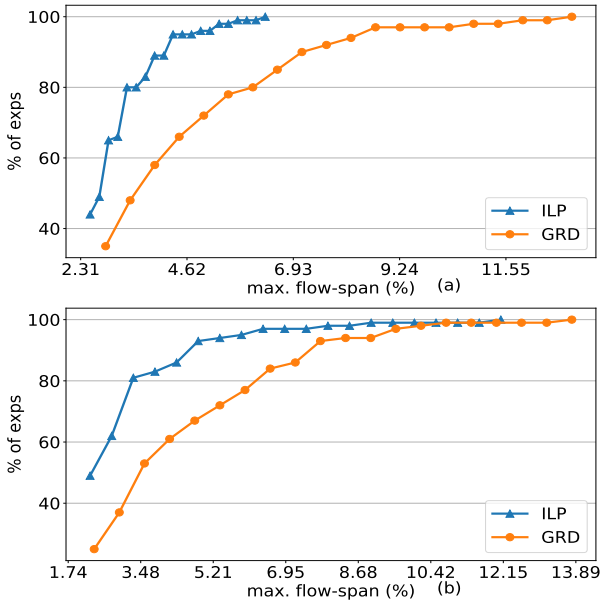
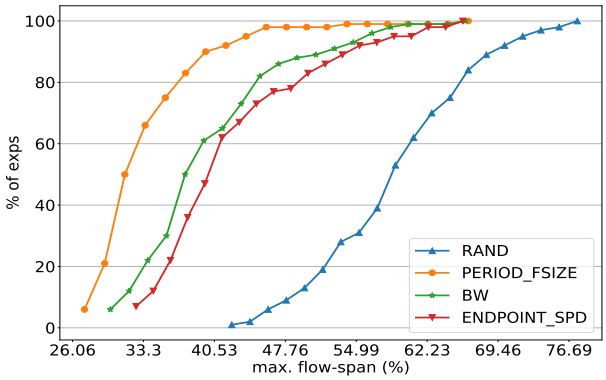


Figure 6.9: The percentage of experiments versus the maximum flow-span time for (a) `RING` and (b) `MESH` topologies.

Table 6.4: Overview of various greedy criterion.

Criterion	Description
`RAND`	Random ordering of requests.
`ENDPOINT_BW`	Ordering of requests in the ascending order of endpoint bandwidth.
`BW`	Ordering of requests in the descending order of requests normalized bandwidth ($fsize_r/T_r$).
`PERIOD_FSIZE`	Ordering of requests in the ascending order of request period; the ties are broken based on the request frame-size ($fsize_r$).



.

Figure 6.10: The percentage of experiments versus the maximum flow-span time for various greedy criteria on `MESH` *with size 10x10 and* $|R| = 250$

6.6.2.3 Wireless Requests Scheduling

For wired-wireless mixed networks, it is essential to evaluate the scheduling algorithm's performance for wireless TT flow requests. For a given number of total requests ($|R|$), we vary the fraction of requests that have wireless endpoints and observe the impact on the time allocation for TT flow requests, the maximum flow-span time and the average GCL length. The mixed network considered for evaluation is the same as the one considered in Section 6.6.2.2.

The impact of %-age of wireless TT flow requests ($|\bar{R}|_{wl}$) on the average time-allocation per-link for TT flow requests is shown in Fig. 6.11 for $|R| = 250$. The height of the blue bar indicates the average time spent in the transmission of TT flow requests and the average guard time for these flows is indicated by the height of the orange bars. The sum of these heights indicates the time allocated by `GRD` to TT flow requests. The rest of the time in a cycle (height of green bars) is the time in which BE flows can be transmitted. Clearly, as $|\bar{R}|_{wl}$, the average time allocated for TT flow requests increases and the BE flow time is reduced. This is because of the longer wireless transmission delays as compared to the wired transmission delays. Therefore, for a given value of $|R|$, a higher number of $|\bar{R}|_{wl}$ results in more packets traversing wireless links thus increasing the average TT allocation time. Furthermore, the %-age of guard time (inefficiency in allocation) increases slightly with $|\bar{R}|_{wl}$.

Fig. 6.12 depicts the impact of ($|\bar{R}|_{wl}$) on the maximum flow-span time for $|R| = 250$. The increase in the %-age of wireless requests is accompanied by an increase in the maximum flow-span time. This stems from the fact the

large value of $|\bar{R}|_{wl}$ results in longer allocation time for TT flow requests thus increasing the maximum flow-span time (cfr. Fig. 6.11). Further, around $|\bar{R}|_{wl} = 80\%$, the maximum flow-span reaches 90% thus indicating high contention for the network bandwidth, especially in wireless links.
The shared memory available on a switch is a precious resource and for some chip architectures, it limits the total number of GCL entries per switch (or port) [25]. The average number of GCL entries (GCL_{len}) resulting from the schedule generated `GRD` is plotted in Fig. 6.13. Although, the number of requests ($|R|$) remains constant, by varying $|\bar{R}|_{wl}$, GCL_{len} increases. This is because the number of requests whose schedule cannot be aligned with the already scheduled requests increases with $|\bar{R}|_{wl}$. This observation also corresponds to the increase in guard time with $|\bar{R}|_{wl}$ (Fig. 6.11).

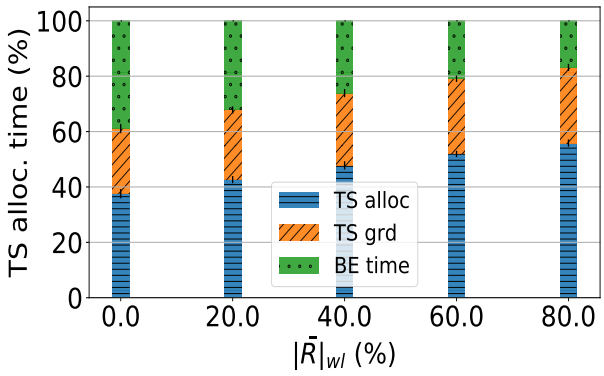


Figure 6.11: The percentage of time allocation for TT flow requests (blue), guard time (orange) and BE traffic (green) versus the fraction of wireless requests.

6.7 Conclusion

Supporting time-sensitive networking in hybrid wired-wireless networking is becoming more and more crucial because of growing applications such as smart factories and Industry 4.0. In this paper, we presented the problem of scheduling in wired-wireless mixed networks. The problem is first defined and then two approaches were discussed to solve it. `ILP` results in an optimal solution but does not scale beyond a small-sized problem instance, whereas `GRD` provides a reasonably good solution and is scalable to larger problem instances.
In our future work, we would like to address the online scheduling problem in TSN, where TT flow requests arrive in an online fashion and are scheduled one by one with the possibility of re-scheduling the already deployed

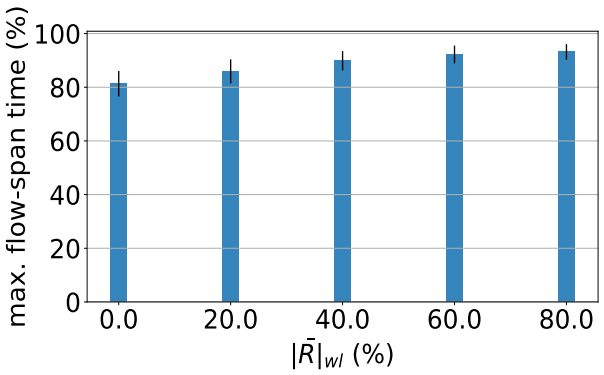


Figure 6.12: The maximum flow-span time (%-age of cycle time) versus the fraction of wireless requests.

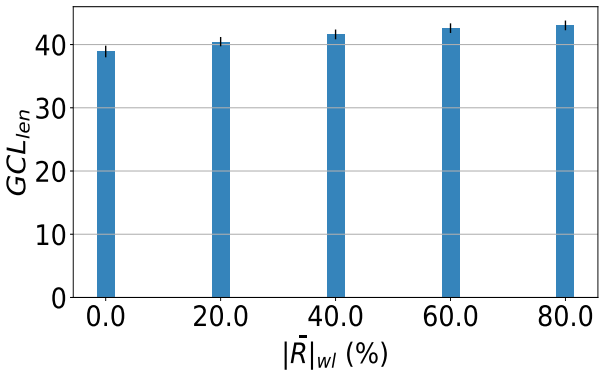


Figure 6.13: The average number of GCL entries (per-node) versus the fraction of wireless requests.

requests. In addition, the scheduling problem in a softTSN network, where the processing delays are not fixed but are bounded within a range, shall be addressed.

6.8 Acknowledgements

This research was funded by the Flemish FWO SBO S003921N VERI-END.com (Verifiable and elastic end-to-end communication infrastructures for private professional environments) project, by the FWO project under grant agreement #G055619N and the Flemish Government under the "Onderzoeksprogramma Artificiele Intelligentie (AI) Vlaanderen".

References

[1] I. S. Association et al. *IEEE Standard for Local and Metropolitan Area Networks—Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*. IEEE Std, 802.

[2] *IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic*. IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015), pages 1–57, 2016. doi:10.1109/IEEESTD.2016.8613095.

[3] L. Zhang and S. Zeadally. *Enabling end-to-end QoS over hybrid wired-wireless networks*. Wireless Personal Communications, 38(2):167–185, 2006.

[4] J. Haxhibeqiri, X. Jiao, E. Municio, J. M. Marquez-Barja, I. Moerman, and J. Hoebeke. *Bringing Time-Sensitive Networking to Wireless Professional Private Networks*. Wireless Personal Communications, 121(2):1255–1271, 2021.

[5] X. Jiao, W. Liu, M. Mehari, M. Aslam, and I. Moerman. *openwifi: a free and open-source IEEE802. 11 SDR implementation on SoC*. In 2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring), pages 1–2. IEEE, 2020.

[6] F. Dürr and N. G. Nayak. *No-wait packet scheduling for IEEE time-sensitive networks (TSN)*. In Proceedings of the 24th International Conference on Real-Time Networks and Systems, pages 203–212, 2016.

[7] S. S. Craciunas, R. S. Oliver, and W. Steiner. *Formal scheduling constraints for time-sensitive networks*. arXiv preprint arXiv:1712.02246, 2017.

[8] E. Schweissguth, P. Danielis, D. Timmermann, H. Parzyjegla, and G. Mühl. *ILP-based joint routing and scheduling for time-triggered networks*. In Proceedings of the 25th International Conference on Real-Time Networks and Systems, pages 8–17, 2017.

[9] D. Tămaş-Selicean, P. Pop, and W. Steiner. *Design optimization of TTEthernet-based distributed real-time systems*. Real-Time Systems, 51(1):1–35, 2015.

[10] D. Tamas-Selicean, P. Pop, and W. Steiner. *Synthesis of communication schedules for TTEthernet-based mixed-criticality systems.* In Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, pages 473–482, 2012.

[11] *IEEE Draft Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications.* IEEE P802.1AS-Rev/D6.0 December 2017, pages 1–496, 2018.

[12] *IEEE Standard for Local and metropolitan area networks– Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams.* IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005), pages 1–72, 2010. doi:10.1109/IEEESTD.2010.8684664.

[13] *IEEE Standard for Local and metropolitan area networks— Bridges and Bridged Networks - Amendment 24: Path Control and Reservation.* IEEE Std 802.1Qca-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qcd-2015 and IEEE Std 802.1Q-2014/Cor 1-2015), pages 1–120, 2016. doi:10.1109/IEEESTD.2016.7434544.

[14] *IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing.* IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016), pages 1–65, 2017. doi:10.1109/IEEESTD.2017.8064221.

[15] *IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability.* IEEE Std 802.1CB-2017, pages 1–102, 2017. doi:10.1109/IEEESTD.2017.8091139.

[16] Z. Hanzálek, P. Burget, and P. Šucha. *Profinet IO IRT message scheduling.* In 2009 21st Euromicro Conference on Real-Time Systems, pages 57–65. IEEE, 2009.

[17] D. Hellmanns, L. Haug, M. Hildebrand, F. Dürr, S. Kehrer, and R. Hummen. *How to Optimize Joint Routing and Scheduling Models for TSN Using Integer Linear Programming.* In Proc. ACM Int. Conf. Real Time Netw. Syst., Nantes, France, 2021.

[18] M. Pahlevan and R. Obermaisser. *Genetic algorithm for scheduling time-triggered traffic in time-sensitive networks.* In 2018 IEEE 23rd

international conference on emerging technologies and factory automation (ETFA), volume 1, pages 337–344. IEEE, 2018.

[19] Y. Wang, J. Chen, W. Ning, H. Yu, S. Lin, Z. Wang, G. Pang, and C. Chen. *A time-sensitive network scheduling algorithm based on improved ant colony optimization.* Alexandria Engineering Journal, 60(1):107–114, 2021.

[20] D. Cavalcanti, J. Perez-Ramirez, M. M. Rashid, J. Fang, M. Galeev, and K. B. Stanton. *Extending Accurate Time Distribution and Timeliness Capabilities Over the Air to Enable Future Wireless Industrial Automation Systems.* Proceedings of the IEEE, 107(6):1132–1152, 2019. doi:10.1109/JPROC.2019.2903414.

[21] T. Adame, M. Carrascosa-Zamacois, and B. Bellalta. *Time-sensitive networking in IEEE 802.11 be: On the way to low-latency WiFi 7.* Sensors, 21(15):4954, 2021.

[22] D. Ginthör, R. Guillaume, J. von Hoyningen-Huene, M. Schüngel, and H. D. Schotten. *End-to-end Optimized Joint Scheduling of Converged Wireless and Wired Time-Sensitive Networks.* In 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), volume 1, pages 222–229. IEEE, 2020.

[23] IBM. *IBM ILOG CPLEX Optimization Studio.* Available from: https://www.ibm.com/analytics/cplex-optimizer.

[24] A. A. Hagberg, D. A. Schult, and P. J. Swart. *Exploring Network Structure, Dynamics, and Function using NetworkX.* In G. Varoquaux, T. Vaught, and J. Millman, editors, Proceedings of the 7th Python in Science Conference, pages 11 – 15, Pasadena, CA USA, 2008.

[25] D. Hellmanns, A. Glavackij, J. Falk, R. Hummen, S. Kehrer, and F. Dürr. *Scaling TSN scheduling for factory automation networks.* In 2020 16th IEEE International Conference on Factory Communication Systems (WFCS), pages 1–8. IEEE, 2020.

7 Conclusion

The remarkable growth in the traffic flowing through communication networks along with the reducing ARPU has forced telecom operators to transform the way network services are created and managed. Traditional architectures that relied heavily on specialized hardware are evolving towards architectures where virtual services are hosted over inexpensive COTS hardware. The broadcast industry is also witnessing a similar evolution. On the one hand, the transport of uncompressed media that is traditionally done via SDI routers is now done through IP networking. On the other hand, media processing functionality that is usually implemented via expensive hardware is now being virtualized and run over COTS hardware.

The first part of the dissertation dealt with optimization algorithms concerning the deployment of virtual network and media services. Telecom operators and broadcasters foresee significant cost reduction by adopting COTS-based architectures. Efficient resource utilization is key to cost reduction, thus optimization algorithms for virtual service deployment were proposed in this dissertation.

The algorithms proposed in the first part do not make any end-to-end latency or jitter guarantees. In order to let real-time traffic coexist with the standard best-effort traffic, a type of packet scheduling is needed. In the second part of this dissertation, the problem of end-to-end packet scheduling for time-sensitive applications is investigated and algorithms were proposed to solve the scheduling problem.

In the following sections of this chapter, we conclude with the key research contributions in this dissertation and discuss potential future research directions.

7.1 Virtualized Service Deployment Algorithms

7.1.1 Research Contributions

Two key challenges before the adoption of COTS hardware-based architectures are their inferior performance and higher power consumption in contrast to the architectures based on middleboxes [1]. Several performance and power optimization solutions have been proposed by researchers [2]. One such solution is to use externally connected hardware accelerators (e.g., GPUs, FPGAs) to offload compute-intensive VNF operations from the CPUs running the VNF. Due to the reduction in CPU usage, additional VNFs can be hosted on the same NFVi but enhanced with hardware accelerators. For instance, offloading of AES and SHA operations from an SSH client VNF to an FPGA accelerator results in a throughput improvement of 62%, as demonstrated in Appendix A.
As hardware accelerators are expected to be increasingly incorporated in the NFVi layer, it is required that the resource allocation processes in the MANO layer take into account their presence [3]. Otherwise, the inefficient allocation would not result in cost reductions as envisaged by telecom operators when adopting NFV. The VNF-PC procedures existing in the literature are mostly agnostic to both the acceleration requirements of VNFs and the locality of hardware accelerators in the NFVi layer [1]. As a result, the chosen VNF-PC algorithm can result in inefficient utilization of NFVi resources. Therefore, the VNF-PC algorithm needs to be altered to make it to take into account hardware accelerators. To this end, we addressed the problem of VNF-AAPC in Chapter 2. First, we presented an ILP formulation of the problem. The ILP formulation is a single-step approach that jointly optimizes VNF placement, chaining and accelerator allocation resulting in an optimal deployment. Next, we developed an accelerator-aware heuristic also to solve the VNF-AAPC problem. The heuristic performs VNF-PC in two phases– first, accelerate-able VNFs are deployed on the nodes containing accelerators and then, the rest of VNF-chain segments are deployed in a hierarchical manner. In simulation experiments, the trade-off between the ILP approach and the heuristic is observed. First, the two approaches were compared in terms of their execution times. The ILP approach is more than three orders of magnitude slower than the heuristic thus infeasible for realistic network topology. Although the heuristic results

in a suboptimal solution, the gap between the two approaches is less than 5% in terms of total nodes cost. In spite of its slightly better performance, the ILP approach is impractical for reasonably sized problem instances. In order to access the efficiency of the accelerator-aware VNF-PC heuristic, its performance was compared with the accelerator-agnostic VNF-PC heuristic. The results clearly demonstrated that the accelerator-aware heuristic results in a nearly 10-15% reduction in the total nodes cost compared to the accelerator-agnostic heuristic. This node cost reduction can be attributed to the improved VNF consolidation as a result of the accelerator-aware VNF-PC. Overall cost analysis shows that in order to achieve significant cost savings, efficient accelerator-aware VNF-PC algorithms are required even though accelerator costs are expected to fall in the future.

The VNF-AAPC procedures presented in Chapter 2 are static, i.e., once an accelerator is allocated to a VNF it is not reallocated to another VNF despite the change in the traffic flowing through the VNF. In Appendix A, a procedure for dynamic allocation of hardware accelerators to VNFs is presented. We have implemented an SSM, as a component of MANO, which feeds on the resource utilization information of VNFs and determines which VNF to allocate available accelerator resources. The VNFs in this case are SSH clients whose cipher (AES) and hash (SHA) operations were accelerated using FPGA-based accelerators. An improvement of 42.1% and 61.9% in throughput for AES128 and AES256, respectively, was noticed because of the hardware acceleration of SSH-client VNFs. In addition, dynamic provisioning of hardware accelerator cores to VNFs is achieved based on their real-time CPU usage.

A failure in the NFVi can result in the disruption of multiple network services implemented via VNF-chains. After detecting such an event, VNF-chains are recovered by re-allocating NFVi resources. This implies that not only VNFs are re-assigned to appropriate server nodes but accelerator resources are also allocated. We addressed the prioritized VNF-chain recovery problem in NFV environments containing hardware accelerators in Appendix B. The problem was modeled first as an ILP whose objective function was to maximize the total traffic restored after a failure, subject to new resource constraints. The restoration is accompanied by the preference for high-priority VNF-chains. Due to the impracticality of the ILP approach for solving the problem for large instances, a greedy-heuristic was also proposed to solve the problem for a sub-optimal solution. However, the performance of the greedy heuristic was found to be on par with the ILP approach as far as the restoration of high and medium priority VNF-chains is concerned, whereas a slight gap is observed for low-priority VNF-chains. In addition, it was observed that accelerator-aware restoration results in

about 30% more traffic restored than accelerator-agnostic restoration.
Traditionally, media transport and processing in broadcast studios is achieved using specialized hardware appliances [4]. Due to the high costs of hardware media appliances, akin to middleboxes, upgrading the studio infrastructure for newer and high-quality formats requires substantial investments. To reduce total CAPEX and OPEX, the broadcast industry is transitioning towards architectures based on COTS hardware. The transition is two folds– (i) in the media transport domain from SDI to IP and (ii) in the media processing domain from specialized hardware to VMFs. Taking inspiration from NFV, we proposed the idea of MFV in Chapter 3. MFV is an architecture where media transport and processing are based on COTS hardware. Apart from cost reduction and other advantages, MFV, offers opportunities that were not available with the traditional studio architecture. For instance, a high-bandwidth video stream can be split into multiple independently switched low-bandwidth video streams. Stream decomposition together with virtualization can be exploited to decompose a given virtualized media service. In Chapter 3, we proposed a procedure to obtain the decomposition of the given virtual media service's VMF-FG. Such an algorithm is used to transform the VMF-FG to an optimized VMF-FG, which when deployed can potentially result in better resource utilization. For VMF-FG deployment, two VMF-PC algorithms were also proposed. NFPC is the first VMF-PC algorithm that is based on the next-fit approach; (decomposed or not) VMF-FG is traversed starting from the sink node while each considered VMF is deployed on the next available server node and chained with its upstream VMF nodes. k-cutPC is an alternative VMF-PC that results in the reduction of total bandwidth usage over the NFPC approach. The given VMF-FG is first partitioned into k components and then each component is deployed using NFPC. The partitioning is done such that the total virtual link bandwidth between partitioned components is minimum. The simulation results revealed an improvement in resource utilization as a result of VMF-FG decomposition. In particular, the total server nodes reservation and CPU utilization improvement were shown as a result of VMF decomposition by 20% and 10%, respectively. Furthermore, VMF-FG decomposition resulted in a slight reduction in network bandwidth usage. As k-cutPC attempts the deployment of a partitioned VMF-FG, which has minimum inter VMF-FG component bandwidth, it has better bandwidth utilization than NFPC. Finally, the latency in terms of end-to-end hops of a deployed VMF-FG is also reduced with VMF-FG decomposition.

7.1.2 Future Directions

As mentioned before, producing broadcast-quality content demands that absolute QoS parameters, i.e., zero packet loss, bounded latency and jitter, are respected with regards to both media transport and processing. The challenge of deterministic transport of media streams in IP can be addressed by employing, e.g., CSQF-based scheduling, in the network nodes. However, the other challenge, i.e., the processing of multi gigabits per second media streams via VMFs in real-time still needs to be tackled. Real-time processing in VMF is challenging because of the unpredictable delays inside the Linux networking stack. Various kernel-bypass mechanisms such as DPDK, netmap, etc, have been proposed to enhance the ability of VMFs to handle data rates of more than tens of Gbps [5], [6]. By using these mechanisms, packets can be transferred directly to userspace applications from the NIC with minimum involvement of the kernel. Although these optimizations have been proven to be useful for VNF implementation, they still need to be benchmarked for their broadcast-specific performance (e.g., packet loss, latency, jitter). In addition to software optimizations focused on the networking part, actual processing at the frame-level (e.g., color correction, picture-in-picture) can be offloaded to an attached hardware accelerator. As GPUs have often been employed in video processing applications, their role for VMF acceleration needs to be investigated. Particularly, GPU virtualization will be important in such scenarios where VMFs based on VMs or Docker containers might want to share the attached GPU [7].

Architectures based on MFV are also expected to provide the same, if not more, reliability as was ensured by SDI. Especially, in live events as opposed to recorded events, it is paramount that the production quality is not impacted by one or more failures in the infrastructure. This entails sufficient redundancy for media transport (in the network) as well as for media processing (in VMFs). Fully redundant media transport can be achieved by exploiting seamless switching based on $1 + 1$ protection as described in the SMPTE 2022-7 standard [8]. Similarly, resilience for media processing can be ensured by replicating VMFs across multiple server nodes. Future research can therefore focus on VMF placement and scheduling algorithms that can ensure a given level of availability.

Traditionally, on-premise studio resources are reserved according to the peak utilization, not the average utilization. This requires huge investment in acquiring, deploying and managing expensive hardware. Recently, broadcasters have shown interest in outsourcing some types of production workflows to the cloud. For instance, Amazon Web Services (AWS) Elemental already offers transcoding functions for live video [9]. It is reasonable to expect that such cloud services will offer a wide range of media processing functional-

ities in the coming years. Adhering to SMPTE 2110-21 constraints when transporting media streams between the on-premise facility and the cloud shall be challenging. Moreover, requirements specific for broadcasting such as IGMP multicast, PTP synchronization, 2022-7 resilience, etc, are not currently provided by typical cloud scenarios [10], [11]. These challenges must be addressed to ensure the on-premise plus cloud hybrid model can be adopted.
Finally, ensuring QoS in such a complex system, which has multiple hardware and software components interacting with each other, will be challenging. Mathematical frameworks like network calculus can be exploited to model such systems so that their QoS can be argued analytically [12].

7.2 Packet Scheduling Algorithms

7.2.1 Research Contributions

The two VMF-PC algorithms proposed in Chapter 3 assume best-effort networking for media transport on a virtual link between two deployed VMFs. This implies broadcast-level QoS KPIs such as packet loss, delay, jitter, etc, cannot be guaranteed with such algorithms. Therefore, there is a need to have packet scheduling for each virtual link in the given VMF-FG so that broadcast-quality guarantees can be made for the deployed virtual media services. A mechanism such as CSQF can be exploited to schedule packet transmissions along the network path between any two adjacent VMFs so that end-to-end guarantees can be made for the complete VMF-FG. In Chapter 4, the VMF-FG scheduling problem was first formulated and proved for its NP-completeness. Next, a greedy heuristic, based on the BFS and DFS graph traversal algorithms, was proposed, whose objective was to find packet schedules for a given VMF-FG in a time-efficient manner. The evaluation of the heuristic shows an increase in the end-to-end delay with increasing cycle time. We also highlighted the improvement in the end-to-end delay with VMF-FG decomposition, particularly for high-quality formats.
Packet scheduling mechanisms such as CSQF ensure that end-to-end latency is bounded by making the queuing process deterministic in each network node along the path from the flow's source to destination. In addition, packet scheduling also ensures that queue occupancy does not increase indefinitely with time, thus preventing packet losses due to congestion. However, flows can still be affected due to the failures in the network (e.g., switch failure due to memory error or link failure due to a fiber break) but can be

prevented by employing dedicated path protection schemes such as $1 + 1$ protection. In Chapter 5, we investigated the problem of $1 + 1$ RTSCH for DetNets flows. The problem takes into account the end-to-end delays along the selected paths to ensure reliable recovery at the destination. The problem is addressed using three approaches– (i) ILP, (ii) Greedy and (iii) Tabu-search. The simulation results revealed the trade-offs between the average traffic accepted versus the solve time for these approaches. Although the ILP approach had the highest average accepted traffic, it was found to be not scalable beyond small networks. On the other hand, the greedy approach was the fastest approach but had the lowest average traffic accepted. The Tabu-search approach provided a compromise between the ILP and Greedy approach; problem instances can be solved in a few seconds which the ILP approach would require several hours to solve but only at a small cost ($< 10\%$) of average traffic accepted. The results also indicated that increasing the cycle time (T_c) decreases the end-to-end delay but reduces the average traffic accepted.
The wired-only networks considered in Chapter 4 and 5 could easily promise end-to-end timing guarantees by employing TSN/DetNet mechanisms. However, due to the increasing demand to support applications such as smart factories and Industry 4.0, it is essential that end-to-end guarantees can also be made in wired-wireless mixed networks [13]. The problem of end-to-end scheduling in wired-wireless mixed networks was investigated in Chapter 6. The problem was modeled as an ILP and a greedy heuristic was proposed to solve the reasonably sized instances of the problem. The simulation results showed that the greedy approach was more than three orders of magnitude faster than the ILP approach, whereas its performance in terms of flow-span time was only 5% inferior as compared to ILP. The impact of request sorting criteria on greedy performance was also highlighted. Finally, the impact of wireless requests on the time allocation for TT flow requests, the maximum flow-span time and the average GCL length were reported.

7.2.2 Future Directions

TSN has enabled Ethernet-based networks to support both time-sensitive applications that are associated with hard timing deadlines as well as standard best-effort applications that have no such constraints. Normally, these mechanisms are implemented in hardware TSN switches because of deterministic latency requirements. In recent years, attempts have been made to make software networking (e.g. Linux's) stacks more predictable. Linux's queuing (qdisc) discipline called Time-aware Priority Shaper (TAPRIO) is one such attempt to implement Time Division multiplexing in Linux [14].

These TSN mechanisms might be useful not only for software TSN emulators but also important in real-life applications. For example, when moving the physical Programmable Logic Controller (PLCs) to the cloud, TSN-enabled switching will be required in the Linux networking stack to connect actuators and sensors in the factory to the vPLC in the cloud [15].
A preliminary investigation conducted on the Linux bridge has revealed that its processing delay is sensitive to various system configurations like CPU power setting, TLB efficiency, etc. Thus, a proper benchmarking of the Linux networking stack is required to come up with an optimal configuration suited for real-time applications. In addition, the packet scheduling algorithms (e.g., in Chapter 4-6) also need to be adapted so that the unpredictable processing delay in a software networking stack does not result in the violation of a TT flow's QoS. Particularly, taking into account the variation in processing delay, appropriate guard times need to be inserted in the GCL. The proposed algorithms should be accompanied by the worst-case analysis for end-to-end latency and jitter values.
A change in an assembly line or migration of vPLC from one server node to another in the cloud requires re-configuring or re-scheduling TT flows. The scheduling problem considered in this dissertation works in an offline fashion, i.e., the set of all requested TT flows and the corresponding parameters are known beforehand. Thus, offline scheduling is not a good suit for such scenarios and these requests should be handled one by one. This leads to the problem of online scheduling of TT flow requests. Moreover, the already scheduled TT flow requests might require re-scheduling periodically so that the network is able to accept as many requests as possible. However, it needs to be guaranteed that while the TT flow is being re-scheduled, its QoS requirements are not violated, i.e., there is no excessive packet delay and no packet loss during re-scheduling. This requirement can be met by ensuring that packets are still scheduled on both the old and the new path while doing re-scheduling. Similar to $1 + 1$ protection, dynamic activation (deactivation) of duplication and elimination functions at source and destination nodes, respectively, shall be required [16]. To summarize, reliable re-scheduling entails a dynamic network along with an efficient scheduling algorithm. Further study focusing on the problem of dynamic scheduling of online TT flow requests is required.
The scheduling problem addressed in Chapter 6 assumed that the wireless data rate is much lower than the maximum possible data rate. This assumption ensures that the packet loss probability is reduced thus ensuring the higher reliability of wireless links. However, the scheduled packet transmissions on a wireless link can still collide with the transmission from an external wireless source. As a consequence, a successful end-to-end deliv-

ery might require one or more transmissions in wireless links as opposed to wired links where a single transmission is enough. This necessitates that the scheduling algorithm allocates time slots for these re-transmissions. Future research can focus on building scheduling algorithms that ensure that a TT flow request's reliability requirement is met by allocating the required number of re-transmission time slots on wireless links.

References

[1] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. *Network function virtualization: State-of-the-art and research challenges.* IEEE Communications surveys & tutorials, 18(1):236–262, 2015.

[2] L. Linguaglossa, S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, and G. Bianchi. *Survey of performance acceleration techniques for network function virtualization.* Proceedings of the IEEE, 107(4):746–764, 2019.

[3] Z. Bronstein, E. Roch, J. Xia, and A. Molkho. *Uniform handling and abstraction of NFV hardware accelerators.* IEEE Network, 29(3):22–29, 2015.

[4] S. Sneddon, C. Swisher, and J. Mayzurk. *Large Scale Deployment of SMPTE 2110: The IP Live Production Facility.* In SMPTE 2019, pages 1–31. SMPTE, 2019.

[5] The Linux Foundation. *Data Plane Development Kit (DPDK).* Available from: https://www.dpdk.org/.

[6] L. Rizzo. *netmap: a novel framework for fast packet I/O.* In 21st USENIX Security Symposium (USENIX Security 12), pages 101–112, 2012.

[7] NVIDIA. *RDG: OpenStack SMPTE 2110 Media Streaming Cloud with NVIDIA Network Hardware Offload*, 2019. Available from: https://docs.nvidia.com/networking/pages/releaseview.action?pageId=18484492.

[8] *ST 2022-8:2019 - SMPTE Standard - Professional Media Over Managed IP Networks: Timing of ST 2022-6 Streams in ST 2110-10 Systems.* ST 2022-8:2019, pages 1–10, 2019. doi:10.5594/SMPTE.ST2022-8.2019.

[9] AWS. *Overview of Amazon Web Services*, 2019. Available from: https://docs.aws.amazon.com/whitepapers/latest/aws-overview/aws-overview.pdf#media-services.

[10] *IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks.* IEEE Std 802.1AS-2011, pages 1–292, 2011. doi:10.1109/IEEESTD.2011.5741898.

[11] S. Standard. *Professional Media Over Managed IP Networks: Traffic Shaping and Delivery Timing for Video*. SMPTE Standard ST, pages 2110–21, 2017.

[12] J.-Y. Le Boudec and P. Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer, 2001.

[13] J. Haxhibeqiri, X. Jiao, E. Municio, J. M. Marquez-Barja, I. Moerman, and J. Hoebeke. *Bringing Time-Sensitive Networking to Wireless Professional Private Networks*. Wireless Personal Communications, 121(2):1255–1271, 2021.

[14] Linux. *tc-taprio(8) — Linux manual page*, 2018. Available from: https://man7.org/linux/man-pages/man8/tc-taprio.8.html.

[15] M. Azarmipour, H. Elfaham, C. Gries, and U. Epple. *Plc 4.0: A control system for industry 4.0*. In IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society, volume 1, pages 5513–5518. IEEE, 2019.

[16] *IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability*. IEEE Std 802.1CB-2017, pages 1–102, 2017. doi:10.1109/IEEESTD.2017.8091139.

A

Dynamic hardware-acceleration of VNFs in NFV environments

Chapter 2 discusses the VNF-AAPC problem whose solution gives a fixed mapping of accelerators to VNFs. For dynamic NFV environments where traffic flows through VNFs varies with time, a static allocation of accelerators to VNFs results in inefficient resource utilization. However, by allocating accelerators on the basis of instantaneous VNF requirements better resource utilization can be achieved.
In this appendix, we propose and experimentally evaluate a solution that dynamically shares hardware accelerators between multiple VNFs based on real-time CPU usage. The proposed solution supplements the contributions made in Chapter 2.

⋆ ⋆ ⋆

G.P. Sharma, W. Tavernier, D. Colle, and M. Pickavet

Published in the Proceedings of the International Conference on Software Defined Systems (SDS), Rome, Italy, 2019, pp. 254–259

Abstract In this paper, we describe a scheme for dynamically provisioning hardware accelerator resources to virtual network functions (VNF) in

an NFV environment. The scheme involves collaboration between various NFV components like the Service-specific Manager (SSM) and Element-management Systems (EMSs) for the management of accelerator resources. Accelerator resources are dynamically allocated to VNFs based on their resource usage information. We present the performance comparison of non-accelerated and accelerated SSH client VNFs. We also demonstrate switching of accelerator resources between the concurrently running SSH tunnels which is triggered by a change in the nature of the data traffic flowing through SSH tunnels.

A.1 Introduction

Network services are conventionally deployed using specialized and proprietary hardware appliances called middleboxes. The objective of Network function virtualization (NFV) is to decouple the packet-processing functionality of these middleboxes from the underlying hardware so that standard IT virtualization technologies can be utilized to execute network functions on general-purpose x86 or ARM servers. NFV has enabled the faster deployment of new network services along with a reduction in capital and operational expenditures. Despite all the benefits that NFV offers, it still faces obstacles towards its widespread acceptance by telecom operators. The biggest challenge is to achieve the same virtual network function (VNF) performance as offered by its hardware counterpart [1]. To overcome this challenge, the use of hardware accelerators (GPUS, FPGAs, smartNICs) in conjunction with general-purpose processors has been advocated.
In the process of migration towards virtual packet-processing implementations from the fixed-hardware implementation, re-configurable compute platforms like FPGAs acting as hardware accelerators for VNFs are gaining special attention. FPGAs offer the best of both worlds, i.e., the flexibility of general-purpose processors and the performance of dedicated hardware boxes. Therefore, compute-intensive portions of a network function running on the CPU could be offloaded to the re-configurable accelerators running on an FPGA. In some COTS servers, a CPU can be integrated with programmable logic on the same die or it can be attached to an FPGA board via a PCIe bus.
The flexible and scalable hardware-acceleration of multiple VNFs in an NFV environment is still a challenge. Moreover, as hardware accelerator resources are limited as compared to other computing resources, an allocation strategy is required for their efficient provisioning in NFV environments. Different components of the NFV reference architecture, proposed by the ETSI, are depicted in Fig.A.1. The key components which are relevant to our scheme

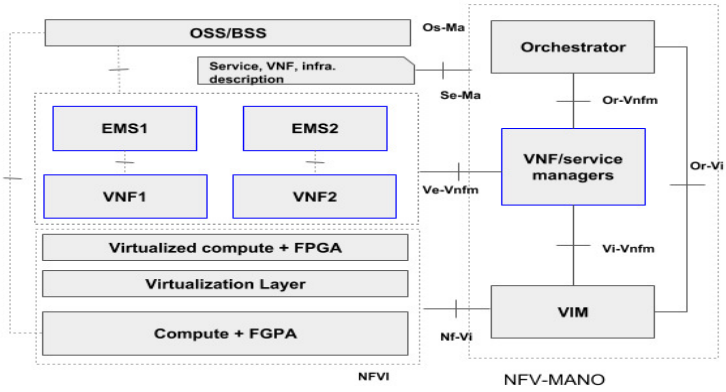


Figure A.1: ETSI's reference architecture for NFV.

are highlighted in blue. The VNF manager (VNFM) or Service-specific Manager (SSM) is responsible for life-cycle management, i.e. starting/stopping, scaling and configuring, of one or more VNFs during the service lifetime. The element management system (EMS) for each VNF and the SSM coordinate with each other to manage the service-specific parameters of VNFs during their life-cycle. We have implemented an SSM that feeds on the resource utilization information of VNFs and determines which VNF to allocate the available accelerator resources. We have chosen the SSH tunneling service to demonstrate a scheme for the dynamic provisioning of hardware accelerators to VNFs. Based on the real-time resource usage, we show how accelerator resources could be dynamically activated for different SSH tunnels.

A.2 System Architecture and Implementation

Oftentimes, an employee of an enterprise sitting in a home needs a secure means to access network services present in a private network of its office or data center. SSH tunneling is the most straightforward method for creating a secure channel between a home user and a server present in the private network. An SSH client running on the user's machine creates an encrypted tunnel to the SSH server of the enterprise passing through the internet. After the creation of the SSH tunnel, the SSH client forwards the packets which it received at the listening port to a specific mapped port on the destination host (local port forwarding) via the SSH server of the private network.

We have implemented and evaluated our scheme for the service based on SSH client VNFs. However, system components and processes remain the same for other scenarios with different VNF types. Next, we will discuss

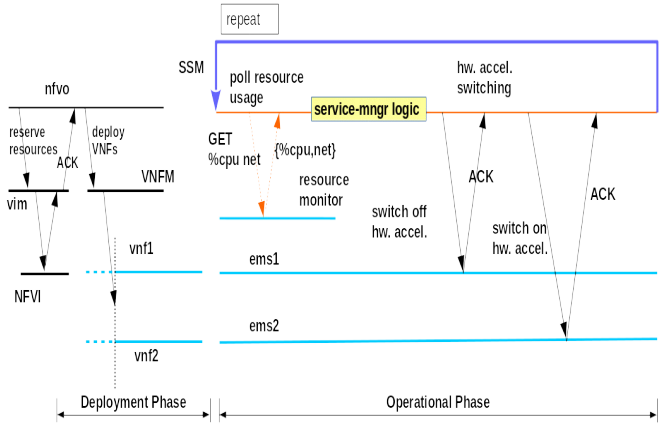


Figure A.2: Components and NFV processes involved in the accelerator allocation scheme.

the mechanism of the scheme which involves SSM and other components of ETSI's NFV architecture.

A.2.1 Service Specific Manager and NFV processes

Fig. A.2 illustrates the components and processes required for the deployment and management of hardware-accelerated VNFs in alignment with the ETSI's NFV architecture. Firstly, the NFV orchestrator (NFVO) delegates the task of reserving resources for VNFs allocation to the virtual infrastructure manager (VIM). The NFVO also specifies the need for any hardware accelerator cores, e.g. AES and SHA for SSH client VNFs. A separate instance (VNF) is instantiated in order to monitor the resource utilization information, e.g. %CPU, network traffic-rate, of the service VNFs.
During the operational phase, the SSM fetches the resource utilization of each VNF using the monitoring system and feeds it to the allocation logic. The allocation logic then selects the most suitable VNF to which the accelerator should be allocated. In our case, SSM's role is to dynamically provision accelerator resources to VNFs based on the allocation logic. The algorithm for VNF selection is discussed in the next subsection.
After determining the next VNF to accelerate, the currently accelerated VNF is triggered to release the accelerator resource. This is followed by granting the accelerator to the selected VNF by triggering its EMS as shown in Fig A.2. The required configuration of a VNF for accelerator de/allocation can be done using the interface between the EMSs and the SSM.

A.2.2 Accelerator allocation algorithm in SSM

The allocation algorithm consists of an infinite while-loop as shown in Alg. A.1. At the start of the algorithm ($Timer = -1$), a set of VNFs having resource usage higher than the high-threshold (Th_{high}) value is selected from all running VNFs (Filter($(usage > Th_{high}), vnfs$)). The resource usage can be %CPU usage of the VNF or the rate of traffic flowing through a network interface of the VNF. Out of the selected set of VNFs, a VNF with minimum access (`LowestTimer`) is chosen for the acceleration. Therefore, a VNF which has had the least access to accelerator resources in the past is picked.

After the choice of VNF for the access to the accelerator has been made, the currently accelerated VNF is requested to release accelerator resources. Next, the newly selected VNF (vnf) is granted access to the accelerator along with resetting the $Timer$ variable to zero. Access to accelerator resources is given for a fixed time-period (T). Both the timers– $Timer$ and $vnf_{acc}.timer$ are incremented with each pass of the while-loop. Upon the expiry of the allocated time ($Timer > T$), the process of selection, deallocation, and allocation is repeated.

Meanwhile, if the resource usage of the accelerated VNF falls below the lower threshold value, the accelerator resources are taken from that VNF. High-threshold (Th_{high}) and low-threshold (Th_{low}) values for the CPU usage are $0.75(CPU_{max})$, and $0.65(CPU_{max})$, respectively, where CPU_{max} is the maximum CPU utilization of a VNF. The minimum accelerator allocation time (T) in our setup is 11s. This choice of threshold levels and allocation time leads to a responsive and stable operation of the accelerator allocation algorithm.

A.2.3 AES and SHA Acceleration in Dropbear

For the SSH client we have chosen Dropbear[1]. The original Dropbear implementation utilizes ciphers and hashes functions provided by libtomcrypt (cryptographic library)[2], to perform en/decryption and hashing on data packets. These functions involve multiple rounds of bit- and byte- level manipulations, e.g. XOR, substitutions, rotations, of data. A software cipher or hash function would require a CPU to execute these operations sequentially on the input data. On the other hand, a hardware implementation could perform these functions much faster owing to the massive parallelism available on an FPGA or ASIC.

In order to accelerate these cryptographic operations in Dropbear, the

[1] https://github.com/mkj/dropbear
[2] https://github.com/libtom/libtomcrypt

Algorithm A.1: Pseudo-code for the accelerator allocation algorithm.

```
1  Global: vnf_acc, Timer
2  Input: T, Th_low, Th_high
3  Procedure SWAP(vnf):
4      if vnf_acc ≠ φ then
5          deallocate(vnf_acc)
6      if vnf ≠ φ then
7          vnf_acc ← vnf
8          allocate(vnf_a)
9      Timer ← 0
10 Procedure INCTIMERS():
11     Timer ← Timer + 1
12     if vnf_acc == φ then
13         vnf_acc.timer ← vnf_acc.timer + 1
14 Timer, vnf_acc ← −1, φ
15 while True do
16     if Timer > T or Timer == −1 then
17         vnf ← LowestTimer(FILTER(usage > Th_high), vnfs)
18         SWAP(vnf)
19     else if vnf_acc.usage < Th_low then
20         SWAP(φ)
21     else
22         INCTIMERS()
```

FPGA-based accelerators cores for AES-128/256 and SHA-256 are utilized. These cores are based on an open-source Verilog implementation [34]. The hardware architecture for accelerating en/decryption and hash using external hardware cores is shown in Fig. A.3. For en/decryption, the hardware core is first initialized by writing keys and initialization vectors into its memory-mapped registers of the AES core. Similarly, for hash initialization, the current hash state is written to SHA-256 core's registers. Initialization is followed by the transfer of input data from the RAM memory to the BRAM of the accelerator core where cipher-text or hash is calculated. The AES/SHA core engine fetches input text from the BRAM one block (BS_{AES} = 4x32-bit words, BS_{SHA} = 16x32-bit words) at a time, processes it and then writes the processed text back to BRAM. The progress of the core is monitored continuously by checking the "progress pointer" of the corresponding core, which indicates the length of the input text which has been processed.

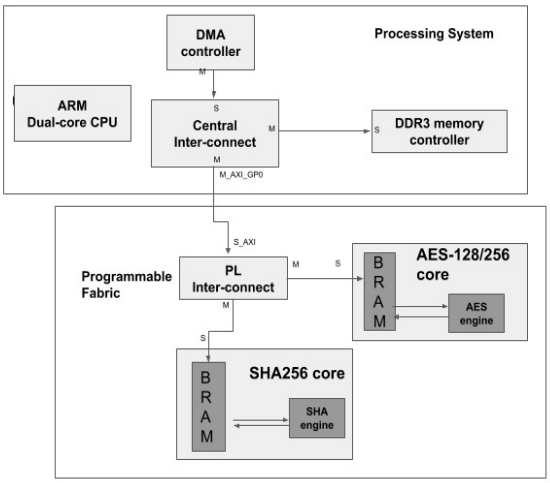


Figure A.3: Hardware design for AES en/decryption and SHA hash acceleration on PYNQ.

When the processing of the input text is complete, cipher-text or hash is transferred back into the main memory (DRAM). All data transfer tasks between the main-memory and accelerator core's BRAM are managed by the direct memory access (DMA) controller (DMAC) present on the ZYNQ processing system (PS) [2]. A kernel module[5] is used to manage DMA transfers from the user-space buffers in Dropbear to the respective cores using

[3]https://github.com/secworks/aes
[4]https://github.com/secworks/sha256
[5]https://github.com/jeremytrimble/ezdma

zero-copy mechanisms. This module creates the scatter-gather list of the memory pages corresponding to the user-space buffer and passes this list to the pl330 driver which configures the DMA controller. Next, the DMAC performs the data transfer between the main memory and the BRAM without the involvement of the CPU. The hardware design for the AES-128 and SHA-256 accelerators was developed and implemented in the Vivado environment [3].
In order to let Dropbear offload functions to accelerator cores, several modifications were added to it. The initialization and the configuration of accelerator cores require mapping of core's address space into Dropbear's address space which is done via `mmap` system call. The transfer of input-text and the processed-text is done by performing `read` and `write` system calls to the char device exposed by the kernel module mentioned earlier. A signal handler to catch `SIGSUR1` is also included in the SSH client. Upon the arrival of `SIGSUR1` from the kernel, Dropbear switches between the two modes, i.e., non-accelerated (software only) and hardware-accelerated.

A.2.4 Complete System and Implementation

The complete system was implemented using a laptop and a PYNQ board. The PYNQ board has a dual-core ARM-A9 processor (PS) and a programmable fabric (PL) on the same ZYNQ chip[6]. The PS part of the ZYNQ chip is used to run Ubuntu-16.04 OS. The PYNQ board is attached to a laptop running Ubuntu 17.04 via an Ethernet cable. To keep our implementation simple, the orchestrator (python script) is just responsible for deploying VNFs using docker-tools (docker-client and docker-daemon) and initiating the SSM. Network functions used for establishing SSH tunnels are the modified Dropbear's SSH clients. The complete implementation is shown in Fig. A.4.
Docker-based VNFs are deployed on the PYNQ board by requesting the docker-engine running on Ubuntu-16.04 of the PYNQ board. A docker-client running on the laptop requests the docker-daemon for VNF instantiation. Each VNF for SSH tunneling is a docker container that runs two applications:

1. Dropbear SSH client (`dbclient`).
2. REST-ful server acting as an element manager (`em`) for the corresponding VNF.

`dbclient` is responsible for creating an encrypted channel between the PYNQ board and laptop, and setting up the required port forwarding be-

[6] http://www.pynq.io/board.html

tween a user on the PYNQ board to the dst-server on the laptop. The user and dst-server applications are the `iperf` client and server applications, respectively.
Upon receiving a trigger from SSM, `em` sends `SIGUSR1` signal to `dbclient` process using the `kill` systen call. The signal handler in `dbclient` catches `SIGUSR1` signal and switches between the accelerated and non-accelerated modes of Dropbear.
The SSM logic requires the resource usage information for all VNFs. To this end, we deploy a container-based monitoring system called Cadvisor on PYNQ. Cadvisor collects, aggregates and then exports the resource monitoring information of all the running containers to a specific port. The SSM scrapes this information from Cadvisor periodically and feeds it to the SSM allocation algorithm. Upon processing this information, the SSM sends `GET` requests to the `em` of appropriate VNFs in order to grant or release accelerator resources.

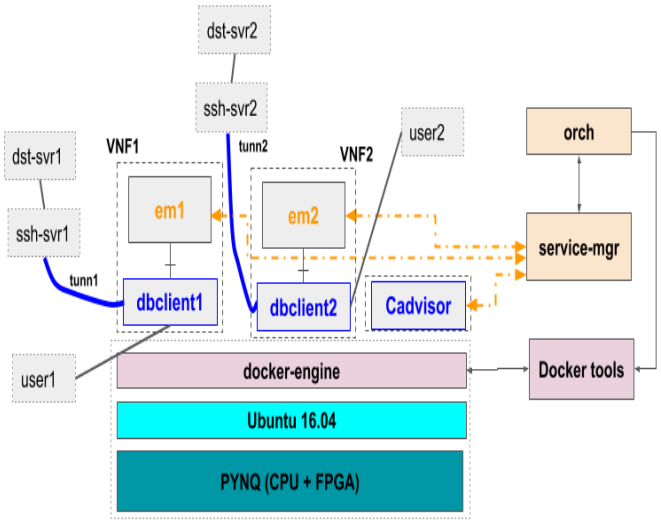


Figure A.4: System implementation for allocation of AES and SHA accelerator on PYNQ board.

A.3 Related works

The use of reconfigurable hardware accelerators has been explored for a long time for several packet-processing applications [4] [5]. In an NFV environment, it is essential to accelerate the performance of specific VNFs

running by the means of external hardware accelerators. In [6], a framework has been proposed to utilize FPGAs in order to implement complete network functions in hardware. This framework allows to build VNFs based on FPGA only and does not allow a VNF running on a CPU to offload selected tasks to FPGA accelerators. Sharing of FPGA fabric among multiple VNFs is accomplished by partial re-configuration technology. FPGAs provide high-level of programmability as compared to ASICs but they are still expensive and less programmable as compared to COTS servers.
Li et al. have developed a dynamic hardware library (DHL), a library to abstract FPGA-based accelerators for VNFs which can be accessed using a set of APIs [7]. This work focused on easing the amount of effort to access accelerators from VNFs.
Byma, et al. have proposed a framework that aggregates partial reconfigurable regions across multiple FPGAs to offer a single FPGA resource to a cloud tenant who can program its allocated region. This framework is useful for cloud service providers who would like to offer FPGAs just like other compute resources [8].
OpenANFV is another such framework that works with OpenStack to manage and virtualize accelerator resources for VNFs requiring high-performance packet processing requirements [9]. Nobach et al. also proposed an architecture for elastic provisioning of accelerators to VNFs [10]. A VNF can offload selected workloads to accelerator hardware modules on-demand basis. However, this work does not discuss any implementation which can be integrated within an NFV environment.
As accelerator resources are limited as compared to general-purpose compute resources, there is a need to efficiently allocate them among different network functions. Therefore, the challenge to dynamically allocate accelerator resources to VNFs needs to be addressed.

A.4 Evaluation and Results

We evaluate our implementation in two parts. In the first part, we compare the performance of non-accelerated and accelerated VNFs in terms of their peak throughput and %CPU usage. In the second part, we verify the allocation algorithm by testing multiple VNFs with varying traffic patterns.

A.4.1 Comparison of original and hardware-accelerated VNF

Table A.1 shows the peak throughput values of software-only and hardware-accelerated SSH client VNFs. We have done this comparison using two types

of ciphers– AES128 and AES256. The mac-algorithm used for hashing data packets in our evaluations is SHA-256.

Table A.1: Comparison of software and hardware ciphers and hashes.

Algorithm	Peak throughput (Mbps)	%CPU
AES128-SHA256-sw	38.8	99
AES128-SHA256-hw	55.1	84.5
AES256-SHA256-sw	31.5	99
AES256-SHA256-hw	51	86

As AES256 involves more number of rounds than AES128, the peak throughput of the SSH tunnel with AES256 cipher is less than the tunnel using AES128 cipher. Moreover, one can notice a 42.1% improvement in the throughput for hardware-accelerated AES128-SHA256 and 61.9% for AES256-SHA256 over non-accelerated SSH clients. In addition to the improvement in the peak throughput, one can clearly see a reduction in %CPU usage when the VNF is accelerated. This results from the fact that the CPU is relieved from performing crypto-operations which are now offloaded to accelerator cores.
The variation of %CPU utilization of the SSH client VNF with the changing data traffic on the tunnel is shown in Fig. A.5. As the traffic on the tunnel increases, the %CPU usage also increases. The increase in %CPU usage is because with the increase in the packet arrival rate the CPU spends increasingly more time in performing cipher and hash operations.

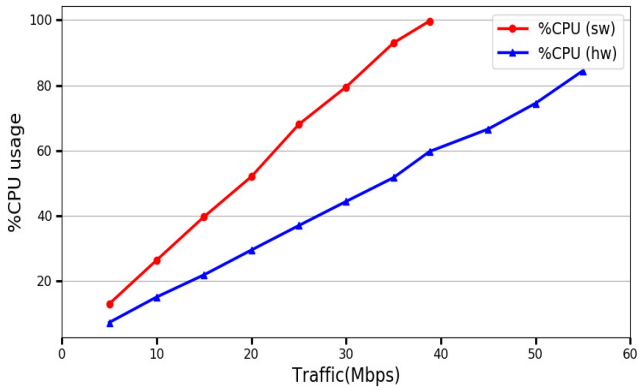


Figure A.5: Variation of %CPU usage of SSH clients VNFS with changing traffic for non-accelerated (sw) and accelerated modes (hw).

A.4.2 Dynamic accelerator allocation

The setup shown in Fig. A.4 is used to verify the operation of the accelerator allocation algorithm. We evaluate our implementation by instantiating `dbclient` VNFs and then loading them with time-varying traffic patterns. VNFs were pre-configured to use AES-128 cipher and SHA-256 hash while establishing SSH sessions with the SSH server. As a result of SSH port forwarding, the configured ports on the PYNQ board are forwarded to the laptop where the destination server is running. The home-user and destination server are the `iperf` client and server applications, respectively.
We start the experiment first by deploying two VNFs on the PYNQ board and then loading them with data traffic from two users (`iperf` clients). Fig. A.6(b) shows the time variation of the peak-throughput of two SSH tunnels. As the CPU usage of the first tunnel crosses the upper threshold (0.70) value at t=12s (Fig. A.6(a)), the SSM triggers *em1* to let VNF1 access the accelerator resources. With access to the accelerator, the throughput of tunn1 is improved and its CPU usage is also reduced (Fig. A.6(a)).
After a few seconds, we load the second tunnel (tunn2) with the traffic as well. At this point in time, the CPU of both tunnels is higher than the upper threshold value. However, the cumulative accelerator time for VNF2 is less than that of VNF1, because VNF1 was granted the accelerator in the previous time period. Therefore, the SSM requests VNF1 to release the accelerator and which is then granted to VNF2 for a time period of about 11s, resulting in a switch-over shown at t=22s. Thereafter, the traffic remains high on both tunnels such that their CPU usages are above the threshold values, the SSM grants accelerator resources to two VNFs according to their cumulative accelerator time resulting in a round-robin allocation. At accelerator switch-overs (t = 22s, 35s, 45s ..), accelerator access time for a VNF is finished and the accelerator is allocated to the other VNF.
We repeat the above experiment for two more VNFs, such that there are four concurrent SSH tunnels established between the PYNQ board and the laptop. Each VNF corresponding to these tunnels has a CPU-share of 0.4. All the four tunnels (tunn1-tunn4) are loaded with the data traffic and the resulting throughput and CPU usage are observed. The average CPU usage of all tunnels remains around 0.4 but a drop can be noticed during the time intervals of hardware acceleration. In the Fig. A.6 (d), traffic variation for four tunnels is shown. The traffic rate and the corresponding CPU usage of the tunn2 are highlighted in red. It can be noticed that the CPU usage of VNF2 is the lowest when it has the highest throughput during the following time intervals – t=18-30, 55-67, 140-116. As the allocated accelerator time for one VNF is completed, a new VNF with lowest cumulative accelerator time is allocated the accelerator. The accelerator allocation period

(T=11s) for four tunnel and their switch-overs from one VNF to another can be observed from Fig. A.6(d).

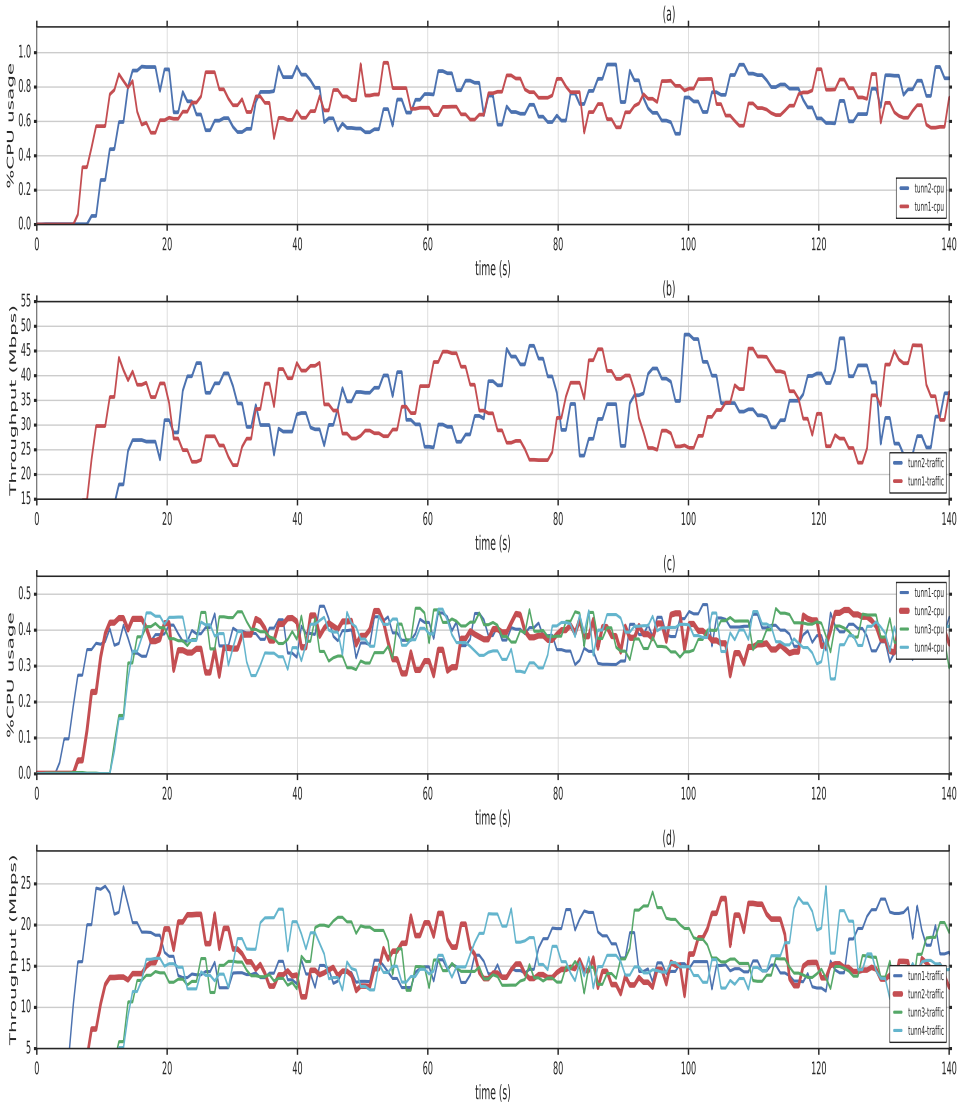


Figure A.6: Variation of (a), (c): CPU usage and (b), (d): traffic rate with time corresponding to two and four concurrent SSH tunnels.

A.5 Conclusion

Hardware accelerators are increasingly becoming a part of NFV infrastructure to accelerate packet processing performance of VNFs such that SLAs are met. A mechanism is required for the flexible and dynamic provisioning of accelerator resources in an NFV scenario. The SSM is a NFV-MANO component that is responsible for VNF management aspects including accelerator de-allocation and allocation tasks. We made the following observation from the evaluation of our implementation of the accelerator allocation scheme:

1. VNF's throughput improvement and a reduction in over- all %CPU usage is achieved by offloading AES and SHA operations to hardware accelerator cores.

2. A dynamic provisioning of accelerator resources among multiple VNFs can be achieved based on the real-time resource usage and cumulative allocation times of VNFs.

3. The proposed accelerator allocation scheme complies with the ETSI's reference architecture for NFV.

This work can be extended by comparing the accelerator allocation algorithm based on the performance profile of VNFs with the current reactive approach.

A.6 Acknowledgments

This work was funded through NGPaaS (761557) and 5GTANGO (761493), in the scope of the EC's Horizon 2020 and 5G-PPP programs.

References

[1] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba. *Network function virtualization: State-of-the-art and research challenges.* IEEE Communications Surveys & Tutorials, 18(1):236–262, 2016.

[2] Xilinx. *Zynq-7000 All Programmable SoC Technical Reference Manual*, v1.10 edition, 2015.

[3] Xilinx. *Vivado Design Suite User Guide*, v2014.1 edition, April 2014.

[4] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood. *Deep packet inspection using parallel bloom filters.* In 11th Symposium on High Performance Interconnects, 2003. Proceedings., pages 44–51. IEEE, August 2003.

[5] A. Wicaksana and A. Sasongko. *Fast and reconfigurable packet classification engine in FPGA-based firewall.* In Proceedings of the 2011 International Conference on Electrical Engineering and Informatics, pages 1–6, July 2011. doi:10.1109/ICEEI.2011.6021782.

[6] C. Kachris, G. C. Sirakoulis, and D. Soudris. *Network Function Virtualization based on FPGAs: A Framework for all-Programmable network devices.* CoRR, abs/1406.0309, August 2014. arXiv:1406.0309.

[7] X. Li, X. Wang, F. Liu, and H. Xu. *DHL: Enabling Flexible Software Network Functions with FPGA Acceleration.* In 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pages 1–11. IEEE, July 2018.

[8] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow. *FPGAs in the cloud: Booting virtualized hardware accelerators with openstack.* In 2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines, pages 109–116. IEEE, May 2014.

[9] X. Ge, Y. Liu, D. H. Du, L. Zhang, H. Guan, J. Chen, Y. Zhao, and X. Hu. *OpenANFV: Accelerating network function virtualization with a consolidated framework in openstack.* In ACM SIGCOMM Computer Communication Review, volume 44, pages 353–354. ACM, 2014.

[10] L. Nobach and D. Hausheer. *Open, elastic provisioning of hardware acceleration in nfv environments.* In 2015 International Conference and Workshops on Networked Systems (NetSys), pages 1–5. IEEE, March 2015.

B

Hardware accelerator aware VNF-chain recovery

While Chapter 2 deals with the VNF-AAPC problem, accelerator-awareness is required in VNF-chain recovery algorithms too. Otherwise, a failure in the NFVi will result in an efficient resource allocation.
This appendix addresses the accelerator-aware VNF-chain recovery problem using an ILP approach and a greedy heuristic. The accelerator-aware VNF-chain recovery problem extends the VNF-AAPC problem in Chapter 2 by adding prioritization to the VNF-chain recovery process.

⋆⋆⋆

G.P. Sharma, W. Tavernier, D. Colle, and M. Pickavet

Published in the Proceedings of the International Conference on the Design of Reliable computer networks (DRCN), Milan, Italy (virtual), 2020

Abstract Hardware accelerators in Network Function Virtualization (NFV) environments have aided telecommunications companies (telcos) to reduce their expenditures by offloading compute-intensive VNFs to hardware accelerators. To fully utilize the benefits of hardware accelerators, VNF-chain

recovery models need to be adapted. In this paper, we present an ILP model for optimizing the prioritized recovery of VNF-chains in heterogeneous NFV environments following node failures. We also propose an accelerator-aware heuristic for solving prioritized VNF-chain recovery problems of large size in a reasonable time. Evaluation results show that the performance of heuristic matches with that of ILP in regard to restoration of high and medium priority VNF-chains and a small penalty occurs only for low-priority VNF-chains.

B.1 Introduction

In the past few years, an exponential growth of internet data traffic has taken place due to the explosion in the total number of the connected-users and network services. This change has compelled telecommunications companies (telcos) to look for alternative network-architecture solutions which are more economical, manageable and scalable. Network Function Virtualization (NFV) proposes to transform the manner in which network services are currently created and managed by leveraging Information Technology (IT) virtualization technologies for network services. With NFV, services that were earlier implemented using proprietary hardware appliances (middleboxes) can now be created using virtual network functions (VNFs) or Cloud-native Network Functions (CNFs). As a result, telcos can significantly reduce their capital expenditures (CAPEX) by running VNFs (or CNFs) on commercial-off-the-shelf (COTS) servers (e.g. x86 or ARM) instead of purchasing costly middleboxes.

In the recent past, hardware accelerators like Graphics Processing Units (GPUs), smartNICs and Field Programmable Gate Arrays (FPGAs) are also being included in the NFV infrastructure (NFVi) to address the performance issues with VNFs [1]. Hardware accelerators can offload the compute-intensive tasks from a VNF running on the CPU resulting in a speedup of execution. This packet-processing offload also frees-up CPU cores which can be used to run other VNFs resulting in better server consolidation.

In order to truly exploit the benefits of NFV, it is essential to allocate NFVi resources to VNFs as efficiently as possible. State-of-the-art (SotA) VNF placement models and strategies only consider usual NFVi infrastructure resources like compute, storage and network while making their allocation decisions. This can lead to an inefficient allocation of NFVi resources in heterogeneous NFVi which contain hardware accelerators along with the usual NFVi resources.

The reliability of network services can be enhanced by utilizing hardware accelerators to satisfy the diverse Quality of Service (QoS) requirements of

service chains. Therefore, QoS is not only impacted by the re-allocation of usual NFVi resources (compute, storage and network) after a failure but also by the hardware accelerator re-allocation. In the case of server node failures, the VNF-chain recovery processes should also take into account the presence of hardware accelerator resources in NFVi along with usual NFVi resources. Moreover, VNF remapping and accelerator-allocation should be able to prioritize traffic according to the QoS requirements. To the best of our knowledge, no work has addressed this problem in the past.
To this end, we make the following contributions in this paper:

1. Modeling of the prioritized VNF-chain recovery problem in heterogeneous NFV environments using the Integer Linear Programming (ILP) approach.

2. Designing the greedy-based heuristic algorithm to solve the above problem.

3. Evaluations of the ILP and the heuristic algorithm.

The rest of the paper is organized as follows. The related work for this paper is discussed in the next section. The problem of accelerator-aware VNF-chain recovery problem is formulated using an ILP model in section B.3. Afterward, a heuristic to solve VNF-chain recovery problem is presented in section B.4. Section B.5 describes the evaluation of the ILP and the heuristic. Finally, the conclusion of the paper is presented in section B.6.

B.2 Related Works

With the growing interest in NFV among industry and academia, various studies have been carried out related to the modeling of resource allocation in NFV. The authors in [2] defined the VNF placement and routing optimization problem and devised a mixed-ILP (MILP) formulation for it. Bari et al. also modeled the VNF Orchestration Problem (VNF-OP) using ILP approach [3]. They proposed a dynamic programming approach to solve VNF-OP instances with larger sizes. The authors in [4] have proposed a Uniform Hardware Acceleration Deployment architecture (UHAD) for simplifying the integration of hardware accelerators in NFVi.
In [5], the authors studied the problem of allocating backup resources for VNFs and virtual-links such that reliability constraints of service-chains are adhered. A scalable heuristic algorithm, aimed at sharing backup resources in order to reduce overhead costs due to backups, was also proposed.

A. Tomassilli et al. investigated two different (dedicated and shared) protection mechanisms for reliable chaining of VNFs in case of a single-link failure [6]. The evaluation of the ILP models shows that the dedicated protection scheme requires 40% more bandwidth and 60% more processing resources as compared to the shared protection scheme. A study regarding service reliability using VNF migration and replications was conducted in [7]. The authors recommended jointly performing VNF migration and replication in order to efficiently utilize server and link resources. An algorithm is also proposed for optimizing the provisioning of hardware accelerators in NFVi nodes. We also proposed a scheme for the dynamic allocation of hardware accelerators to VNFs in NFV environments by the use of specific service managers (SSMs) [8].
Although numerous resource allocation models have been proposed aiming at optimizing various parameters like cost, performance, load-balancing, etc, there has been little focus on prioritized VNF-chain placement models for heterogeneous NFV environments. In [9], the authors addressed the problem of joint scaling, placement and routing for heterogeneous services. They presented an MILP approach for single-step exact solutions along with a heuristic algorithm for fast and near-optimum solutions. We formulated the accelerator-aware VNF placement problem in the form of an ILP and also proposed a scalable algorithm based on best-fit heuristic [10].
In summary, the challenge of prioritized VNF-chain recovery in a heterogeneous environment in case of server node failures still remains unaddressed.

B.3 Problem Formulation

The description of various parameters and decision variables involved in the ILP formulation is given in Table B.1.
The objective of this ILP formulation is to maximize the total amount of traffic restored after a failure subject to the resource capacity. In addition to that, restoration is prioritized based on the traffic class of VNF-chains. In the objective function (B.1), x_s is the decision variable indicating if the VNF-chain s is restored after the failure. The product $\mu_s x_s$ denotes the amount of traffic restored after the recovery of VNF-chain s. The scaling factor μ_s in the objective function is the cost-gain for restoring a service-chain $s \in S$. The prioritization of VNF-chains is performed by calculating μ_s of VNF-chains based on their respective traffic class as explained below. Depending on the traffic class of VNF-chains, all VNF-chains are categorized into three sets, namely– high-priority ($S_I \subseteq S$), medium-priority ($S_{II} \subseteq S$) and the low-priority ($S_{III} \subseteq S$) VNF-chains. After any node-failure in NFVi, the preference for restoration is given to VNF-chains $s \in S$ in the

following order: first for $s \in S_I$, then for $s \in S_{II}$, and at last for $s \in S_{III}$. The requirement to prioritize VNF-chain restoration can be implemented by adding soft constraints to the ILP formulation. This is done through the assignment of restoration gain μ_s for each VNF-chain s as indicated in (B.2). Here T_{II} and T_{III} can be evaluated using (B.3). By scaling the throughput values of traffic ($x_s t_s$) for each VNF-chain request with a restoration-gain value μ_s, we have avoided the requirement for adding hard constraints with regards to prioritized restoration of traffic.

$$obj : \max \sum_{s \in S} t_s \mu_s x_s \tag{B.1}$$

$$\begin{gathered} \mu_{s_{III}} = 1, \quad \mu_{s_{II}} t_{s_{II}} = T_{III}, \quad \mu_{s_I} t_{s_I} = T_{II} + T_{III} \\ \forall s_{III} \in S_{III}, \quad s_{II} \in S_{II}, \quad s_I \in S_I \end{gathered} \tag{B.2}$$

$$T_{III} = \sum_{s \in S_{III}} \mu_s t_s, \quad T_{II} = \sum_{s \in S_{II}} \mu_s t_s \tag{B.3}$$

B.3.1 Node constraints

For each server node $n \in N$, the total number of CPU cores utilized by all VNFs placed on n is constrained by the number of cores available on it as indicated in (B.4). The constraint on resources available on the hardware accelerator fabric for the instantiation of accelerators is shown in (B.5). Constraint in (B.6) ensures that the PCIe bandwidth required for communication between VNFs and accelerators does not exceed the available bandwidth.

$$\sum_{s \in S, f^s \in F^s} \alpha^n_{f^s} cpu_0(f^s) - \beta^n_{f^s} cpu_r(f^s) \leq \mathcal{R}_{cpu}(n) \quad \forall n \in N \tag{B.4}$$

$$\sum_{a \in A} r(a) \delta^n_a \leq \mathcal{R}_{acc}(n) \quad \forall n \in N \tag{B.5}$$

$$\sum_{s \in S, f^s \in F^s} 2 t_s \beta^n_{f^s} \leq \mathcal{R}_{bus}(n) \quad \forall n \in N \tag{B.6}$$

B.3.2 Acceleration constraints

The constraint in (B.7) is a consequence of the fact that a VNF f^s can only be allocated an accelerator on n if the f^s is placed on n. Also, binary decision variable δ^n_a is assigned a value of 1 if atleast one VNF is allocated accelerator a on n as indicated in (B.8). This constraint can be linearized by

Table B.1: Description of parameters and decision variables

Input parameters	
Notation	Description
N	Set of all non-failed computational nodes within NFVi.
$\mathcal{R}_{cpu}(n)$	Maximum CPU resources (cores) available on $n \in N$.
$\mathcal{R}_{acc}(n)$	Maximum accelerator fabric resources (logic elements) available on $n \in N$.
$\mathcal{R}_{bus}(n)$	Maximum bandwidth (Gbps) of the PCIe bus of node $n \in N$.
A	Set of all available accelerator types.
$r(a)$	Resource requirement (logic elements) of the accelerator type $a \in A$.
S	Set of all service requests, including top (S_I), medium (S_{II}) and low (S_{III}) priority services.
F^s	Set of all VNFs corresponding to the service-chain request $s \in S$.
t_s	Throughput requirement (Gbps) of the service request $s \in S$.
$cpu_0(f^s)$	CPU requirement (cores) of VNF f of the service request $s \in S$.
$cpu_r(f^s)$	CPU reduction (cores) for VNF f of the service request $s \in S$.
$atype(f^s)$	Type of accelerator needed for acceleration of VNF f of the service request $s \in S$.
Decision variables	
Notation	Description
x_s	Binary decision variable indicates if the service $s \in S$ is restored after the failure.
$\alpha^n_{f^s}$	Binary variable indicates if VNF f of service request s is placed on n after the failure.
$\beta^n_{f^s}$	Binary variable indicates if VNF f of service request s is accelerated on n after the failure.
δ^n_a	Binary variable indicates if accelerator of type a is instantiated on the node n after the failure.

the re-formulation shown in (B.9a-B.9b). If no VNF is allocated accelerator a on n, (B.9a) forces δ_a^n to be equal to zero. Conversely, if atleast one VNF is assigned accelerator a on n, LHS of (B.9b) is ≥ 1 resulting in δ_a^n to take a value 1. The factor M_1 in the RHS of (B.9b) is a constant greater than the total number of VNFs in all VNF-chains, i.e., $M_1 = \sum_{\substack{\forall s \in S \\ \forall f^s \in F^s}} 1$.

$$\beta_{f^s}^n \leq \alpha_{f^s}^n \quad \forall n \in N, \forall s \in S, \forall f^s \in F^s \tag{B.7}$$

$$\delta_a^n = \begin{cases} 1, & \text{if } \sum_{\substack{\forall s \in S, \forall f^s \in F^s, \\ a = atype(f^s)}} \beta_{f^s}^n \geq 1 \\ 0, & \text{otherwise} \end{cases} \qquad \forall n \in N, \forall a \in A \tag{B.8}$$

$$\delta_a^n \leq \sum_{\substack{\forall s \in S, \forall f^s \in F^s \\ a = atype(f^s)}} \beta_{f^s}^n \quad \forall a \in A, \ \forall n \in N \tag{B.9a}$$

$$\sum_{\substack{\forall s \in S, \forall f^s \in F^s \\ a = atype(f^s)}} \beta_{f^s}^n \leq M_1 \delta_a^n \quad \forall a \in A, \ \forall n \in N \tag{B.9b}$$

B.3.3 Other Constraints

A VNF-chain is said to be restored if all the VNFs constituting the VNF-chain are placed on computational nodes. This requirement is expressed by a set of constraints given in (B.10). This constraint can be linearized using the method discussed before for linearizing the constraint in (B.8). Constraints in (B.11) force binary variables $x_s, \alpha_{f^s}^n, \beta_{f^s}^n, \delta_a^n$ to only take binary values (0 or 1).

$$x_s = \begin{cases} 1, & \text{if } \sum_{\forall n \in N} \alpha_{f^s}^n = 1, \quad \forall f^s \in \mathcal{F}^c \\ 0, & \text{otherwise} \end{cases} \qquad \forall s \in S \tag{B.10}$$

$$x_s, \alpha_{f^s}^n, \beta_{f^s}^n, \delta_a^n \in \{0, 1\} \quad \forall n \in N, \forall s \in S, \forall f^s \in F^s \tag{B.11}$$

B.4 Proposed Algorithm

The algorithm we propose to solve the prioritized VNF-chain recovery problem is based on the greedy heuristic. The pseudo-code for the algorithm is presented in Alg. B.1. The algorithm takes as an input the following: a list of all VNF-chains S, a list of non-broken chains $S' = \{s \in S : x_s = 1\}$), VNF assignments ($\alpha$) before failure, accelerator allocations (β) before failure and a set of all non-failed server nodes (N) with their respective resource

usages.
A sorted list S_b of all broken chains in the order of their decreasing priorities, which is based on their μ_s values is created. Also, a list of all chains S_{rev} in increasing priority is created.
For accelerator-agnostic VNF-chain recovery, placement decisions are agnostic to the availability of accelerators. As a result, placement decisions are based solely on CPU resources and accelerator-allocation is done only if hardware accelerator resources are available on that node. In other words, no explicit effort is made by the heuristic to allocate an accelerator to a VNF resulting in an inefficient resource utilization. The algorithm for accelerator-agnostic VNF-chain recovery does not contain lines (9-15) of Alg. B.1.
With accelerator-aware allocation, the decision for accelerator allocation is decoupled from other placement decisions. As opposed to the accelerator-agnostic placement, accelerator-aware placement makes use of `accelVNF` procedure for the explicit allocation of an accelerator to a VNF. The pseudo-code for `accelVNF` procedure is shown in Alg. B.2. First, it is checked if enough resources are available on any server node with an attached hardware accelerator by using `AccAlloc` procedure. If not, a node with a hardware accelerator is randomly selected (n_a) from a list (N_a) of server nodes with hardware accelerator sorted in the order of increasing available CPU. To accommodate f^s on n_a, CPU, bus or accelerator resources are made available by removing VNFs of the lowest priority VNF-chain if the priority of VNF-chain s is more than s_b. A VNF-chain is selected in-order from S_b for its restoration and the set of non-placed VNFs is assigned to F^b. For every VNF f^b which has an accelerator implementation available in A, its placement is first tried on a server node with hardware accelerator available using the procedure `accelVNF` (Alg. B.2). If procedure `accelVNF` returns $None$, its placement is continued as usual.
If not enough CPU resources are available on any server node, VNF-chains are removed sequentially in the greedy manner i.e. lower-priority chain first from S_{rev}. The removed VNF-chain s_r is added to the sorted-list of the broken chain S_b and resources are updated using the `RemoveVNFs` procedure. In the case when all VNFs of s_b are placed successfully, S' is updated and the next VNF-chain with the lower priority is considered for recovery. At the end of the algorithm, S' contains all the VNF-chains which have all their VNFs restored.

Algorithm B.1: VNF-chain recovery algorithm

Input : S, S', α, β, N
Output: S'

$S_b \leftarrow$ sorted list of broken chains in decreasing priorities;
$S_{rev} \leftarrow$ sorted list of chains in increasing priorities;
$i_b, i_r \leftarrow 0$;
$l \leftarrow 0$;
while $i_b < |S_b|$ **do**
  $s_b \leftarrow S_b[i_b]$;
  $F_b \leftarrow \{f^s \in F^{s_b} :$ VNFs in F^{s_b} which are not placed$\}$;
  $fail \leftarrow$ **False**;
  for f^b **in** F_b **do**
    if `atype(`f^b`)`$\in A$ **then**
      $n_a \leftarrow$ `accelVNF(`f^b, N`)`;
      if $n_a \neq$ **None then**
        $\alpha[f^b], \beta[f^b] \leftarrow n_a, n_a$;
        break;
    $n_p \leftarrow \arg\max_{n \in N} \mathcal{R}_{cpu}(n)$;
    while $cpu_0(f^b) > \mathcal{R}_{cpu}(n)$ **and** $i_r \leq |S_{rev}|$ **do**
      $s_r \leftarrow S_{rev}[i_r]$;
      if `ChainPrior(`s_b`)>ChainPrior(`s_r`)` **then**
        `RemoveVNFs(`F^b`)`;
        $n_p \leftarrow \arg\max_{n \in N} \mathcal{R}_{cpu}(n)$;
        $i_r \leftarrow i_r + 1$;
      else
        break;
    if $cpu_0(f^b) > \mathcal{R}_{cpu}(n_p)$ **then**
      $\alpha[f^b] \leftarrow n_p$;
    else
      $fail \leftarrow$ **True**;
      break
  if $fail ==$ **True then**
    `RemoveVNFs(`F^b`)`;
  else
    $S' \leftarrow S' \cup s_b$;
  $i_b \leftarrow i_b + 1$;
end

B.5 Evaluation

The ILP model for accelerator-aware VNF-chain recovery problem is implemented in the CPLEX (v12.9) framework using the doCPLEX Python

Algorithm B.2: VNF accelerator allocation procedure

```
1  Procedure accelVNF(f^s, N):
2      N_a ← sorted list of nodes N with accelerator in increasing CPU;
3      for n_a in N_a do
4          if AccAlloc(f^s, n_a)== True then
5              return n_a;
6      n_a ← RandomChoice(N_a);
7      if cpu_0(f^s) − cpu_r(f^s) > R_cpu(n_a) then
8          s_b ← lowest priority chain with atleast one VNF placed on n_a.;
9          if ChainPrior(s)>ChainPrior(s_b) then
10             RemoveVNFs({f^{s_b} ∈ F^{s_b} : α[f^{s_b}] = n_a});
11     if 2t_s > R_cpu(n_a) or r(atype(f^s)) > R_acc(n_a) then
12         s_b ← lowest priority chain with atleast one VNF allocated
             accelerator on n_a.;
13         if ChainPrior(s)>ChainPrior(s_b) then
14             RemoveVNFs({f^{s_b} ∈ F^{s_b} : α[f^{s_b}] = n_a});
15     if AccAlloc(f^s, n_a)== True then
16         return n_a;
17     else
18         return None;
19 end
```

API [11] and the heuristic algorithm proposed in Alg. B.1 is implemented in Python. In this section, we describe the evaluations carried out in order to assess the efficiency of the ILP and the heuristic with regard to the VNF-chain recovery problem. First, the placement and accelerator allocation of VNFs is performed by using the heuristic which we proposed in [10]. The result of this heuristic is used as an input allocation for both the ILP and heuristic. A fixed number of server nodes are chosen at random from all the server nodes to cause the failure. In order to highlight the impact of accelerator-allocation criterion on the traffic restoration, we compared the performance of accelerator-agnostic and accelerator-aware heuristics.
These evaluations were carried on a machine with Intel Xeon CPU and 16GB of memory running Ubuntu 16.04. Table B.2 describes the value (or range) of various parameters involved in evaluations of ILP and heuristic.

Table B.2: Description of parameters and decision variables

Parameter	Value or range	Parameter	Value or range		
$\mid S \mid$	15, 150	$c_o(f^c)$	3-5		
$R_{cpu}(n)$	20-28	$c_i(f^c)$	(0.40 - 0.60)$c_o(f^c)$		
$R_{acc}(n)$	0,1	f^{vnf}_{acc}, f^{n}_{acc}	0.20		
$R_{bus}(n)$	80-120 (Gbps)	accel. type	a_1	a_2	a_3
Chain length	4	accel. size	0.40	0.28	0.30
t_s	1.0-5.0 (Gbps)	traffic class	I	II	III
		Prob.	0.10	0.20	0.70

B.5.1 Execution time

The comparison of maximum execution times for the ILP model and heuristic method with a single-node failure is shown in Table B.3. As expected, the computational time for the ILP model increases rapidly with the number of VNF-chains. This becomes an issue, especially for the case when the total number of VNF-chains is greater than 15 and execution time for the ILP becomes orders of magnitude higher than that of heuristic. Therefore, solving VNF-chain recovery for problem instances of large-sizes becomes infeasible using the ILP model. However, the heuristic method can be used for larger problems as discussed in the next section.

B.5.2 ILP and Heuristic comparison

Here, we compare the average amount of traffic lost following node failures for each priority class without any traffic-recovery scheme with the

Table B.3: Comparison of maximum execution time for ILP model and heuristic algorithm

Total chains ($\mid S \mid$)	ILP time	Heuristic time
10	230 ms	1.3 ms
15	350 ms	1.8 ms
20	$>$ 100 s	3 ms

amount of traffic lost after running the ILP-based traffic recovery model and heuristic approach. The comparison of traffic-lost for the single-node and multi-node failure is depicted in Fig. B.1 and B.2 with an input of 15 chains.
As expected an exact approach like ILP always has the minimum amount of lost traffic. However, it can be observed that the performance of the heuristic is close to the ILP method. For VNF-chains of high and medium priorities amount of traffic lost is equal with ILP and heuristic approach. The amount of traffic lost for the lowest priority with the heuristic is not more than 15% as that of ILP in all cases. Moreover, the proposed heuristic can be used to solve the problem instances of large sizes.

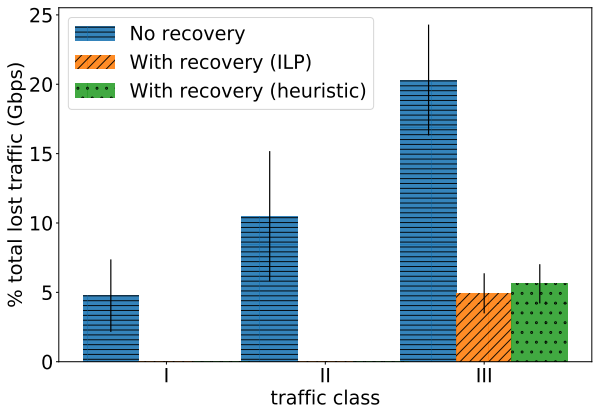


Figure B.1: Performance comparison of ILP and heuristic for a single node failure in terms of lost traffic.

B.5.3 Impact of accelerator allocation criterion

In order to highlight the impact of accelerator-allocation on the amount of lost traffic, we compare the performance of the accelerator-agnostic heuris-

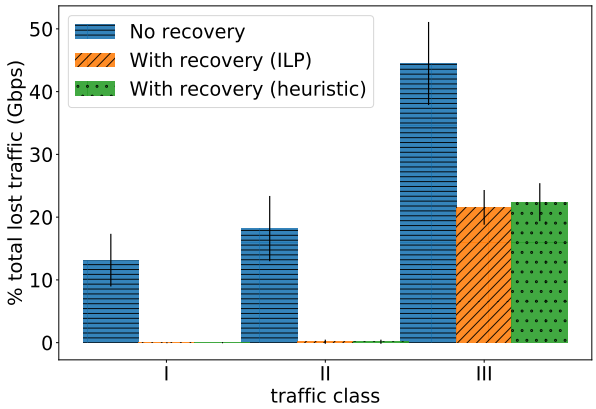


Figure B.2: Performance comparison of ILP and heuristic for multi-node (three) failure in terms of lost traffic.

tic with our heuristic as depicted in Fig. B.3 with an input consisting of 150 chains.
As the higher-priority VNF-chains are restored equally by both ILP and heuristic, we focused only on the lowest-priority traffic. It can be observed that with the accelerator-aware heuristic about 30% of more traffic can be restored as compared to the accelerator-agnostic heuristic. This can be explained by the fact that a better VNF consolidation is achieved with the accelerator-aware heuristic in contrast with the accelerator-agnostic heuristic.

B.6 Conclusion

Advantages of integrating hardware accelerators in NFVi include performance speed-ups and cost savings due to the reduction in VNF CPU usage. In order to minimize losses incurred to telcos due to failures in heterogeneous NFV environments, VNF-chain recovery models need to be modified. This paper presented an ILP formulation to model the accelerator-aware, prioritized VNF-chain recovery problems.
As the execution time for exact methods like ILP is high, a greedy-based heuristic is introduced to solve larger instances of this problem. The performance of the heuristic is on par with ILP as far as the restoration of high and medium priority VNF-chains is concerned, with a slight penalty only for low-priority VNF-chains. The evaluation also shows that the performance

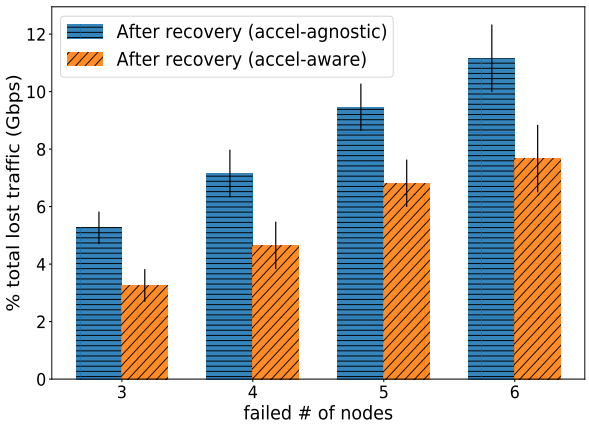


Figure B.3: Impact of accelerator allocation on the heuristic performance in terms of lost traffic.

of accelerator-aware heuristic is 30% better than the accelerator-agnostic heuristic in terms of the total amount of traffic lost.
For future work, we would investigate the availability of VNF-chains hardware accelerators along with usual reliability parameters like VNF and server node reliabilities.

B.7 Acknowledgments

This work was funded through NGPaaS, under the grant number 761557, in the scope of the European Commission Horizon 2020 and 5G-PPP programs.

References

[1] L. Linguaglossa, S. Lange, S. Pontarelli, G. Rétvári, D. Rossi, T. Zinner, R. Bifulco, M. Jarschel, and G. Bianchi. *Survey of Performance Acceleration Techniques for Network Function Virtualization.* Proceedings of the IEEE, 107(4):746–764, 2019.

[2] B. Addis, D. Belabed, M. Bouet, and S. Secci. *Virtual network functions placement and routing optimization.* In 2015 IEEE 4th International Conference on Cloud Networking (CloudNet), pages 171–177. IEEE, 2015.

[3] M. F. Bari, S. R. Chowdhury, R. Ahmed, and R. Boutaba. *On orchestrating virtual network functions.* In 2015 11th International Conference on Network and Service Management (CNSM), pages 50–56, Nov 2015. doi:10.1109/CNSM.2015.7367338.

[4] H. Fan, Y. Hu, S. Zhang, and Q. Ren. *Hardware Acceleration Resource Allocation Mechanism for VNF.* Procedia computer science, 131:746–755, 2018.

[5] M. T. Beck, J. F. Botero, and K. Samelin. *Resilient allocation of service Function chains.* In 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pages 128–133, Nov 2016. doi:10.1109/NFV-SDN.2016.7919487.

[6] A. Tomassilli, N. Huin, F. Giroire, and B. Jaumard. *Resource requirements for reliable service function chaining.* In 2018 IEEE International Conference on Communications (ICC), pages 1–7. IEEE, 2018.

[7] F. Carpio and A. Jukan. *Improving reliability of service function chains with combined vnf migrations and replications.* arXiv preprint arXiv:1711.08965, 2017.

[8] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet. *Dynamic Hardware-Acceleration of VNFs in NFV Environments.* In 2019 Sixth International Conference on Software Defined Systems (SDS), pages 254–259. IEEE, 2019.

[9] S. Dräxler and H. Karl. *SPRING: Scaling, Placement, and Routing of Heterogeneous Services with Flexible Structures.* In 2019 IEEE Conference on Network Softwarization (NetSoft), pages 115–123. IEEE, 2019.

[10] G. P. Sharma, W. Tavernier, D. Colle, and M. Pickavet. *VNF-AAP: Accelerator-aware Virtual Network Function Placement.* working paper or preprint, September 2019. Available from: https://hal.archives-ouvertes.fr/hal-02292930.

[11] IBM. *IBM ILOG CPLEX Optimization Studio.* Available from: https://www.ibm.com/analytics/cplex-optimizer.