5-6-2019

Multicast Aware Virtual Network Embedding in Software Defined Networks

Evrim Guler
Georgia State University

MULTICAST AWARE VIRTUAL NETWORK EMBEDDING IN SOFTWARE DEFINED NETWORKS

by

EVRIM GULER

Under the Direction of Xiaojun Cao, PhD

ABSTRACT

The Software Defined Networking (SDN) provides not only a higher level abstraction of lower level functionalities, but also flexibility to create new multicast framework. SDN decouples the low level network elements (forwarding/data plane) from the control/management layer (control plane), where a centralized controller can access and modify the configuration of each distributed network element. The centralized framework allows to develop more network functionalities that can not be easily achieved in the traditional network architecture.

Similarly, Network Function Virtualization (NFV) enables the decoupling of network services from the underlying hardware infrastructure to allow the same Substrate (Physical) Network (SN) shared by multiple Virtual Network (VN) requests. With the network virtualization, the process of mapping virtual nodes and links onto a shared SN while satisfying the computing and bandwidth constraints is referred to as Virtual Network Embedding (VNE), an NP-Hard problem. The VNE problem has drawn a lot of attention from the research community.

In this dissertation, we motivate the importance of characterizing the mode of communication in VN requests, and we focus our attention on the problem of embedding VNs with one-to-many (multicast) communication mode. Throughout the dissertation, we highlight the unique properties of multicast VNs and explore how to efficiently map a given Virtual Multicast Tree/Network (VMT) request onto a substrate IP Network or Elastic Optical Networks (EONs). The major objective of this dissertation is to study how to efficiently embed (i) a given virtual request in IP or optical networks in the form of a multicast tree while minimizing the resource usage and avoiding the redundant multicast tranmission, (ii) a given virtual request in optical networks while minimizing the resource usage and satisfying the fanout limitation on the multicast transmission. Another important contribution of this dissertation is how to efficiently map Service Function Chain (SFC) based virtual multicast request without prior constructed SFC while minimizing the resource usage and satisfying the SFC on the multicast transmission.

INDEX WORDS: Multicast, Software Defined Networks, Virtual Network Embedding, Network Function Virtualization

MULTICAST AWARE VIRTUAL NETWORK EMBEDDING IN SOFTWARE DEFINED NETWORKS

by

EVRIM GULER

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2019

MULTICAST AWARE VIRTUAL NETWORK EMBEDDING IN SOFTWARE DEFINED NETWORKS

by

EVRIM GULER

DEDICATION

I dedicate this work to my beautiful wife (Tugba Guler), son (Arda Guler), and my beloved family.

For their love, encouragement, and endless support...

ACKNOWLEDGEMENTS

This dissertation work would not have been possible without the support of many people. I want to express my gratitude to my dear advisor Dr. Xiaojun Cao, for believing in me at first, staying behind me at difficult times, and guiding me with his expertise. I would also like to thank each of my committee members - Dr. Anu Bourgeois, Dr. Jonathan Shihao Ji, and Dr. Yi Zhao, for their guidance and encouragement.

A special note of thanks to all my professors, back in Turkey at Ege University, the University of Michigan-Flint and Georgia State University who contributed to my growth, both personally and professionally. Finally, I take this opportunity to thank all my good friends, especially Yasin, Berkay, Ahmet, Deniz, Murat, Rahim, Olcay, Bayram, Harun and Abbas, who made my last five years more enjoyable.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

- SDN - Software Defined Networking
- NFV - Network Function Virtualization
- SN - Substrate (Physical) Network
- VN - Virtual Network
- VNE - Virtual Network Embedding
- VMT - Virtual Multicast Tree/Network
- EONs - Elastic Optical Networks
- API - Application Programming Interface
- REST - Representational State Transfer
- NFVI - Network Function Virtualization Infrastructure
- COTS - Commercial-Off-The-Shelf
- VNF - Virtual Network Function
- N-PoPs - Network Point of Presences
- DPI - Deep Packet Inspection
- VPN - Virtual Private Network
- InPs - Infrastructure Providers
- VNO - Virtual Network Operator
- SP - Service Provider

- VNM - Virtual Node Mapping
- VLM - Virtual Link Mapping
- MVN - Multicast Virtual Network
- MVNE - Multicast-Aware Virtual Network Embedding
- VMTE - Virtual Multicast Tree Embedding
- EGNM - Enhanced Greedy Node Mapping
- CC - Closeness-Centrality
- IF - Impact Factor
- CC-MVNE - Closeness-Centrality based Multicast Virtual Network Embedding
- VMTE-IF - Virtual Multicast Tree Embedding based on dynamic Impact Factor
- VOR - Virtual Optical Request
- SON - Substrate Optical Network
- SFL - Substrate Fiber Link
- VONE - Virtual Optical Network Embedding
- MVONE - Multicast-Aware Virtual Optical Network Embedding
- VOMTE - Virtual Optical Multicast Tree Embedding
- IF-VOMTE - Impact Factor based Virtual Optical Multicast Tree Embedding
- VNA - Virtual Network Assignment
- CPU - Central Processing Unit
- BFS - Breadth First Search

- MR - Multicast Request
- GNM - Greedy Node Mapping
- FF - First Fit
- CF - Candidate Factor
- RCM - Redundancy Checking Mechanism
- FFNM-SP - First Fit Node Mapping with the Shortest Path
- GNM-SP - Greedy Node Mapping with the Shortest Path
- MSE-FL - Multicast Service Embedding in EONs with Fanout Limitation
- C-DB-SPT - Centrality-based Degree Bounded Shortest Path Tree
- GN-DB-SPT - Greedy Node Mapping and Degree Bounded Shortest Path Tree
- GRC-DB-SPT - Global Resource Capacity based Degree Bounded Shortest Path Tree

PART 1

INTRODUCTION

Computer networks are generally constructed from a large number of network devices (e.g., routers, switches and different types of middleboxes that manipulate traffic for purposes other than packet forwarding such as a firewall) implemented by many complex protocols [1]. To realize a wide range of network applications and events, network operators have the responsibility of configuring network policies and transforming these high-level policies into low-level commands to manage network devices while adapting to dynamically changing real network conditions. In addition to network policy configurations, the network operators need to manage very complex tasks with limited access to the network tools. Hence, the management and performance tuning of network systems are quite challenging while vertically-integrating network devices exacerbates the network operations and administrations.

Morever, the invincible/unsurmountable network challenge to be faced by network operators or researchers is referred to as ”Internet Ossification,” which has the huge deployment base and is considered the main part of critical physical infrastructure such as the transportation and power grids. Thus, the Internet has become extremely difficult to be evolved in terms of its physical infrastructure, protocols and performance. However, emerging current Internet applications and services has become more complex and dynamically demanding that is able to evolve to address new network challenges.

To facilitate the network evolution, the new idea of ”Programmable Networks” has been proposed as ”Software Defined Networking” in which the forwarding hardware is decoupled from the control plane.

1.1 Software Defined Networking

Software Defined Networking (SDN) defines a new concept to design and manage computer networks. The SDN has two defining characteristics as shown in Fig. 1.1: (i) an SDN decouples the control plane, which manages how to operate the traffic, from the data plane, which forwards the traffic flow based on the decisions of a control plane, (ii) an SDN consolidates the centralized control plane to handles multiple data plane elements [2]. The SDN control plane practices directly operation over the state in the network elements of the data plane (i.e., routers, switches, and other middleboxes) via a well-defined Application Programming Interface (API) (i.e., OpenFlow [3]).

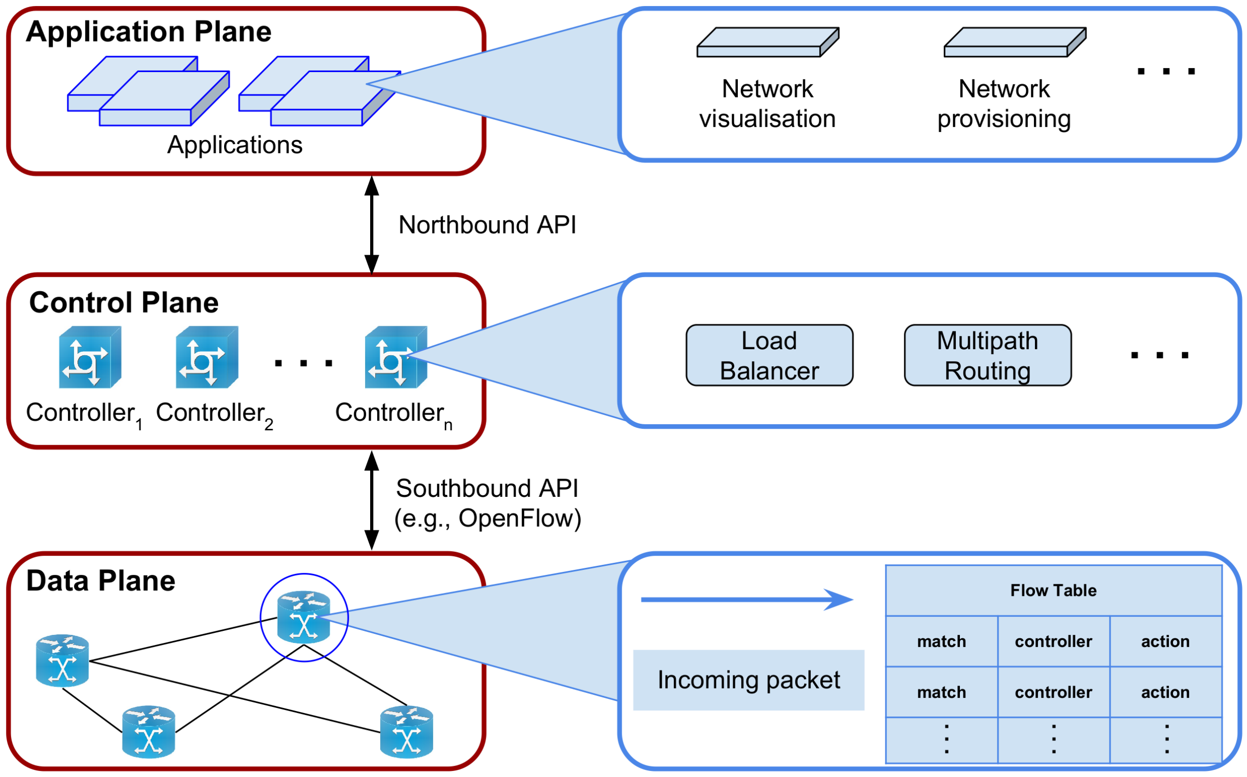


Figure (1.1) SDN Architecture: conceptual planes and communication interfaces

In Fig. 1.1, the architecture of SDN introduces three conceptual planes and communication interfaces, which are (i) application plane, (ii) control plane, and (iii) data plane. The application plane runs applications over the network infrastructure, and allows to perform modifications regarding network aspects (i.e., network policies and routing behavior). The control plane provides control logics (i.e., routing schemes) to manage the collected information from the switches of the data plane such as flow statistics. In the control plane, we have the global view of the network system to be able to make traffic distributions and enforce Quality of Service policies. In the data plane, the physical devices are responsible to forward data when the OpenFlow switches have programmable flow tables that can be dynamically configured by the control plane.

Furthermore, to occur the communication between the planes, the northbound API enables to program network controller by abstracting the network data with REpresentational State Transfer (REST) protocol. On the other hand, the southbound API implements the communication between control plane and data plane to configure switches with forwarding actions based on received notifications of incoming packets from the data plane by using OpenFlow protocol.

As a consequence of the SDN principles, the separation of network policies implemented in switching hardware and the forwarding of traffic is key to provide the flexibility by breaking the network control problem into the tractable pieces, and introduce new abstractions in networking by simplifying network management and facilitating network evolution and innovation [4]. The SDN and OpenFlow, even though, started as academic experiments [2], most vendors of commercial switches currently supports OpenFlow API in their equipment. As an example, Google has a deployed SDN to interconnect its data centers across the globe by helping the company to improve operational efficiency and significantly reduce costs [5].

With the development of Software Defined Networking, service providers are interested in facilitating deployment of new network services by abstracting network devices and appliances to fill specialized roles such as routing, switching, spam filter, load balancer, firewall, and so forth. Integrated with SDN, the Network Function Virtualization (NFV) is proposed to address flexible provisioning, deployment and centralized management of virtual network functions while offering agile traffic steering and joint optimization of network functions and resources.

1.2 Network Function Virtualization

Network Function Virtualization (NFV) separates software instance from hardware platform and decouples the functionality to provision more flexible and faster network service while transforming the management of underlying network infrastructure by leveraging the virtualization technology [6]. Essentially, NFV implements and runs network functions (i.e., well-defined functional behavior such as firewall, deep packet inspection, virtual private network, etc.) on the hardware (i.e., industry standard servers, storage and switches) through software virtualization, as shown in Fig. 1.2. To implement lower cost agile network infrastructure as an innovative step, the benefits of NFV can dramatically change the landscape of telecommunications industry. The NFV may reduce capital investment and energy consumption by consolidating networking appliances, decrease the time to market of a new service by changing the typical innovation cycle of network operators (e.g., through software-based service deployment), and rapidly introduce targeted and tailored services based on customer needs.

NFV reduces the equipment cost and forming a strong, scalable and elastic network ecosystem while decreasing the time to market. Firstly, the NFV is minimizing the spent time on a new service evaluation and testing for automated Network Function Virtualization Infrastructure (NFVI) management and orchestration. Next, NFV allows to run software based network functions on Commercial-Off-The-Shelf (COTS) hardware rather than purpose built hardware for network operators and service providers. NFV network ecosystem also builds general purpose and cheap infrastructure, where the software based functions (i.e., Virtual Network Functions (VNFs)) can be executed on middle-boxes. To place VNFs on the Network Point of Presences (N-PoPs) of the underlying network and construct VNF chain in a particular sequence are to achieve better network performance while providing as least good service provision as the original proprietary hardware [7].

In other words, SDN and NFV provide programmable network control and flexible network services. The virtualization techniques in SDN and NFV allow multiple tenants to

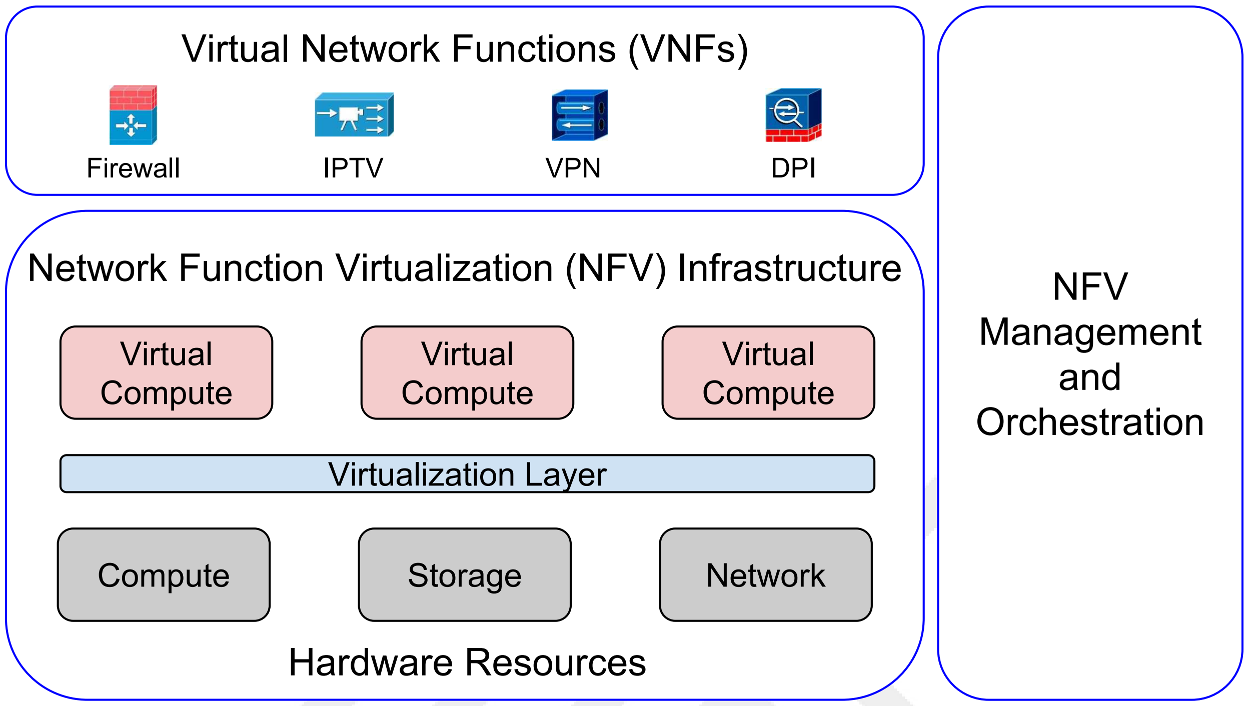


Figure (1.2) Hardware-based appliances for network services such as Firewall, Deep Packet Inspection (DPI), Virtual Private Network (VPN), IPTV, to software-based NFV solutions

share the same physical infrastructure while reducing the CAPEX and OPEX, and increasing the physical resource utilization [8]. The network virtualization also has an important role to support simultaneously multiple architectures for the future Internet and a major challenge how to deal with efficient usage of physical network resource for multiple tenants, referred as virtual network embedding.

1.3 Virtual Network Embedding

The primary entity in the network virtualization is a Virtual Network or Request (VN). A VN is a combination of active and passive network elements (virtual network nodes and links) on the top of a Substrate Network (SN). Each virtual node in a VN is interconnected through a virtual link to form a virtual network topology. By virtualizing both node and link resources of an SN, multiple virtual network topologies with widely varying characteristics can be created and co-hosted on the same (shared) physical hardware. Moreover, the introduced abstraction by the resource virtualization mechanism allows network operators to highly flexible (or dynamically) manage and modify network systems. The introduction of network virtualization separates the management and business roles of the Service Provider (SP) by identifying three main players as shown in Fig. 1.3: (i) the Virtual Network Provider (VNP) which assembles virtual resources from one or more Infrastructure Providers (InPs), (ii) the Virtual Network Operator (VNO) which installs, manages and operates the VN according to the needs of the SP, and (iii) the SP which is free of management and concentrates on business by using the VNs to offer customized services.

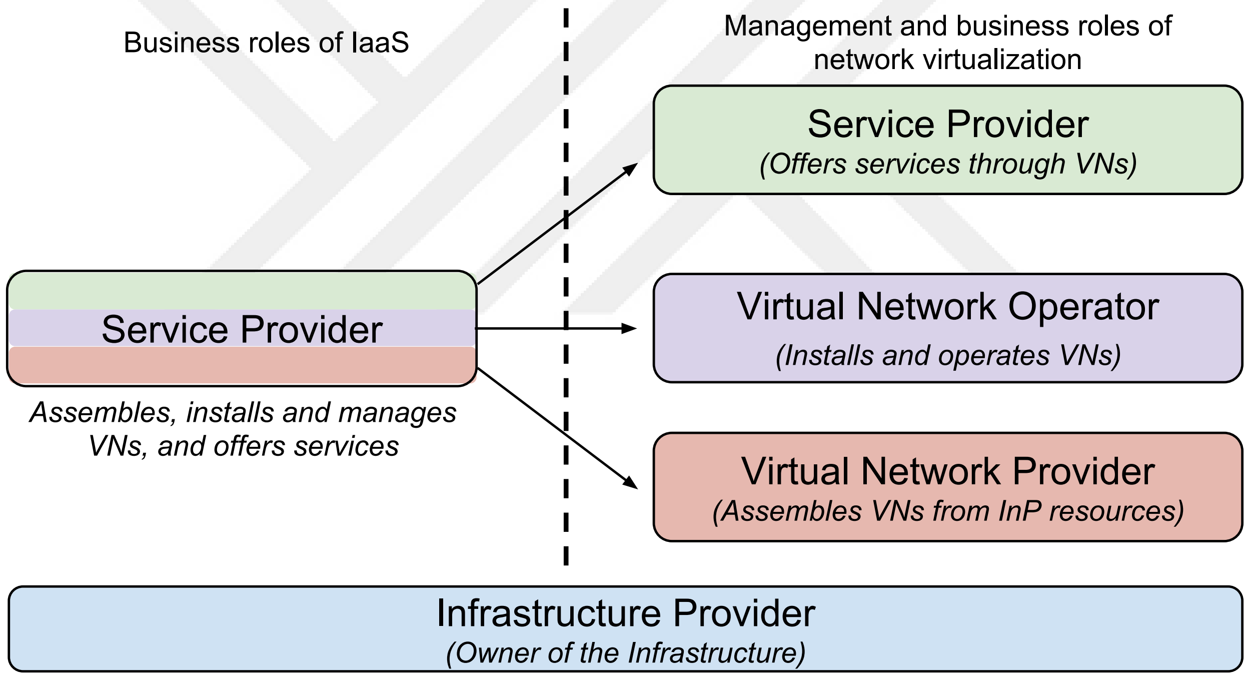
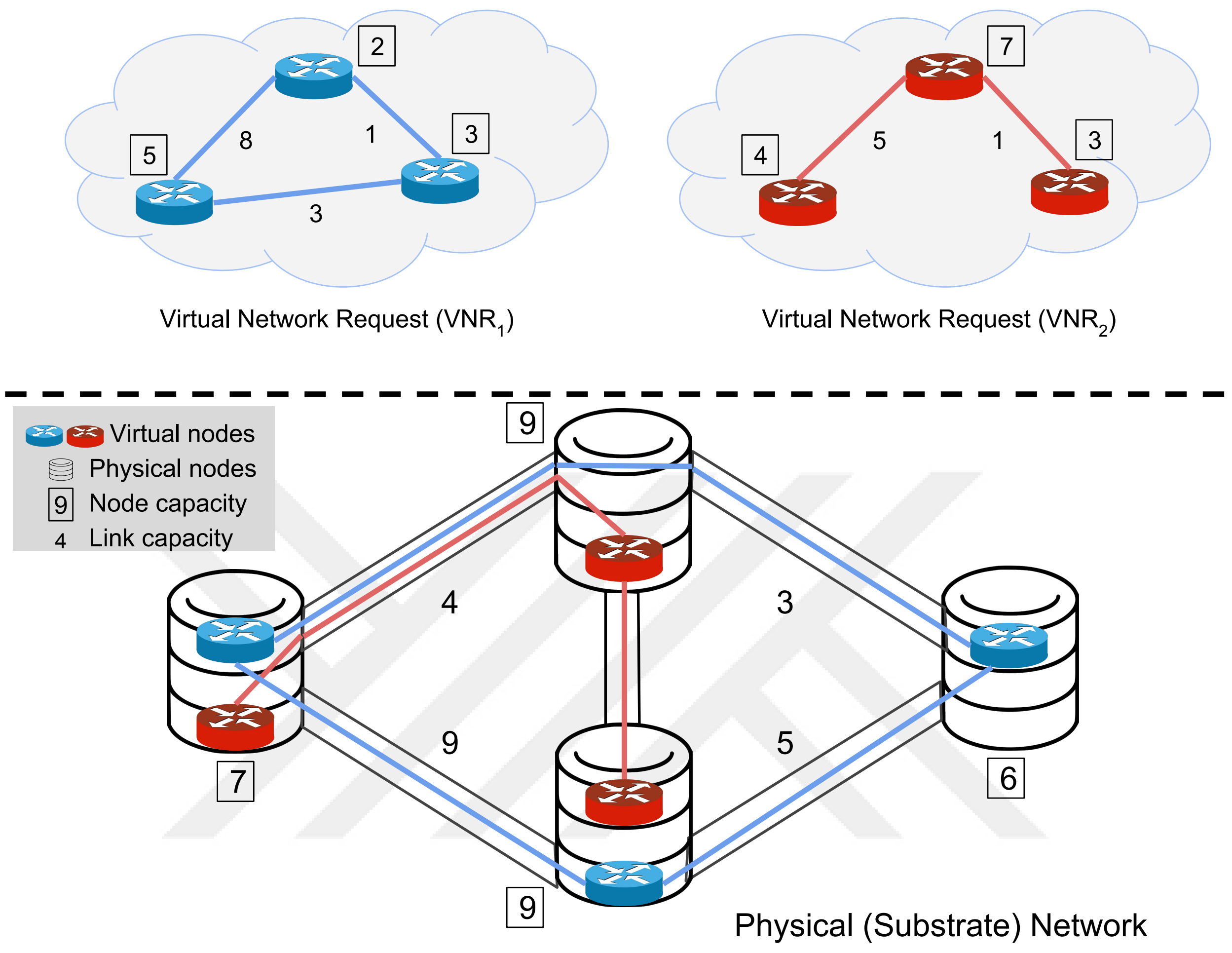

Business roles of IaaS
Management and business roles of network virtualization
Service Provider
(Offers services through VNs)
Service Provider
Assembles, installs and manages VNs, and offers services
Virtual Network Operator
(Installs and operates VNs)
Virtual Network Provider
(Assembles VNs from InP resources)
Infrastructure Provider
(Owner of the Infrastructure)

Figure (1.3) Next Generation Network Model: Network virtualization

The problem of embedding virtual networks in a shared substrate network is the main resource allocation challenge in network virtualization and referred to as the Virtual Network Embedding (VNE) problem, where the benefit gained from existing hardware can be maximized while mapping of virtual resources onto physical hardware. Optimal resource allocation, leading to self-configuration and organization of next generation networks, will be necessary to provide customized end-to-end guaranteed services to end users.



2
5
8
1
3
3
Virtual Network Request (VNR$_1$)
7
4
5
1
3
Virtual Network Request (VNR$_2$)
Virtual nodes
Physical nodes
9 Node capacity
4 Link capacity
9
4
3
7
9
5
6
9
Physical (Substrate) Network

Figure (1.4) The process of Virtual Network Embedding

The VNE problem deals with the allocation of virtual resources both in nodes and links as shown in Fig. 1.4. Therefore, the problem generally includes two important subprocesses: (i) **Virtual Node Mapping (VNM)** and (ii) **Virtual Link Mapping (VLM)**. In the former subprocess, virtual nodes have to be allocated in substrate (physical) nodes that provide sufficient computing resource. In the latter subprocess of VNE, each virtual link in VN is mapped onto a substrate (physical) path by reserving sufficient bandwidth (or subcarriers). Solving optimally VNE problem is NP-Hard, as it is related to the multi-way separator problem [9]. Even with a given virtual node mapping, the problem of optimally allocating a set of virtual links to substrate paths reduces to the unsplittable flow problem also known as NP-Hard [10]. Therefore, optimal solutions in a polynomial time can only be gained for small problem instances, and currently main focus of work within the research community is on heuristic or meta-heuristic approaches.

1.4 Multicast Virtual Network Embedding

Many schemes have been proposed to address the general VN mapping problem for unicast services (e.g., [11–15]) to efficiently design provisioning strategies for such VN requests.

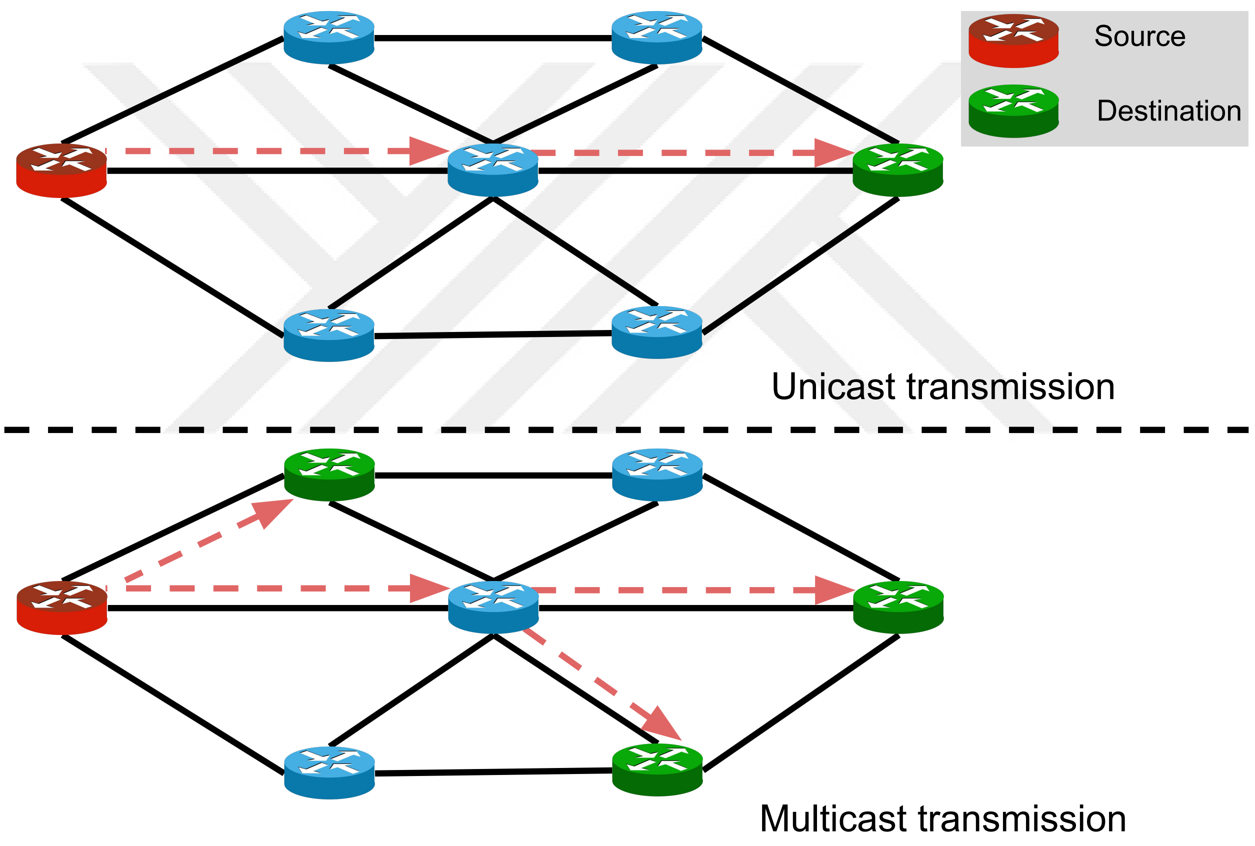


Figure (1.5) Unicast and Multicast Transmission

However, in many widely used applications, emerging distributed file systems, big data and Internet applications such as IPTV, video-conferencing, live stock quotes demand multicast communication to improve the utilization of the physical resource. Unlike the unicast service where a single sender transmits data to a single receiver, multicast service requires that the same data packet flows through a selected group of destinations (receivers), which can share the data transmission along the common links (e.g., [16–19]), as shown in Fig. 1.5. In other words, handling multicast communication as unicast requires transmitting multiple copies of same data to reach each receiver from same source. Thus, to handle a one-to-many communication as multicast, these multiple unicast transmission can be replaced by a single multicast transmission while reducing the computation effort at the same source, bandwidth consumption in the network, and improving the throughput of applications with the response time. Thus, supporting multicast service for high traffic in cloud providers is essential in their data centers. To this extent, multicast service in data center network has become prominent research topic with focusing the resource allocation problem Multicast Virtual Networks (MVNs) [20]. The problem of network resource allocation consists of VNM and VLM processes. While allocating network resources is widely discussed for unicast VNs, embedding MVNs is greatly different than unicast services. In MVNs, the routing of a traffic flow from a source to multiple receivers (destinations) consists of constructing a multicast distribution tree to avoid redundant traffic.

Therefore, in this work, the Multicast-Aware VNE (MVNE) or Virtual Multicast Tree Embedding (VMTE), Multicast Service Embedding with Fanout Limitation (MSE-FL) and Multicast-aware Service Function Tree Embedding (M-SFTE) problems have been defined by minimizing required resource allocation in substrate networks while reducing redundant multicast data transmission, satisfying the fanout limitation and functionality requirements, respectively.

PART 2

RELATED WORK

2.1 Virtual Network Embedding (VNE)

2.1.1 VNE over IP Networks

The VNE problem is known to be NP-hard [9]. Hence, many researchers have investigated efficient heuristic and meta-heuristic approaches [11–15].

The authors in [11] introduce the algorithm design to solve on-demand VN assignment problem, which is upon the arrival of a VN request, assigning its topology to the substrate network to achieve low and balanced load on both substrate nodes and links, namely VN assignment without reconfiguration (VNA-I) and VN assignment with reconfiguration (VNA-II) problems. For the VNA-I problem, where the VN assignment is fixed throughout the VN lifetim, the authors develop a basic scheme to achieve near optimal substrate node performance and use it as a building block for all other advanced algorithms. For the VNA-II problem, a selective VN reconfiguration scheme that prioritizes the reconfiguration for the most critical VNS is developed to achieve most performance benefits of the reconfiguration without excessively high cost.

In [12], the VN embedding problem by proposing a more flexible substrate network to better support virtual network embedding is rethought. The flexibility of the substrate network includes path splitting and migration. Path splitting (i.e. multipath) has been a recurring theme in many network research topics, and the authors demonstrate the power

Table (2.1) Classification of Multicast Virtual Network Embedding

	VNE	MVNE
IP Network	[11–15]	[20–28]
Optical Network	[29–33]	[16–18]

of multipath in substrate network for more cost-effective virtual network embedding while attaining better resource utilization.

Policy-based inter-domain VN embedding (PolyViNE) framework to heuristically address the inter-domain VN mapping problem is investigated in [13]. The authors introduce policy-based end-to-end VN embedding framework that embeds VNs across multiple InPs in a globally distributed manner while allowing each concerned InP to enforce its local policies. PolyViNE introduces a distributed protocol that coordinates the participating InPs and ensures competitive pricing through repetitive bidding at every step of the embedding process.

In [14], ViNEYard proposes a collection of VN embedding algorithms to leverage better coordination between node mapping and link mapping. ViNEYard algorithms include Deterministic VN Embedding (D-ViNE), Randomized VN Embedding (R-ViNE), and their extensions. The major contributions of the algorithms:

- D-ViNE and R-ViNE are two rounding-based VN embedding algorithms that increase the VN request acceptance ratio and InP revenue without incurring additional cost by leveraging coordinated node and link mapping

- D-ViNE-LB and R-ViNE-LB are extensions to D-ViNE and R-ViNE that focus on balancing load across substrate resource to improve the acceptance ratio, often causing inflated link utilization across the substrate network

- A generalized window-based VN embedding (WiNE), which allows batch processing of VN requests, is a mechanism for equipping any existing online VN embedding algorithm with lookahead capabilities

- ViNEYard model presents a flexible and extensible mathematical programming formulation of the VN embedding problem

Path-based Integer Linear Programming (ILP) model for VNE problem (P-VNE) is proposed to obtain an optimal solution for some cases by using of the column generation in [15]. Based on the dual formulations of the P-VNE model, a column generation process is presented, which can be embedded into a branch-and-bound framework to effectively resolve VNE problem optimally in practice.

2.1.2 VNE over Optical Flexible Networks

As Virtual Optical Network Embedding (VONE) problem known to be NP-Hard [34], many researchers have explored efficient heuristic or meta-heuristic approaches to solve the VONE problem [29–33].

In [29], the authors study a dynamic traffic scenario by using an auxiliary graph model to address the multilayer optical network embedding while minimizing bandwidth blocking ratio in three possible underlying substrates for inter-datacenter networks, namely an optical-layer-based, electrical-layer-based and multilayer-based (optical and electrical layer) substrate network. For the node mapping, the applied policies select a node with different wavelength or electrical resource distribution by setting the adjacent edge's weight of the node in the auxiliary graph (AG). Similarly, different link-mapping policies can also be designed by adjusting weights of the edges of the AG. Hence, the main contribution of the AG in [29] is to provide a general approach for addressing a dynamic VNE over multilayer optical networks.

The Alignment and Consecutiveness-Aware Transparent Virtual Network Embedding (ACT-VNE) algorithm in [30] uses consecutiveness-aware node ranking to measure node capacities for the mapping process in transparent VNE (t-VNE). The algorithm takes into account the spectrum alignment and consecutiveness between adjacent fiber links while mapping virtual nodes onto the substrate nodes. The authors also propose an extended scheme, namely Importance, Alignment and Consecutiveness-aware Transparent Virtual Network Embedding (iACT-VNE) that assigns virtual nodes to substrate nodes within close proximity to establish connections with less bandwidth utilization.

The authors in [31] present Integer Linear Programming (ILP) formulations and a lightweight Greedy Randomized Adaptive Search (GRASP) heuristic algorithm to accommodate the Virtual Optical Request (VOR) over a transparent optical substrate network. The Virtual Optical Network Allocation (VONA) is solved over transparent and opaque services while validating the benefits of exact ILP and GRASP against simpler VONA techniques.

The dynamic transparent virtual network embedding algorithm, which considers node and link mapping jointly, is proposed for network virtualization over optical orthogonal frequency-division multiplexing (O-OFDM) based elastic optical infrastructure in [32] in order to minimize VOR blocking probability. For each VOR, the algorithm first transfers the substrate optical network into a layered-auxiliary-graph according to the spectrum usage of each fiber link. In the next step, a node mapping is applied by considering the local information of all substrate nodes, and the link mapping is accomplished in a single layer of auxiliary graph.

A static and dynamic versions of virtual topology mapping problems based elastic optical networks have been explored in [33]. The authors propose ILP model and two heuristic approaches, which are greedy based First Fit (FF) and Link List (LL) algorithms, to maximize the subcarrier utilization while minimizing blocking ratio for dynamic traffic. The FF algorithm maps a virtual node with the lowest index value from a VOR onto a substrate candidate node that has enough available resource with the lowest index value. On the other hand, the LL algorithm considers both node requirement and their bit rate request and sorts virtual and substrate nodes with their connected bit rates of links in decreasing order. After completing the ordering, the algorithm maps a most constrained virtual node in the virtual node list onto a substrate node with the highest order.

The majority of these existing V(O)NE approaches targets on the point-to-point (unicast) data traffic. With the rising demand on simultaneous bulk data transfer and the support to path multiplicity from network topologies have made multicast necessary and practical recent data centers.

2.2 Multicast Virtual Network Embedding (MVNE)

With more Internet applications demanding multicast transmission such as IPTV, video-conferencing and live stock quotes, Multicast VNE (MVNE) recently attracts much attention to transmit the same content from one or multiple starting points through multiple end points.

2.2.1 MVNE over IP Networks

In [20–28], the works try to minimize the resource allocation on SN to satisfy only multicast VN requests.

The authors in [21] propose the mapping problem in the context of Virtual Multicast service-oriented Network subject to Delay and Delay Variation Constraints (VMNDDVC) and define an efficient heuristic algorithm to tackle the problem based on a sliding window approach while minimizing the cost of VMNDDVC request mapping and achieving the load balancing to increase the accepting ratio of Virtual Multicast Network (VMN) request. In order to meet these constraints in VMNDDVC, a sliding window method is proposed to construct a set of feasible paths and solve the problem based on feasible paths.

In [22], a multicast mapping algorithm (MMPC) is proposed to achieve MVNE while improving the efficiency of physical resource utilization. The authors explore the multicast networks mapping for enabling Multiple Description Coding (MDC) based video applications. The MMPC algorithm maps multicast trees onto the substrate network through the path convergence approach to meet diverse VN request requirements while minimizing the VNM mapping cost to improve global physical resources utilization efficiency.

An improved layered overlay multicast scheme based on Embedded Structure by introducing the gossip scheme into the data distribution progress to deal with the problems of transmission isolation and unbalanced load allocation is proposed in [23]. With the help of gossip process, the algorithm achieves cross-cluster transmission to maintain a more balanced overlay topology by addressing the leaf members while delivering to the cluster leaders who have competence to dispose the expected data in higher topology hierarchy.

In [24], the authors introduce a heuristic algorithm called Multicast Virtual Network Embedding with strategies of waiting tolerant and load prediction (MVNE-WL) and consider the delay constraints to resolve the VNE problem for multicast services.

The importance of characterizing the type of communication for VN requests with multicast communications in cloud computing is studied in [25]. The authors represent 3-step heuristic algorithm to solve MVNE problem with end-delay and delay variation constrants when the location of all the virtual machines in a given multicast VN is unknown. The proposed algorithm is consisting of graph pruning, finding feasible subgraph and performing node mapping schemes with well-known Breadth First Search (BFS) and Depth First Search (DFS) to increase the mapping revenue and reduce the blocking ratio. Also, the authors propose a Tabu-based search for solving the MVNE problem for multicast services with heterogeneous resource demands over arbitrary network topologies in [20] while increasing network admissibility in considerably fast runtime.

In [26], the authors consider the delay constraints of each VN edge while mapping and routing the virtual link on the substrate links, and the constraint on locations of each VN node while mapping the virtual node onto a substrate node in order to provision a multicast virtual request. Also, in this work, the survivability of MVNE is studied while minimizing VMN request mapping by implementing Mixed ILP (MILP) and proposing heuristic algorithm, namely Survivable Multicast Virtual Network Provisioning (SMVNP) based on minimum set covering scheme.

The authors of [27] propose VNE schemes on fat-tree DCNs, which places the Virtual Machines (VMs) of a multicast-capable VN without any disturbance to existing traffic and manages to keep VMs in an even more compact way to reduce cost by allowing a small degree of VM migration.

In [28], the impact of physical link failure on VMNs is investigated while minimizing link mapping cost and increase admittance ratio with end-delay and delay variation requirements. The proposed algorithm in [28] finds a constrained shortest path between multicast nodes utilized by the failed VMNs to minimize the recovery time while evaluating the algorithm in terms of link mapping cost, restoration time and admittance ratio.

2.2.2 MVNE over Optical Flexible Networks

In addition to the IP-based networks, multicast services draw much attention in the traditional Elastic Optical Networks [16–18].

Similarly, the layered based Shortest Path Tree (SPT) and Minimum Spanning Tree (MST) approaches are employed to minimize the required resource for multicast transmission in [16]. For each multicast request in EONs, the proposed algorithms decompose the physical topology into several layered auxiliary graphs according to the network spectrum utilization.

In [17], the authors propose two multicast-capable routing and spectrum allocation algorithms, namely, SPT and Steiner Minimal Tree (SMT), to maximize the resource utilization while reducing the blocking probability of multicast services.

In [18], the authors propose two ILP models based on SPT and MST schemes, and the Genetic Algorithm (GA) for online and offline multicast services. The joint and separate ILP models optimize all multicast requests together and one request at a time by sequentially handling the requests, respectively. To reduce the computational complexity, the authors also propose Genetic Algorithm (GA) to minimize the cost and blocking probability of multicast services.

Based on multicast services in traditional EONs, the authors of [35] define the multicast-service oriented VN mapping that can support big data applications over EONs. In [35], an efficient heuristic algorithm, called Integrated Genetic and Simulated Annealing (IGSA), is proposed to minimize spectrum consumption and blocking probability while the algorithm jointly optimizes the node and link mapping for all the multicast request in a global way. To study the reliability in Multicast Virtual Optical Network Embedding (MVONE), the authors of In [36] introduce efficiently VN mapping for multicast services over Orthogonal Frequency Division Multiplexing (OFDM)-based EONs while taking into consideration the max-min fairness in terms of reliability among distinct VNs. The proposed Reliability-Aware

Genetic (RAG) algorithm addresses to reliable multicast VN mapping with a low computational complexity while achieving higher reliability fairness, lower bandwidth (spectrum) consumption and transmission delay.

PART 3

MULTICAST-AWARE VIRTUAL NETWORK EMBEDDING PROBLEM FORMULATION

3.1 Substrate Network

The Substrate Network (SN) is modeled as an undirected graph $G_S = (N_S, L_S)$, where N_S and L_S are the set of substrate nodes and substrate links, respectively.

$$N_S = \{n_1, n_2, ..., n_m\}, \quad m = |N_S| \in Z^+$$

$$L_S = \{l_1, l_2, ..., l_k\}, \quad k = |L_S| \in Z^+$$

The cardinality of sets ($|N_S|$, $|L_S|$) denoted by m and k is the number of nodes and edges in SN, respectively. In order to ensure a connected graph, the number of links needs to be in the range of boundaries as in the following expression [37].

$$m - 1 \leq k \leq m * (m - 1)/2$$

In the SN, each node has a certain computing capacity (e.g., CPU), and each link has a bandwidth capacity and weight (e.g., distance or delay).

$$C_S = \{c(n_1), c(n_2), ..., c(n_m)\}, \quad c : N_S \rightarrow Z^+$$

$$B_S = \{b(l_1), b(l_2), ..., b(l_k)\}, \quad b : L_S \rightarrow Z^+$$

$$W_S = \{\omega(l_1), \omega(l_2), ..., \omega(l_k)\}, \quad \omega : L_S \rightarrow Z^+$$

C_S represents the set of $c(n)$ ($\forall n \in N_S$) to specify the available CPU of each substrate node. For a substrate link $\forall l \in L_S$, $b(l)$ in the set of B_S denotes the available bandwidth of this substrate link, while $\omega(l)$ (or $\omega_{ij}, ij \in L_S, \forall i, j \in N_S$) in the set of W_S represents the cost or weight (e.g., distance or delay) of this substrate link.

3.2 Virtual Multicast Tree

A Virtual Request (VR) in the form of multicast tree can be modeled as an undirected graph $G_V = (N_V, L_V)$, where N_V and L_V represent the set of virtual nodes and links, respectively.

$$N_V = \{v_1, v_2, ..., v_p\}, \quad p = |N_V| \in Z^+$$

$$L_V = \{e_1, e_2, ..., e_r\}, \quad r = |L_V| \in Z^+$$

In the set of N_V and L_V, p and r denote the total number of virtual nodes and links, respectively. In the following expression, $c(v)$ in the set of C_V specifies the requested computing resource (i.e., CPU) by a virtual node and $b(e)$ denotes the bandwidth (or number of subcarriers for optical flexible networks) demand from a virtual link.

$$C_V = \{c(v_1), c(v_2), ..., c(v_p)\}, \quad c : N_V \to Z^+$$

$$B_V = \{b(e_1), b(e_2), ..., b(e_r)\}, \quad b : L_V \to Z^+$$

3.3 Multicast Data Transmission Request

We assume that a projected multicast service to run on G_V is denoted by a vector $MR =< s, D, \beta >$ where $\exists s \in N_V$ is the virtual source node, $D = N_V - \{s\}$ is the set of multicast destinations in VR, and $\beta \in Z^+$ is the requested bandwidth for the multicast service. Without loss of generality, we consider that all virtual links request the same amount of multicast bandwidth, while each virtual node requests a random amount of computing resource.

$$D = \{d_1, d_2, ..., d_t\}$$

$$t = |N_V - \{s\}| = |N_V| - |\{s\}| = p - 1$$

3.4 Virtual Multicast Tree Embedding

To map a virtual multicast tree onto a shared SN, the processes of node and link mapping has to be accommodated.

3.4.1 Node Mapping

$$M_{vn} = \begin{cases} 1, & \text{if virtual node } v \in N_V \text{ is mapped onto physical node } n \in N_S \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

$$\sum_{n \in N_S} M_{vn} = 1, \quad \forall v \in N_V \tag{3.2}$$

$$\sum_{v \in N_V} M_{vn} \leq 1, \quad \forall n \in N_S \tag{3.3}$$

$$\sum_{v \in N_V} c(v) * M_{vn} \leq c(n), \quad \forall n \in N_S \tag{3.4}$$

For such one-to-one node mapping, we use M_{vn} to represent whether a virtual node $v \in N_V$ is mapped onto the substrate node $n \in N_S$ and hold all M_{vn} in a list. Eq. 3.2 ensures that each virtual node is mapped onto one substrate node. A substrate node, which is able to satisfy enough available computing resource demanded by the virtual node, can host at most one virtual node from the same VR as shown in Eq. 3.3. To ensure the node mapping, we use Eq. 3.4 to ensure one substrate node with enough computing capacity by embedded a virtual node.

3.4.2 Link Mapping

In the process of link mapping, a physical path (or link) with enough bandwidth (or subcarrier) has to be identified in the SN for each virtual link.

$$l_{ij}^{uv} = \begin{cases} 1, & \text{if physical link } (i,j) \in L_S \text{ is used by a virtual link } (u,v) \in L_V \\ 0, & \text{otherwise} \end{cases} \tag{3.5}$$

$$l_{ij}^{uv} * b(uv) \leq b(ij) \tag{3.6}$$

$$\chi_{ij} = \sum_{uv \in L_V} l_{ij}^{uv}, \quad \forall ij \in L_S \tag{3.7}$$

For link mapping, we define l_{ij}^{uv} in Eq. 3.5 to specify whether or not virtual link ($uv \in L_V$) from virtual node $u \in N_V$ to $v \in N_V$ is using the substrate link ($ij \in L_S$) between substrate node $i \in N_S$ and $j \in N_S$. To ensure demand of each virtual link satisfied by a physical path (or links), Eq. 3.6 provides a substrate link(s) that has enough available bandwidth capacity (or number of subcarriers). Eq. 3.7 denotes how many times a substrate link is used for the mapping of all the virtual links.

During the process of node and link mapping, different mapping strategies will map virtual nodes differently and use different physical paths to embed the virtual links, resulting in various amount of resource consumption. Here, we define the objective function in Eq. 3.8, which is proportional to minimize the total cost of required bandwidth and multicast transmission.

$$\min\left(\beta * \sum_{ij \in L_S} \chi_{ij} * \omega_{ij}\right) \tag{3.8}$$

3.5 Virtual Multicast Tree Embedding Problem

Given the substrate network $G_S = (N_S, L_S)$ and a multicast tree request $G_V = (N_V, L_V)$, how to map the virtual multicast tree onto the substrate network while (i) satisfying the aforementioned constraints of node/link mapping, (ii) minimizing the required resource and the redundant multicast transmissions in the substrate network.

Similar to the traditional VNE, VMTE consists of the subprocess of node and link mapping. As a tree structure can be converted into a mesh network by adding necessary links with zero bandwidth request in a polynomial time (bounded by the number of links) and traditional VNE optimization is proved to be NP-Hard [9], the VMTE problem is also difficult to resolve optimally. Hence, we introduce efficient heuristic algorithms to solve VMTE problem in IP-based and Elastic Optical Networks (EONs), called Closeness-Centrality based Multicast-aware Virtual Network Embedding (CC-MVNE), Virtual Multicast Tree Embedding with dynamic Impact Factor (VMTE-IF) and Dynamic Impact Factor based Virtual Optical Multicast Tree Embedding (IF-VOMTE), in the following section.

PART 4

MULTICAST-AWARE VIRTUAL NETWORK REQUEST EMBEDDING

In general, the existing Multicast-aware Virtual Network Embedding (MVNE) approaches process the node mapping according to the order of the requested computing resource and/or path cost. As shown in Fig. 4.1a, there are three virtual nodes in a multicast VN request where S is the multicast source node, V_1 and V_2 are the multicast destination nodes. The requested CPU resource from node S, V_1 and V_2 are listed along the respective node. Fig. 4.1c shows the substrate node where the number beside a node indicates the available CPU resource and the number on the link is the cost of the link resource. Based on the requested CPU load or link resource/cost as Enhanced Greedy Node Mapping (EGNM) algorithm in [38], the VN request in Fig. 4.1a will be mapped onto Fig. 4.1c as shown in Fig. 4.1e. In other words, nodes S, V_1 and V_2 are mapped to substrate node A, C and B, respectively. The virtual link $S - V_1$ is mapped to a physical path $A - B - C$ while $S - V_2$ is mapped to physical path $A - B$. As a result, when node S sends multicast traffic to node V_1 and V_2, the same multicast traffic goes through paths $A - B - C$ and $A - B$ of the substrate network. Clearly, the same multicast traffic travels the physical link $A - B$ twice (or redundantly), which results in network resource wastage. Similarly, the scenario in Fig. (b-d-f) shows the multicast traffic to node V_1 travels from node A to node B, then to node C. The same multicast traffic (from node V_1) to node V_2 travels from C to B. The fact that the same multicast traffic cycles around the link $B - C$ will lead to low resource usage in the network.

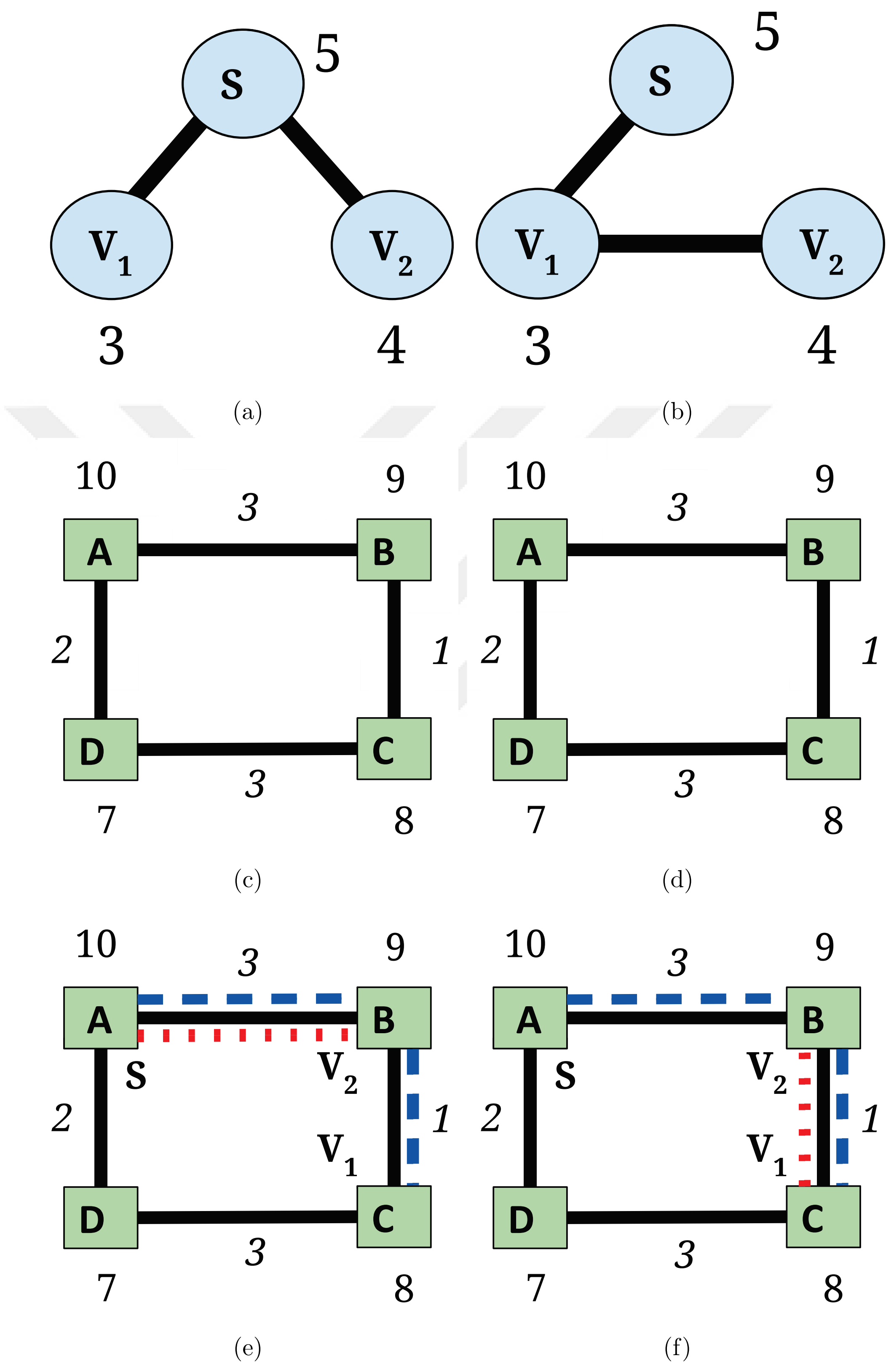


Figure (4.1) Resource wastage in MVNE

In [39], we propose an efficient Closeness-Centrality based Multicast-aware VNE (CC-MVNE) algorithm to minimize the resource wastage as indicated in Fig. 4.1, while maximizing the sharing of substrate nodes and links. The basic idea of the proposed CC-MVNE algorithm is to reduce the number of used links to access all destinations from source node by using of the technique of closeness-centrality (CC) [40]. During the mapping process, CC-MVNE simultaneously maps the nodes and links together while minimizing the needed resources for the virtual nodes/links and multicast transmission in the network.

4.1 Closeness-Centrality based Multicast Virtual Network Embedding

Intuitively, mapping a virtual node to a substrate node with the most available computing capacity may allow more CPU sharing while selecting the shortest (or least-cost) substrate path to embed virtual link e will help in reducing the bandwidth consumption of e. However, such greedy approach may result in the resource wastage for the multicast service as shown in Fig. 4.1. In the following, similar to building a multicast tree in IP network, we consider mapping a virtual node to a substrate node that is close to nodes within the multicast group (i.e., substrate nodes with mapped virtual multicast nodes). We propose an efficient Closeness-Centrality based Multicast-aware VNE (CC-MVNE) algorithm for the aforementioned MVNE problem. Fig. 4.2 and Algorithm 1 show the flowchart and pseudocode of CC-MVNE.

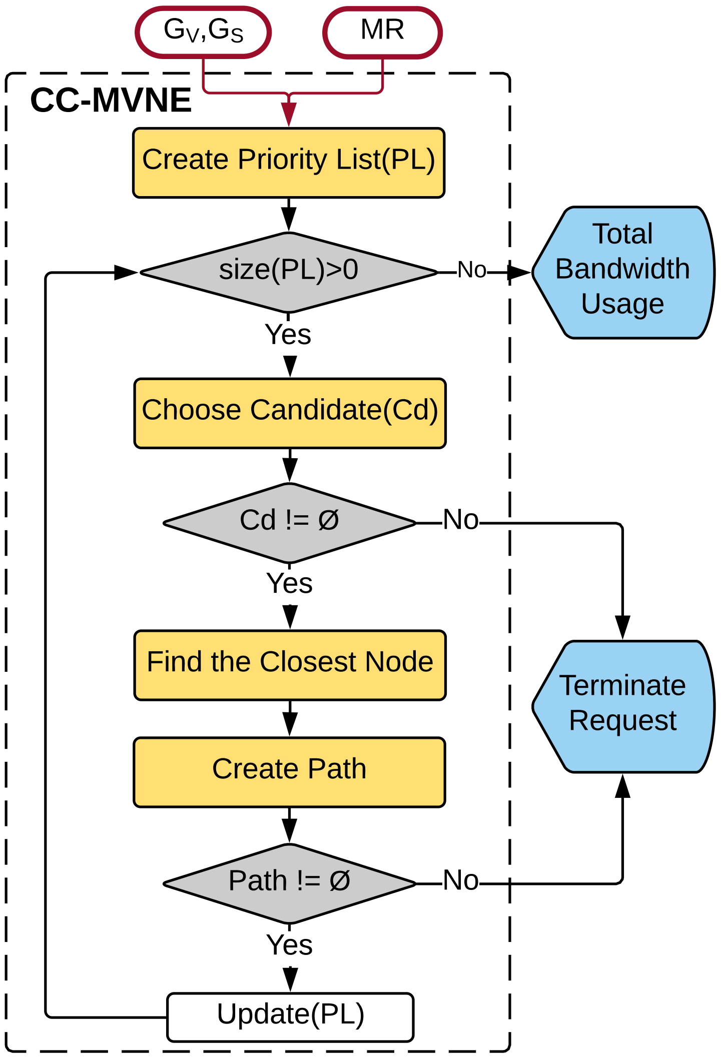


Figure (4.2) Flowchart of CC-MVNE Algorithm

As shown in Algorithm 1, the inputs of CC-MVNE algorithm are the information of Substrate Network (SN), Virtual Network (VN) and projected Multicast service Request (MR). First, the algorithm creates a priority list (PL) using the technique of Breath First Search (BFS) [41], to sort the virtual nodes in G_V in descending order based on the CPU, bandwidth demands and connected link degree.

Algorithm 1 Heuristic CC-MVNE Algorithm

```
1: procedure CC-MVNE(G$_V$,G$_S$, $MR$)
2:     PL $\leftarrow PriorityList(G_V, MR)$
3:     M$_N \leftarrow \emptyset$                    ▷ holding set of mapped nodes on the SN
4:     M$_L \leftarrow \emptyset$                    ▷ holding set of mapped links on the SN
5:
6:     while size(PL) >0 do
7:         if size(M$_N$) $\neq 0$ then
8:             Cd$_{First} \leftarrow Candidate(G_S, PL[0]) \setminus M_N$
9:             M$_N \leftarrow M_N \cup Closeness(Cd_{First}, NR(M_N))$
10:            M$_L \leftarrow M_L \cup Path(M_N, G_S)$
11:            PL $\leftarrow PL \setminus PL[0]$
12:        else
13:            Cd$_{First} \leftarrow Candidate(G_S, PL[0])$
14:            Cd$_{Second} \leftarrow Candidate(G_S, PL[1])$
15:            M$_N \leftarrow M_N \cup Closeness(Cd_{First}, Cd_{Second})$
16:            PL $\leftarrow PL \setminus PL[0]$
17:        end if
18:        Map Cd$_{First}$ to the return node of *Closeness()*
19:        Using least-cost path to map the corresponding
20:        virtual links connected to Cd$_{First}$
21:    end while
22:    return $G_S(M_N, M_L)$
23: end procedure
```

PriorityList() function is getting MS over VN as shown in Algorithm 2. In MS, we have one source node that has the highest priority in all other virtual nodes because we could not start sending data packets if the system does not map source node on a shared

SN. In the algorithm, after adding source node in priority list as the first node, the function is sorting all destination nodes in the order of priority by using of *Sorting*() function defined in Algorithm 3 by using of extended bubble sort [41]. The algorithm 3 is getting the set of vertices, and returns back the all vertices in the order of priority according to their CPU, bandwidth, and degree levels. At the beginning, all nodes (vertices) are in descending order by CPU. If the CPU levels are same for multiple nodes in the list, the function compares their bandwidth demands on VN. If it is also in same level for multiple nodes, last checking point is how many connection the virtual nodes in VN have in order to find their priority. All these three attributes are same for the nodes, then the system handles all nodes in the same order of node list.

Algorithm 2 PriorityList

```
 1: procedure PriorityList($G_V, MR$)
 2:                                                   ▷ $MR \leftarrow < s, D, \beta >$
 3:     List ← $\emptyset$
 4:     List ← $List \cup s$                          ▷ s $\in V^v$ as Virtual Source Node
 5:     D ← $Sort_{cpu,bw,deg}(D)$
 6:     List ← $List \cup D$
 7:     I ← $Sort_{cpu,bw,deg}(I)$
 8:     List ← $List \cup I$
 9:     return List
10: end procedure
```

Algorithm 3 $\text{Sort}_{cpu,bw,deg}$

```
procedure SORT_{cpu,bw,deg}(V)
    List ← $V$
    n ← $size(List)$
    if  n $\leq$ 1 then return List
    end if
    swapped← $false$
    while !swapped do
        for  $i \leftarrow 0$ to $n - 1$  do
            if CPU(List[[i] ¡ CPU(List[i+1] then
                swap(List[i],List[i+1])
                swapped ← $true$
            else if CPU(List[[i] = CPU(List[i+1] then
                if BW(list[i] ¡ BW[i+1] then
                    swap(List[i],List[i+1])
                    swapped ← $true$
                else if BW(list[i] = BW[i+1] then
                    if Deg(List[i] ¡ Deg(List[i+1]) then
                        swap(List[i],List[i+1])
                        swapped ← $true$
                    end if
                end if
            end if
        end for
    end while
end procedure
```

4.1.1 Virtual Node Mapping

Line 3-4 initializes M_N and M_L to a set of mapped substrate nodes and links, respectively. Then, Line 6-21 tries to map all virtual nodes in PL. If M_N is not empty, for the next virtual node in PL, the *Candidate*() function as shown in Algorithm 4 called in Line 8 finds all available candidate substrate nodes. With the candidate set, CC-MVNE finds the closest node to the neighbors of the virtual nodes that are already mapped with substrate nodes by using the *Closeness*() function. As shown in Algorithm 5, the closeness-centrality is computed through Eq. 4.1.

$$\sum_{u \in N_S, V \subset N_S \setminus \{u\}} 1/Dijkstra(u, V) \tag{4.1}$$

Algorithm 4 Find the Candidate Set

```
1: procedure CANDIDATE($G_S$,$v_V$)
2:     List $\leftarrow \emptyset$
3:     for $v_i^s \in N_S$ do
4:         if $CPU(v_V) \leq CPU(v_i^s)$ then
5:             if $BW(v_V) \leq BW(v_i^s)$ then
6:                 List $\leftarrow List \cup v_i^s$
7:             end if
8:         end if
9:     end for
10: end procedure
11: return List
```

Algorithm 5 Determine the Closest Node

```
1: procedure CLOSENESS($V_1, V_2$)
2:     min ← $MAX\_INT$
3:     saved ← $V_1[0]$
4:     for v ∈ $V_1$ do
5:         total ← 0
6:         for v ∈ $V_2$ do
7:             total = total + $Dijkstra(u, v)$
8:         end for
9:         if min > total then
10:            min ← total
11:            saved ← u
12:        else if min = total then
13:            if $deg(saved) < deg(u)$ then
14:                if $max(BW(saved)) < max(BW(u))$ then
15:                    min ←total
16:                    saved ←u
17:                end if
18:            end if
19:        end if
20:    end for
21:    return saved
22: end procedure
```

4.1.2 Virtual Link Mapping

In Line 10 of Algorithm 1, $Path()$ function as shown in Algorithm 6 generates the shortest paths from the closest node in Line 9 to its neighbors in M_N, which is defined as $NR(M_N)$. The shortest paths are generated by using the Dijkstra's shortest path algorithm

[42] while satisfying the requested bandwidth demand. The substrate node with the highest closeness ratio is then selected to map the current virtual node and the shortest paths are used to map the corresponding virtual links as shown in Line 18-20 of Algorithm 1. If M_N has no mapped node, the algorithm takes first and second elements of PL to find the candidate sets.

Algorithm 6 Find the Shortest Path

```
 1: procedure PATH(M_N,G_S)
 2:     vertex ← $M_N[size(M_N) - 1]$
 3:     minDistance ← $Dijkstra(M_N[0], vertex)$
 4:     saved ← $M_N[0]$
 5:     for i←1 to size(M_N)-2 do
 6:         cost = $Dijkstra(M_N[i], vertex)$
 7:         if cost < min || cost = min then
 8:             BW_1 ← $min(BW(Dijkstra(M_N[i], vertex)))$
 9:             BW_2 ← $min(BW(Dijkstra(saved, vertex)))$
10:             if BW_1 >BW_2 then
11:                 min ← cost
12:                 saved ← $M_N[i]$
13:             end if
14:         end if
15:     end for
16:     return $Dijkstra(saved, vertex)$
17: end procedure
```

4.1.3 An Example of CC-MVNE

As an example, Fig. 4.3 shows the mapped results of CC-MVNE if the input VN and SN are the ones in Fig. 4.1a, 4.1b and 4.1c. According to the requested VN in Fig 4.1a, CC-MVNE creates PL as $[S, V_2, V_1]$. The first element in PL, S will find the closest node

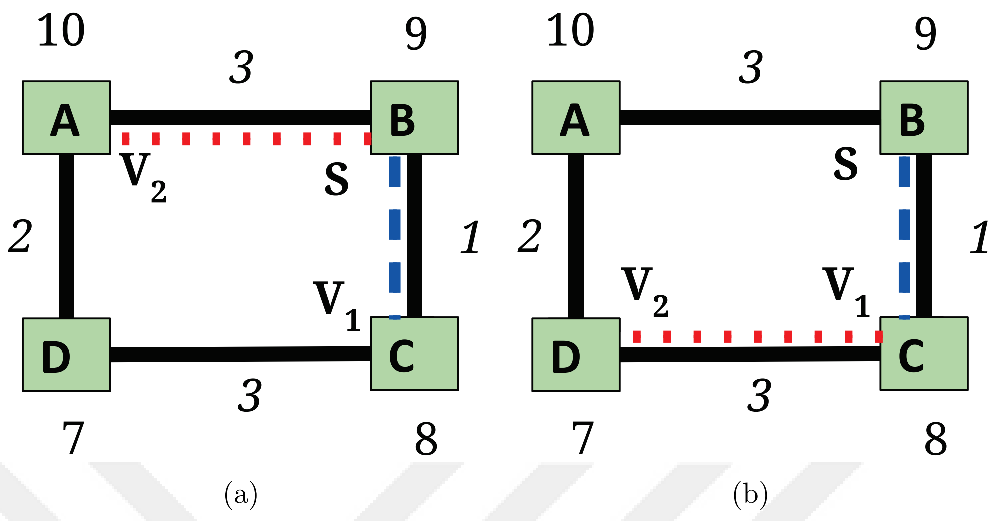
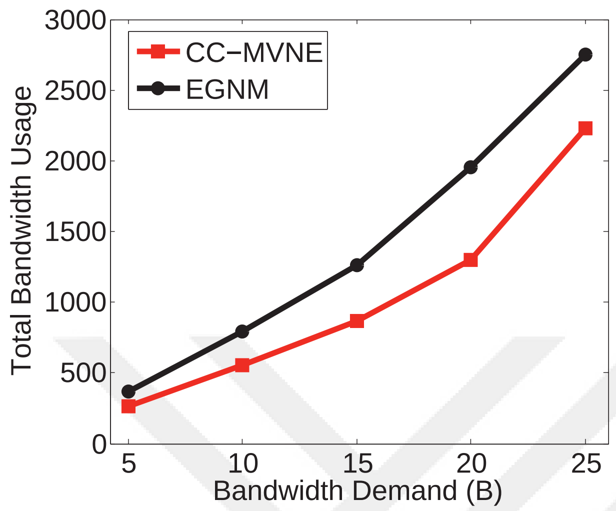


Figure (4.3) The mapping result of CC-MVNE

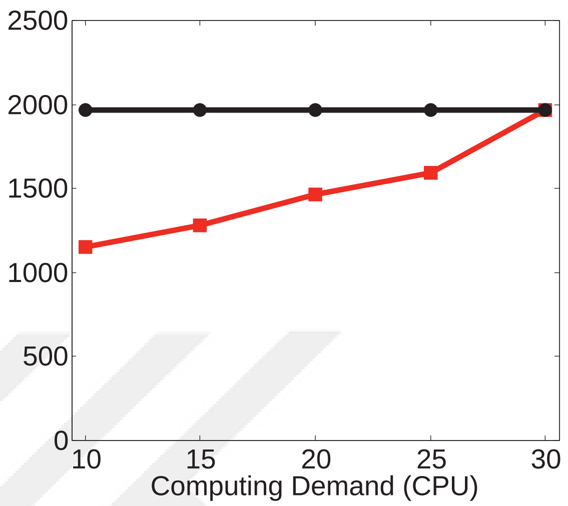
as B and add B in M_N. Next, it finds the closest node of the second element in PL, which is A. Then, the algorithm creates a path from B to A. The last node in PL is mapped onto C, and CC-MVNE generates a path from B to C. In the end, Fig. 4.1a is mapped as Fig. 4.3a by CC-MVNE. Similarly, the VN request in Fig. 4.1b is mapped as Fig.4.3b, which does not incur resource wastage for multicast services.

The computing complexity of CC-MVNE depends on the pre-defined functions. The PL creation uses BFS and sorting algorithms. BFS has an average time complexity of $\mathcal{O}(|V| + |E|)$ [41] where $|V|$ and $|E|$ are the number of vertices and edges in the graph, respectively. In CC-MVNE, the functions of $Closeness()$ and $Path()$ use the Dijkstra's shortest path algorithm. The computing complexity of Dijkstra's algorithm for vertex-based cost is $\mathcal{O}(|E|+|V|\log|V|)$ [42]. The closeness-centrality algorithm takes $\mathcal{O}(|V||E|+|V|^2\log|V|)$ computing time [40]. Hence, the CC-MVNE algorithm can take $\mathcal{O}(|V|^2|E| + |V|^3\log|V|)$ computing time.

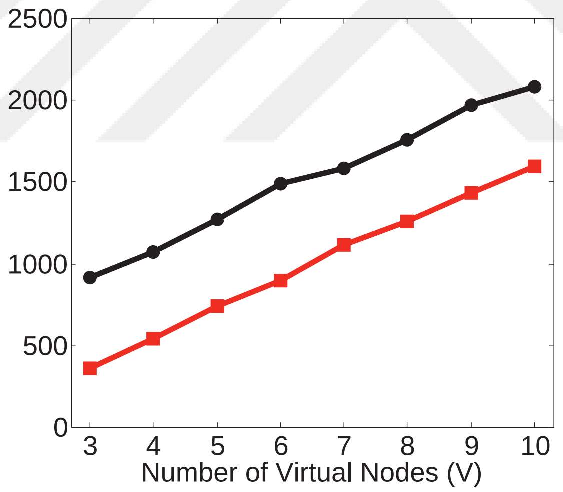
4.1.4 Experimental Results



(a) Bandwidth Usage vs B



(b) Bandwidth Usage vs CPU

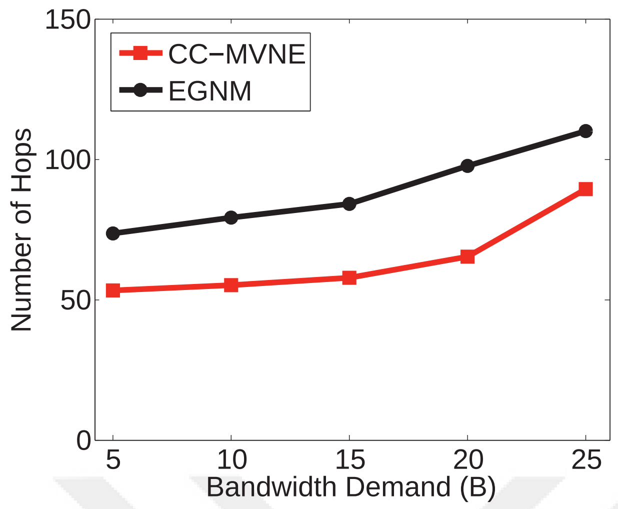


(c) Bandwidth Usage vs V

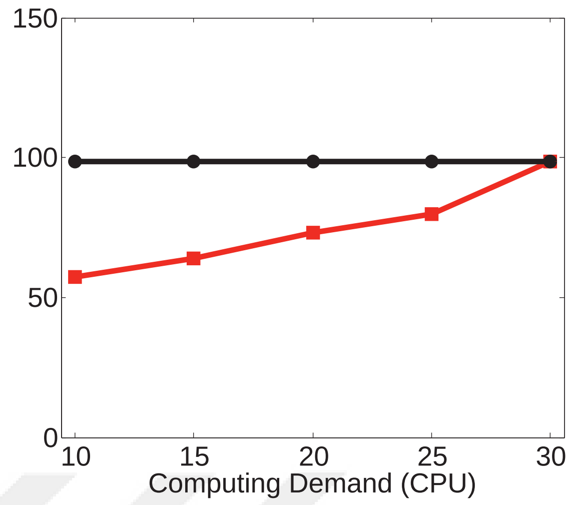
Figure (4.4) Performance results of bandwidth usage while varying B, CPU and V

We evaluate the proposed CC-MVNE algorithm, and compare it with the Enhanced Greedy Node and link Mapping (EGNM) algorithm in [38].

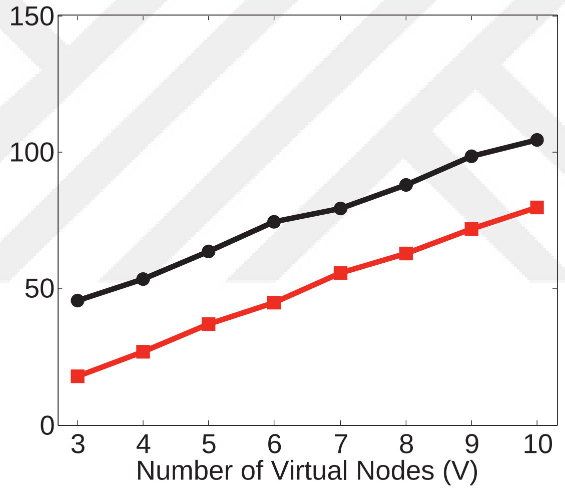
In the simulation, we use the random graph generator defined by Erdos-Renyi graph structure to create the substrate network (SN) and multiple virtual networks (VNs) [37]. We generate the SN graph with 30 nodes that are randomly connected by 90 links. As shown in Table 4.1, the computing resource of substrate nodes are generated in the range of



(a) Number of Hops vs B



(b) Number of Hops vs CPU



(c) Number of Hops vs V

Figure (4.5) Performance results of used hops while varying B, CPU and V

$[5 - CPU]$ where CPU denotes the maximum computing resource a substrate node employs. The bandwidth capacity of substrate links is in the range between 5 and B where B is the maximum bandwidth a substrate link has.

A large number of virtual networks with $[3 - 10]$ VN nodes that have random computing and bandwidth demands are generated. The computing demands of each virtual node are between 10 and CPU, and the requested bandwidth demands of a VN link is between 5 and B. The value ranges of CPU and B are listed in Table 4.1. Multiple random multicast service (MS) requests with different source and destinations are also generated. For each SN, VN and MS, we vary the requested CPU and B based on different random seeds to obtain the following average results.

Table (4.1) Network Simulation Parameters of CC-MVNE Performance

	Substrate Network	Virtual Network
Computing Resource	[5-40]	[10-30]
Bandwidth	[5-35]	[5-25]

Figs. 4.4a, 4.4b, 4.4c show the total bandwidth usage performance when varying the virtual link bandwidth, computing demand and number of virtual nodes, respectively. The performance of the total number of used hops are showed in Fig. 4.5a - 4.5c. In Fig. 4.4a, the maximum demand of computing resource is 15, and the requested bandwidth, varies from 5 to 25. As one can see that the proposed CC-MVNE outperforms the EGNM algorithm by as much as 40% when increasing B. This is because the proposed CC-MVNE can effectively use the closeness-centrality technique to map the virtual network with multicast service onto the substrate network while minimizing the bandwidth wastage(or transmission redundancies) as identified in Fig. 1. In fact, the advantages of the proposed CC-MVNE lie in two aspects. First, the proposed CC-MVNE can map the virtual network onto the substrate network with less bandwidth usage, which is further verified by the smaller number of hops in Fig. 4.5a. Second, the closeness-centrality technique enables the mapping the VN with multicast services while avoiding the bandwidth wastage(or transmission redundancies) in Fig. 1.

The total bandwidth usage increases with B in Fig. 4.4a while the number of hops for the mapping slightly increases with B as shown in Fig. 4.5a. Similarly, when increasing the requested computing demand and the number of requested virtual nodes, both the total bandwidth usage and the number of hops for virtual links from the proposed CC-MVNE increases. This is due to the fact that the increasing requested computing resource, node and the limited SN resource may force some virtual links to take longer alternative paths (i.e., more hops), resulting more bandwidth consumed by the VNs.

Interestingly, when the computing demand increases to 30 as shown in Figs. 4.4b and 4.5b, the curves from CC-MVNE get very close to (or overlap with) that from EGNM. This is because when the computing demand of the virtual nodes is large enough, there is very limited number of physical nodes, which a virtual node can map to. In other words, when there is not many options to map a virtual node, both CC-MVNE and EGNM essentially just carry out the link mapping with the shortest paths, resulting in similar performance.

Figs. 4.4c and 4.5c show that increasing the number of virtual nodes will lead to more consumed bandwidth and longer paths by both CC-MVNE and EGNM. Again, the proposed CC-MVNE outperforms the EGNM algorithm by a significant margin in terms of total bandwidth usage and total number of used hops when varying the number of virtual nodes.

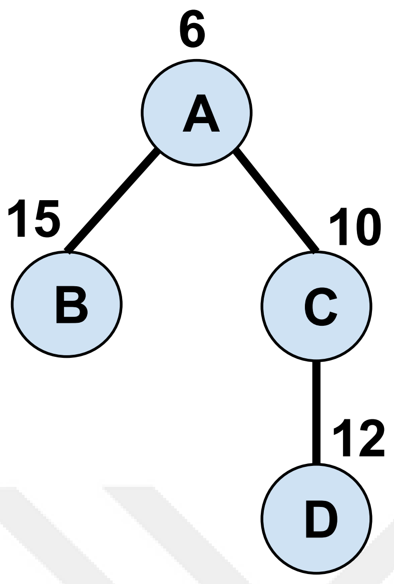
4.2 Chapter Summary

In this chapter, we have introduced the Closeness-Centrality based Multicast-Aware VNE (CC-MVNE) algorithm that minimizes the resource consumption and maximizes the sharing of the resource for mapping multicast-oriented virtual network requests. The proposed closeness-centrality technique allows the process of multicast VNE to use less bandwidth or short paths to satisfy the virtual network request while avoiding the bandwidth wastage for multicast services. Our simulation and analysis have shown that the proposed CC-MVNE outperforms the existing approach as much as by 40%.

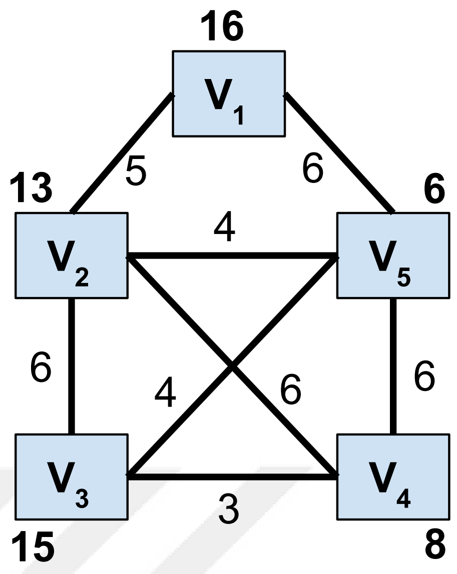
PART 5

VIRTUAL MULTICAST TREE REQUEST EMBEDDING

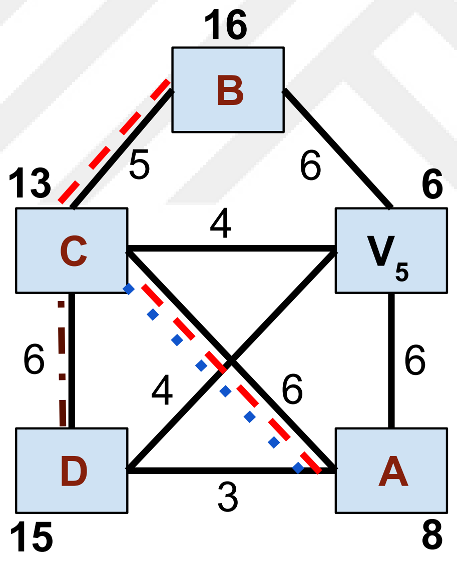
When a given virtual request in the form of multicast tree [43], we have investigated how to efficiently embed virtual multicast trees while minimizing the required resource and the redundant multicast transmissions in the substrate networks has not been investigated in the existing works. As an example shown in Fig. 5.1a, virtual node A is the source node of the multicast request. Node B, C and D are the destination nodes receiving multicast streams from node A. In this Virtual Multicast Tree (VMT) request, each node also demands a certain CPU resource as shown by the number along the node. Fig. 5.1b is the substrate network, where the number beside a substrate node indicates the available CPU resource of the node and the number on a substrate link shows the cost of the substrate link. If we assume that each substrate link has enough bandwidth and apply traditional VNE or MVNE schemes (e.g., the one in [11]), we will map the nodes according to the request/available CPU resource as shown in Fig. 5.1c. Based on the request/available CPU resource, node A, B, C and D are embedded to substrate node V_4, V_1, V_2 and V_3, respectively. The virtual link $A-B$, $A-C$ and $C-D$ are mapped to substrate path $V_4-V_2-V_1$, V_4-V_2, and V_2-V_3 by using of the traditional minimum cost strategy, respectively. Hence, when node A sends multicast traffic in Fig. 5.1a, the multicast data will be transmitted to B and C along virtual link $A-B$ and $A-C$, respectively, as shown by the red dash line and blue dot line in Fig. 5.1c. Node C then forwards the same multicast data to D along virtual link $C-D$, as shown by the brown dotdashed line. Note that, when examining the traces of the multicast data in the substrate network, the same multicast data travels on the substrate link V_4-V_2 twice (or redundantly), which leads to the resource wastage or inefficiency in the substrate network.



(a)



(b)



(c)

Figure (5.1) Redundant multicast transmissions for Multicast Tree Embedding

In [43], for the first time, we study how to design efficient algorithms to map a virtual network (in the form of virtual multicast tree) request onto the substrate network while minimizing the required resource and multicast transmissions. We define a novel problem, namely, Virtual Multicast Tree Embedding (VMTE), which is different from traditional MVNE and VNE problems. In MVNE, we are given a set of virtual nodes and need to map these nodes onto the substrate network for multicast services, which can lead to the redundant transmission as showed in Fig. 5.1. Similarly, in traditional VNE, one is given a virtual network graph and the VNE results may not be efficient in handling multicast traffic, while encountering similar inefficiencies as showed in Fig. 5.1. We note that VMTE is also different from IP multicast routing, whereas IP multicast routers are fixedly located and mapping multicast routers (or virtual nodes) is not needed. Accordingly, we propose an efficient algorithm, called Virtual Multicast Tree Embedding based on dynamic Impact Factor (VMTE-IF), to minimize the required resource and multicast transmissions. The proposed VMTE-IF algorithm maps the virtual nodes and links simultaneously to reduce the resource needed for the multicast traffic from a source to all destinations by using the Closeness-Centrality (CC) technique [40] with a dynamic local capacity metric (or impact factor).

5.1 Virtual Multicast Tree Embedding with Dynamic Impact Factor

In this section, we propose the Virtual Multicast Tree Embedding based on dynamic Impact Factor (VMTE-IF) algorithm to conduct the node and link mapping by using the technique of Closeness-Centrality (CC) and a proposed dynamic local Impact Factor (IF) as shown in Algorithm 7. Intuitively, to map a virtual multicast tree, we need to find a similar tree structure in the substrate network that yields less bandwidth and multicast transmission redundancy. Hence, the basic idea of VMTE-IF is to embed a virtual node onto a substrate node that is close to substrate candidates of multicast group members and map virtual links to the nearby mapped neighbors with minimal resource and redundant multicast transmission.

Algorithm 7 Heuristic VMTE-IF Algorithm

```
1: procedure $VMTE\text{-}IF(G_V, G_S, MR)$
2:     $\Lambda_S \leftarrow \emptyset$                                   ▷ hold mapped substrate nodes
3:     $\Lambda_V \leftarrow \emptyset$                                   ▷ hold mapped virtual nodes
4:     $\chi \leftarrow \emptyset$                                        ▷ hold all substrate paths
5:     $\zeta \leftarrow \emptyset$                                       ▷ hold all substrate candidate sets
6:     $G_S^{'} \leftarrow Prune(G_S, \beta)$
7:     for $v \in N_V$ do
8:         $\varsigma_v \leftarrow Candidate(G_S^{'}, v, \beta)$
9:         $\zeta \leftarrow \zeta + \varsigma_v$
10:    end for
11:    while $size(\Lambda_V) < size(N_V)$ do
12:        $\eta \leftarrow ImpactFactor(G_V, \Lambda_V, \zeta, \beta)$
13:        $\mu \leftarrow Closeness(G_S^{'}, \eta, \Omega(\eta), \zeta, \beta)$
14:        if $\mu \neq \emptyset$ then
15:            $\Lambda_S \leftarrow \Lambda_S \cup \mu$
16:            $\Lambda_V \leftarrow \Lambda_V \cup \eta$
17:        else
18:            $terminate$                                                ▷ block the mapping process
19:        end if
20:        if $\Lambda_V \cap \Omega(\eta) \neq \emptyset$ then
21:            $P, G_S^{'} \leftarrow ShortestPath(G_S^{'}, \Lambda_S, \Lambda_V, \Omega(\eta), \mu, \beta)$
22:            if $P \neq \emptyset$ then
23:                $X \leftarrow X + P$                                   ▷ hold mapped substrate link(s)
24:            else
25:                $terminate$                                            ▷ block the mapping process
26:            end if
27:        end if
28:        $\zeta \leftarrow \zeta - \Lambda_S$
29:    end while
30:    Hold all links to connect mapped substrate nodes($X$)
31:    Calculate the cost of link usage with $\beta$ demand
32:    return $\beta * \Sigma_{\forall p \in X}\Big(\chi_p * \omega_p\Big)$          ▷ $\forall p \in L_S$
33: end procedure
```

Finding Substrate Candidates

As shown in Algorithm 7, given a virtual network (in the form of a tree), a shared substrate network, and multicast tree request (MR) over the virtual network, VMTE-IF algorithm begins with the process of pruning SN to check the available bandwidth of each substrate links in the SN. In Line 6, the $Prune()$ function removes substrate links that cannot satisfy the bandwidth demand (β) of MR. Line 7-9 finds all substrate candidates that are satisfying the total bandwidth demand of virtual nodes with their neighbors (Ω) and save them in the set of candidates (ζ) as shown in Algorithm 8. Each virtual node has a set ($\varsigma_v, \forall v \in N_V$) of substrate candidate nodes that own enough resource.

Algorithm 8 Find the Candidate Set

```
1:  procedure CANDIDATE($G'_S, v, \beta$)
2:      $List \leftarrow \emptyset$                                   ▷ hold all candidate nodes
3:      for $n \in N'_S$ do
4:          if $c(v) \leq c(n)$ then
5:              $b_{req} \leftarrow |\Omega(v)| * \beta$
6:              $b_{res} \leftarrow \Sigma_{\forall m \in \Omega(n)} b(nm)$
7:              if $b_{req} \leq b_{res}$ then
8:                  $List \leftarrow List \cup n$
9:              end if
10:         end if
11:     end for
12:     return $List$
13: end procedure
```

5.1.1 Virtual Node Mapping

Impact Factor and Closeness-Centrality

Line 11-28 represents the embedding process of virtual nodes and virtual links. In Line 12, the $ImpactFactor()$ method dynamically calculates IF of each virtual node as ψ_v ($\forall v \in N_V$) in Eq. 5.1. The process of calculating IF takes CPU request of a virtual node over the total CPU demand of all virtual nodes as a normalized value, and bandwidth demands of each virtual node connected with the neighbors (Ω) over the total bandwidth demand in VR. The last component of Eq. 5.1 defines the ratio of maximal number of the candidate sets among all virtual nodes over the number of nodes in ς_v as shown in Algorithm 9. Line 13 identifies the substrate node that is close to all substrate candidates of its virtual neighbors with the $Closeness()$ function. In Algorithm 10, the current most important unmapped virtual node is mapped onto a substrate node that has the highest CC ratio calculated as in Eq. 5.2, where $Dijkstra(n, m)$ calculates the available shortest path between substrate node n and m. The equation finds the normalized value of the average distance from the candidate of the current most important virtual node to all candidates of its neighbors. If there is no available substrate node to host the current most important virtual node, the algorithm needs to block the mapping request. After mapping the current most important virtual node onto the closest substrate node with higher computing resource (or bandwidth) in the SN, the algorithm tries to find any mapped neighbor of the virtual node in Λ_V with Ω to collect the virtual neighbors.

Algorithm 9 Identify Next Virtual Node to Map

```
1:  procedure ImpactFactor($G_V, \Lambda_V, \zeta, \beta$)
2:      $\eta \leftarrow null$                    ▷ hold the most important virtual node
3:      $max \leftarrow 0$                        ▷ save the maximum impact factor
4:      for v $\in N_V - \Lambda_V$ do
5:          $\psi_v \leftarrow$ Eq.(5.1)          ▷ calculate the impact factor
6:          if $max \leq \psi_v$ then
7:              $max \leftarrow \psi_v$
8:              $\eta \leftarrow v$
9:          end if
10:     end for
11:     return $\eta$
12: end procedure
```

$$\psi_v = \frac{c(v)}{\Sigma_{u \in N_V} c(u)} * \frac{\Sigma_{u' \in \Omega(v)} b(vu')}{\Sigma_{e \in L_V} b(e)} * \frac{|max(\zeta)|}{|\varsigma_v|} \tag{5.1}$$

$$CC(n, V) = \Big(\frac{\sum_{m \in V \subseteq N_S} Dijkstra(n, m)}{|V|}\Big)^{-1} \tag{5.2}$$

Algorithm 10 Determine the Closest Node

```
procedure CLOSENESS($G'_S, \eta, \Omega(\eta), \zeta, \beta$)
    $max \leftarrow 0$
    $\mu \leftarrow null$
    for $n \in \varsigma_\eta$ do
        $V \leftarrow \cup_{m \in \Omega(\eta)} \varsigma_m$          ▷ find all candidates of neighbors
        $V \leftarrow V \setminus n$          ▷ remove current candidate from $V$
        $\psi_n \leftarrow CC(n, V)$          ▷ Eq. 5.10
        if $max < \psi_n$ then
            $max \leftarrow \psi_n$
            $\mu \leftarrow n$
        else if $max = \psi_n$ then
            if $c(\mu) < c(n)$ then
                $\mu \leftarrow n$
                $max \leftarrow \psi_n$
            else if $\Sigma_{m \in \Omega(\mu)} b(\mu m) < \Sigma_{m' \in \Omega(n)} b(nm')$ then
                $max \leftarrow \psi_n$
                $\mu \leftarrow n$
            end if
        end if
    end for
    if $max \neq 0$ then
        return $\mu$          ▷ return the closest node
    else
        return $\emptyset$          ▷ return empty set
    end if
end procedure
```

5.1.2 Virtual Link Mapping

After mapping a virtual node onto the substrate node, if the virtual node has the mapped neighbor(s), *ShortestPath*() function in Line 21, defined in Algorithm 11, creates the shortest substrate path(s) to all mapped virtual neighbor(s) in order to reduce the redundancy and satisfy their connections. In Line 23, X holds all mapped substrate links. Line 28 updates the set of substrate candidates of virtual nodes (ζ) by removing the mapped substrate node. In Line 32, the algorithm returns the cost of all mapping process as the total bandwidth usage of virtual links.

Algorithm 11 Find the Shortest Path

```
 1: procedure ShortestPath($G_S^{'}, \Lambda_S, \Lambda_V, \Omega(\eta), \mu, \beta$)
 2:     $\Theta \leftarrow getNeighbors(\Lambda_S, \Lambda_V \cap \Omega(\eta))$
 3:     $P \leftarrow \emptyset$
 4:     for $v \in \Theta$ do
 5:         $path \leftarrow Dijkstra(G_S^{'}, \mu, v, \beta)$
 6:         if $path \neq \emptyset$ then
 7:             $P \leftarrow P + path$
 8:             $G_S^{'} \leftarrow Update(G_S^{'}, path, \beta)$
 9:         else
10:             return $\emptyset$
11:         end if
12:     end for
13:     return $P, G_S^{'}$
14: end procedure
```

5.1.3 An Example of VMTE-IF

As an example, Fig. 5.2 represents the mapped results of VMTE-IF if the input VN and SN are the ones in Fig. 5.1a and 5.1b, respectively. According to the requested virtual multicast tree in Fig 5.1a, VMTE-IF determines IF of all virtual nodes, and begins with the most important unmapped virtual node, B. Node B is mapped onto substrate node V_3 based on the CC-ratio to the substrate candidates of virtual neighbors of B. Next, virtual node C is mapped onto substrate node V_2. Now, there is no mapped virtual neighbor of node B and C in SN. Hence, the algorithm continues selecting virtual node D to map onto substrate node V_1 while creating a path between mapped virtual neighbors from $D - C$. The last virtual node A is embedded onto substrate node V_5. Similarly, VMTE-IF creates substrate paths for $A-C$ and $A-B$. Finally, Fig. 5.1a is mapped as Fig. 5.2 by VMTE-IF. As a result, there is no redundant multicast transmissions in the substrate network when the virtual network operator sends multicast traffic through $A - C$ to $C - D$ and $A - B$.

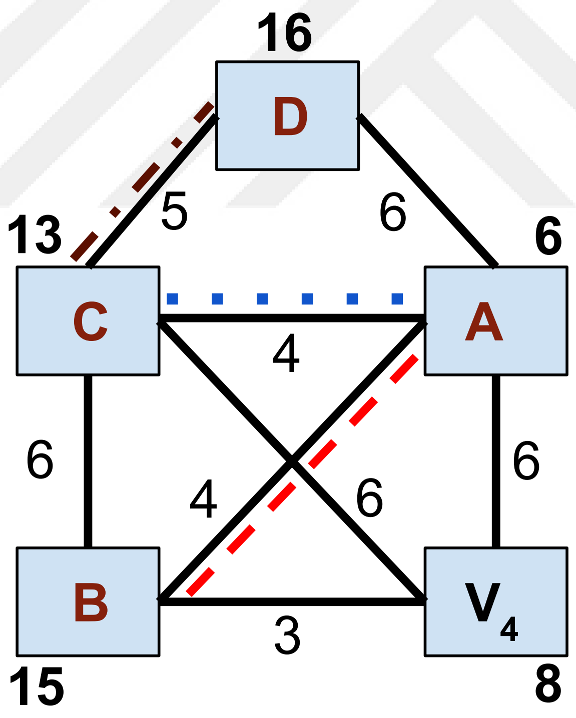
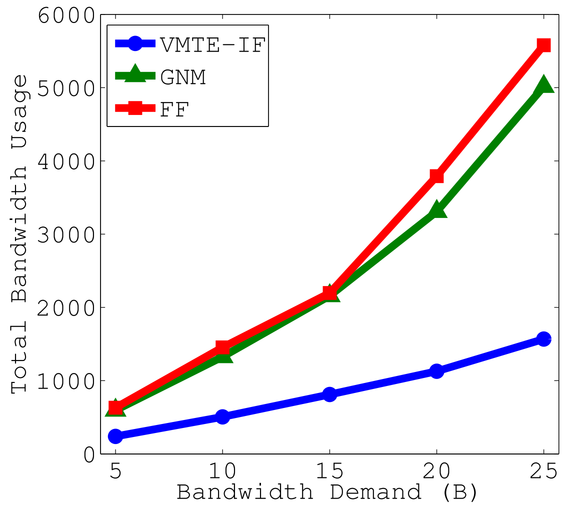
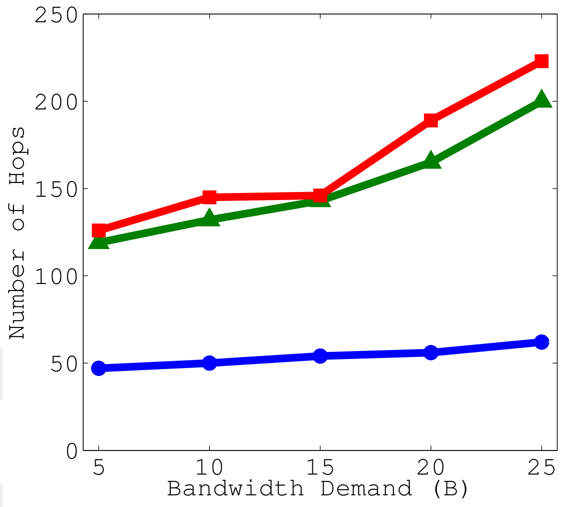


Figure (5.2) The mapping result of VMTE-IF

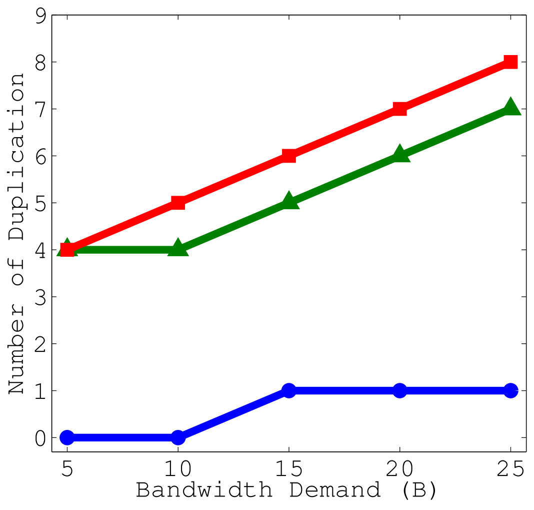
5.1.4 Experimental Results



(a) Bandwidth Usage vs B

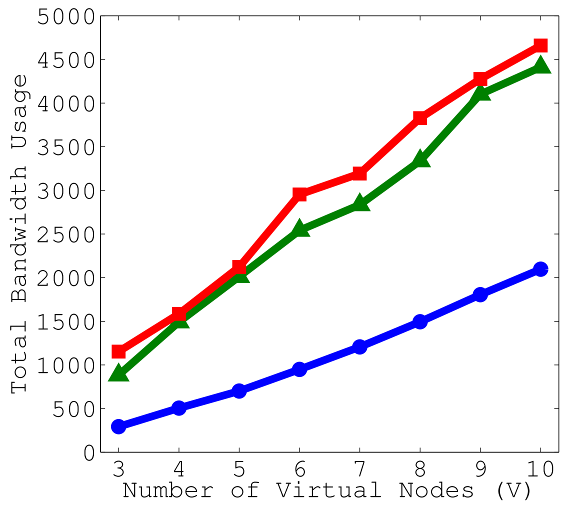


(b) Number of Hops vs B

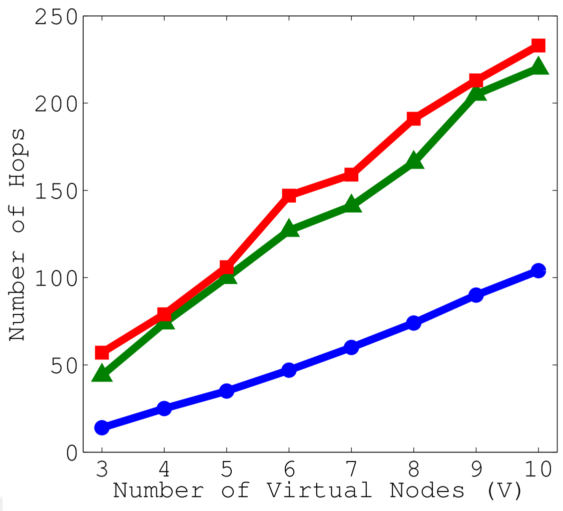


(c) Redundancy vs B

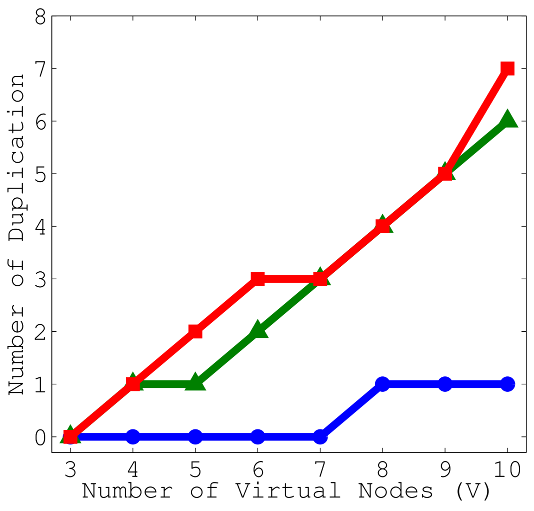
Figure (5.3) Performance results of bandwidth, hop, and redundancy while varying B



(a) Bandwidth Usage vs V



(b) Number of Hops vs V



(c) Redundancy vs V

Figure (5.4) Performance results of bandwidth, hop, and redundancy while varying V

In this section, we analyze the proposed VMTE-IF algorithm comparing with the Greedy Node Mapping (GNM) in [11] and First Fit node mapping (FF) [25].

In the simulation, we create the substrate network and virtual multicast trees by using the random graph generator defined by Erdos-Renyi [37]. The substrate network graph has 30 nodes that are randomly connected with 90 links. The computing resource of substrate nodes is randomly generated in the range of $[5 - 40]$, and the bandwidth capacity of each substrate link is in the range of $[5 - 35]$. Similarly, a large number of virtual multicast tree requests with $[3 - 10]$ VN nodes are generated. The computing resource demand of each virtual node in the request is randomly assigned in the range of $[10 - 30]$, and the bandwidth demand of the virtual multicast request is in the range of $[5 - 25]$. For each series of experiments, we vary the CPU and B based on different multicast services demand to obtain the average results, and we define link duplication as the number of SN links used by multiple VN links.

Figs. 5.3a, 5.3b and 5.3c represent the total bandwidth usage, number of hops and number of link duplication while varying the requested bandwidth demands from 5 to 25 with the maximum CPU as 15. Similarly, Figs. 5.4a - 5.4c show the total bandwidth usage, number of hops and number of link duplication while varying the number of virtual nodes from 3 to 10 with bandwidth demand as 20. In Fig. 5.3a, the proposed VMTE-IF outperforms GNM and FF algorithms at least 50% when increasing B through 15. When B is bigger than 15, the performance of our algorithm is as much as three times better than GNM and FF. This is because the proposed VMTE-IF can effectively use the CC technique with dynamic IF to simultenaously map the virtual nodes in a multicast service onto the substrate network while minimizing the required bandwidth. The fact that the proposed VMTE-IF maps the virtual multicast request onto SN with less bandwidth usage is further verified by the smaller number of hops in Fig. 5.3b, which also shows the effectiveness of VMTE-IF to minimize the redundant multicast transmission as shown in Fig. 5.3c.

In Fig. 5.4a, 5.4b and 5.4c, as one can see, the proposed VMTE-IF outperforms the GNM and FF algorithms in terms of the bandwidth usage, the total number of hops, and multicast traffic redundancy when increasing the number of virtual nodes. GNM and FF yields close results due to the similar node mapping process in both schemes. VMTE-IF is as much as two times better than GNM and FF algorithms. This is due to the fact that VMTE-IF takes advantages of the group of substrate candidates that are close each other. More specifically, the proposed VMTE-IF effectively explores the availability of substrate network and the connection structure of the substrate/virtual network to use less bandwidth for mapping while reducing the redundant transmissions. In Fig. 5.4c, when the number of virtual node is more than 7, more bandwidth resource is needed and the choices of link mapping is limited. As a result, the more substrate links are used multiple times, the larger amount of inevitable transmission redundancies by multicast traffic come from the proposed VMTE-IF.

5.2 Impact Factor Based Virtual Optical Multicast Tree Embedding

In our earlier work [43], the embedding of virtual multicast trees is studied in the IP-based SDN. However, with the employment of orthogonal frequency-division multiplexing or the technology of Elastic Optical Networks (EONs), how to efficiently map virtual optical multicast trees over EONs or Flexgrid networks while minimizing the resource usage and avoiding redundant multicast transmission in the substrate networks has not been explored in the existing studies. As an example shown in Fig. 5.5a, a Virtual Optical Request (VOR) requires multicast data transmission with 2-subcarrier from virtual node S to V_1, and from V_1 to V_2. In the VOR, each node requests a certain amount of computing resource as shown by the number along the virtual node. Fig. 5.5b is the Substrate Optical Network (SON), where the number beside a substrate node shows the available computing resource. Each column bar beside the Substrate Fiber Link (SFL) indicates the spectrum availability, where the blank square is available subcarriers while the gray one is occupied subcarrier. If we assume that each SFL has 5 subcarriers, the schemes in [12, 35] will create a virtual node list according to the descending order of the requested computing resource in Fig. 5.5a and a substrate node list based on the descending order of the available computing resource in Fig. 5.5b. As a result of mapping these two lists, virtual node S, V_1 and V_2 are embedded onto substrate node A, C and B, respectively. To satisfy the requested connections, the virtual link $S-V_1$ and V_1-V_2 are mapped to SFLs $A-C$ and $C-A-B$ using subcarriers $1-2$ and $3-4$, respectively[1]. When the user of VOR sends multicast streams from virtual node S, in Fig. 5.5a, the multicast data will be transmitted to V_1 and V_2 through virtual links $S-V_1$ and V_1-V_2. In the SON, the same multicast data goes through SFL $A-C$ as shown by the *green dashed line*, and $C-A-B$ as shown by the *red dotted line* in Fig. 5.5c. As a result, the same multicast data travels on SFL $A-C$ twice (or redundantly), which leads to resource wastage or inefficient resource usage in the SON. In addition, if $A-C$ is failed or broken in the SON, both multicast transmission over virtual link $S-V_1$ and

[1]We assume that the virtual link $S-V_1$ and V_1-V_2 can use different spectrums from the SON.

$V_1 - V_2$ will be impacted. However, for the traditional Virtual Optical Network Embedding (VONE) [29, 30, 34, 44], a given virtual network graph needs to be mapped onto the shared SON and the VONE process is oblivious to the redundant multicast transmission in Fig. 5.5.

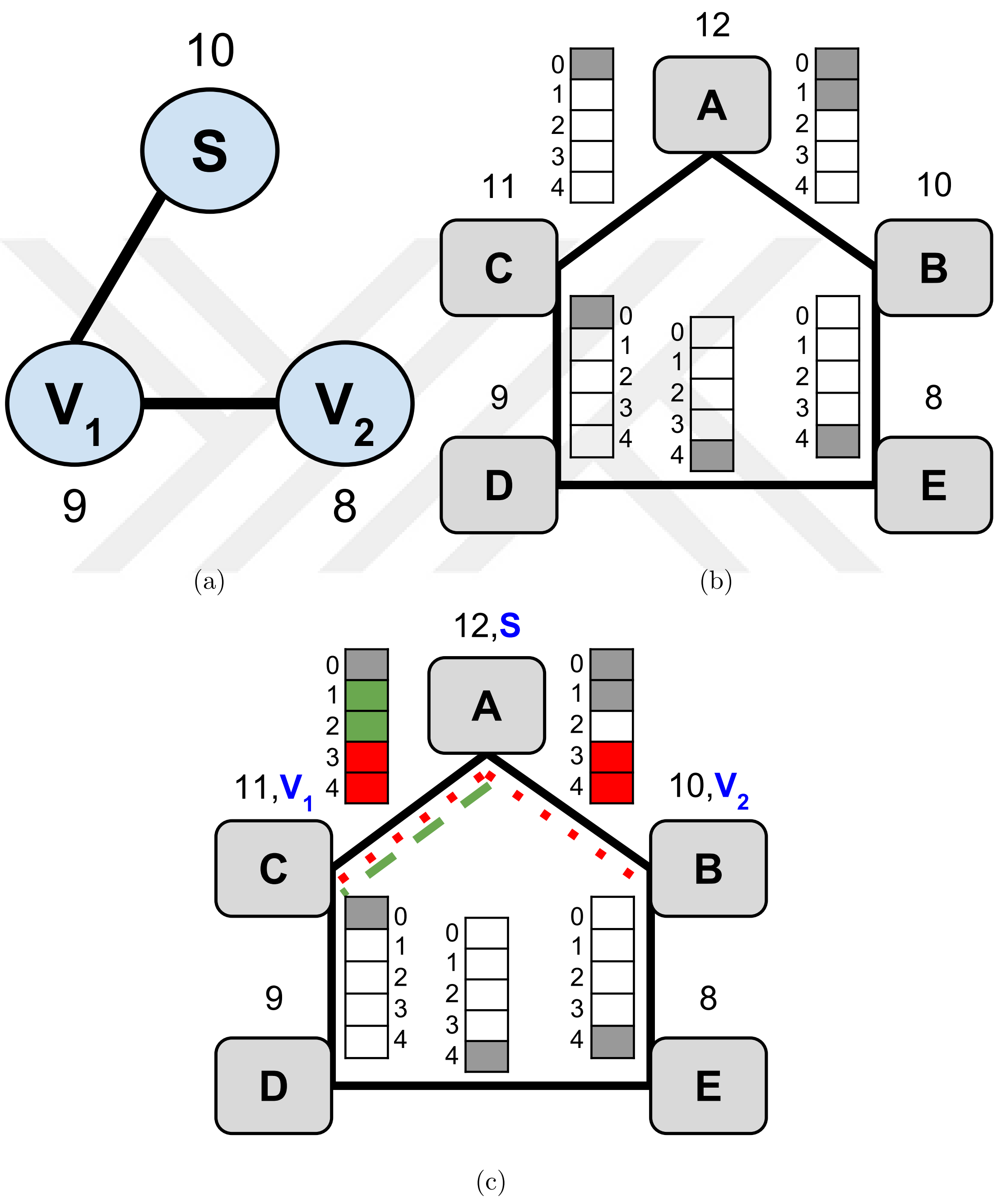


Figure (5.5) Redundant transmission from Multicast Virtual Optical Network Embedding

In [45], we study how to embed a VOR (in the form of multicast tree) onto the shared SON while minimizing the required resource usage and avoiding redundant multicast transmission. For the first time, we define the problem as Virtual Optical Multicast Tree Embedding (VOMTE), which is different from traditional MVONE problem. In MVONE, a set of virtual nodes are given without any specific virtual link connection for the mapping process. Therefore, we propose an efficient algorithm, namely, Impact Factor based Virtual Optical Multicast Tree Embedding (IF-VOMTE), to minimize the resource usage and avoid redundant multicast transmission. The proposed IF-VOMTE algorithm jointly maps virtual nodes and links by using the proposed dynamic Impact Factor (IF) and Closeness-Centrality (CC) techniques [40].

With the technology of O-OFDM based EONs, we propose the Impact Factor based Virtual Optical Multicast Tree Embedding (IF-VOMTE) algorithm to map a request of virtual optical multicast tree onto a shared Substrate Optical Network (SON) while reducing the bandwidth usage and avoiding redundant multicast transmission. To minimize bandwidth usage and avoid redundant multicast transmission, the algorithm jointly optimizes the node and link mapping processes by using the techniques of Closeness-Centrality (CC) and dynamic Impact Factor (IF) as shown in Algorithm 12. The basic idea of IF-VOMTE algorithm is to use the CC technique to map a selected virtual node by IF onto a substrate node that is close to the substrate candidates of adjacent multicast members.

Algorithm 12 IF-VOMTE Algorithm

```
1: procedure IF-VOMTE($G_S, G_V, MR$)
2:   initialize the set of mapped substrate node ($\Gamma_S$) as $\emptyset$
3:   initialize the set of mapped virtual node ($\Gamma_V$) as $\emptyset$
4:   initialize each substrate link usage($\chi$) as 0 in the list $X$
5:   prune $G_S$ by removing links without $\beta$ unused consecutive subcarriers and save it as $G_P$
6:   while the size of $\Gamma_V$ is less than the size of $N_V$ do
7:     call Candidate() to find substrate candidate sets for all virtual nodes and save in $\zeta$
8:     call ImpactFactor() to identify next virtual node $\eta$, to be embedded
9:     call Closeness() to find the candidate substrate node $\mu$ for virtual node $\eta$
10:    if $\mu$ is not $\emptyset$ then
11:      add $\mu$ in $\Gamma_S$ and add $\eta$ in $\Gamma_V$
12:    else
13:      terminate the mapping process
14:    end if
15:    if $\eta$ has any neighbor in $\Gamma_V$ then
16:      find the substrate path(s), denoted as P, to connect $\mu$ with substrate nodes mapped by
         virtual neighbor(s) of $\eta$
17:      if $P$ is not $\emptyset$ then
18:        update substrate link(s) usage in $X$ according to $P$
19:        prune and update $G_P$ by removing links without $\beta$ unused consecutive subcarriers
20:      else
21:        terminate the mapping process
22:      end if
23:    end if
24:  end while
25:  return $\beta * \Sigma_{\forall ij \in X}\left(\chi_{ij} * \omega_{ij}\right)$
26: end procedure
```

As shown in Algorithm 12, the IF-VOMTE algorithm starts with initializing the set of mapped substrate nodes (Γ_S), mapped virtual nodes (Γ_V), and the set of all SFL usage (X). In Line 5, the algorithm prunes the SON by removing substrate links that do not have β unused consecutive subcarriers and creates $G_P = (N_P, L_P)$, where N_P and L_P are the set of substrate nodes and fiber links in the pruned SON, respectively. Line 6-24 outlines the simultaneous mapping process of virtual nodes and links until embedding all virtual nodes in a VOR, which are further elaborated in the following sections.

Candidate Factor

In Line 7 of Algorithm 12, *Candidate*() method finds the set of all substrate candidate nodes for all virtual nodes. The method calculates the Candidate Factor (CF) of unmapped substrate nodes whether the computing resource, and bandwidth demand of the connected links with an unmapped virtual node can be satisfied by using Eq. (5.3). As shown in Eq. (5.4), C_S denotes the normalized value of the available computing resource of $n \in N_P$ over the computing demand of a virtual node $v \in N_V$. B_S checks whether there are enough available spectrum blocks (θ) in the SFLs from a substrate node to its neighbors as shown in Eq. (5.5). To generate the candidate sets of all virtual nodes, if CF is greater than or equal to 1, the method adds the substrate node (n) into the candidate set (ς_v) of the virtual node (v). Otherwise, n is not a candidate for v. After finding the candidate sets of all virtual nodes, *Candidate*() as defined in Algorithm 13 returns all candidate sets in the set ζ.

$$\begin{gathered} CF(v,n,B) = C_S(v,n) * B_S(v,n,\beta) \\ \forall n \in N_P, \quad \forall v \in N_V, \quad \beta \in \mathbb{Z}^+ \end{gathered} \tag{5.3}$$

$$C_S(v,n) = \left\lfloor \frac{c_S(n)}{c_V(v)} \right\rfloor \tag{5.4}$$

$$\begin{gathered} B_S(v,n,B) = \left\lfloor \frac{\sum_{m \in Neighbor_n} \sum_{\theta \in Block_{mn}} \left\lfloor \frac{size(\theta)}{\beta} \right\rfloor}{size(Neighbor_v)} \right\rfloor \\ \forall m \in N_P, \quad \forall mn \in L_P, \quad m \neq n \end{gathered} \tag{5.5}$$

Algorithm 13 Find Substrate Candidate Sets

```
1: procedure CANDIDATE($G_P, G_V, \Gamma_S, \Gamma_V, MR$)
2:     initialize $\zeta$ as $\emptyset$
3:     for each unmapped virtual node $v \in (N_V - \Gamma_V)$ do
4:         initialize $\varsigma_v$, substrate candidate set of $v$, as $\emptyset$
5:         for each unmapped substrate node $n \in (N_P - \Gamma_S)$ do
6:             if CF in Eq. (5.3) is greater than or equal to 1 then
7:                 add $n$ into $\varsigma_v$
8:             end if
9:         end for
10:        add the set $\varsigma_v$ into $\zeta$
11:    end for
12:    return $\zeta$
13: end procedure
```

5.2.1 Virtual Node Mapping

Impact Factor

ImpactFactor() method, which is defined in Algorithm 14, in Line 8 of Algorithm 12 finds the most appropriate virtual node to be embedded. The method calculates the Impact Factor (IF) of each unmapped virtual node by using Eq. (5.6). C_V as shown in Eq. (5.7) is the ratio between the computing demand of the current virtual node and the total computing demand in VOR, and B_V as shown in Eq. (5.8) is the ratio between the requested bandwidth demand by the current virtual node and the total bandwidth demand in VOR. In Eq. (5.6), Z calculates the ratio between the maximum number of candidates in ζ and the number of candidates for the current virtual node in order to increase the chance of mapping the virtual node that has less number of substrate candidate(s) as shown in Eq. (5.9). After finding IF, the method returns the virtual node (η) with the highest IF. If there are multiple virtual nodes with the same highest IF, the method returns the virtual node (η) with higher computing demand.

$$IF(v) = C_V(v) * B_V(v) * Z(v), \quad \forall v \in N_V \tag{5.6}$$

$$C_V(v) = \frac{c_V(v)}{\sum_{u \in N_V} c_V(u)} \tag{5.7}$$

$$B_V(v) = \frac{\sum_{u \in Neighbor_v} b_V(uv)}{\sum_{u'v' \in L_V} b_V(u'v')}, \qquad \forall u, u', v' \in N_V, \quad uv \in L_V, \quad u \neq v, \quad u' \neq v' \tag{5.8}$$

$$Z(v) = \frac{size(max\{\zeta\})}{size(\varsigma_v)} \tag{5.9}$$

Algorithm 14 Identify Next Virtual Node to be Embedded

```
1: procedure ImpactFactor($G_V, \Gamma_V, \zeta, MR$)
2:     initialize maximum IF as 0 and save it as temp
3:     for each unmapped virtual node $v \in (N_V - \Gamma_V)$ do
4:         calculate IF of $v$ in Eq. (5.6) and save it as $f$
5:         if $f$ is greater than temp then
6:             update current temp as $f$ and save $v$ in $\eta$
7:         else if $f$ is equal to temp and $v$ has more CPU than $\eta$ then
8:             save $v$ in $\eta$
9:         end if
10:    end for
11:    return $\eta$
12: end procedure
```

Closeness Centrality

Closeness() method in Line 9 of Algorithm 12 identifies the substrate node in ς_η that is close to all substrate candidates of η's virtual neighbors. In Algorithm 15, the method finds substrate candidates for unmapped virtual neighbors of η (set_1) and substrate nodes mapped by virtual neighbors of η (set_2) as set_0,(i.e., $set_0 = set_1 \cup set_2$). After finding the set_0, Eq. (5.10) calculates the Closeness Centrality (CC) ratio of the current substrate candidate of η. In Eq. (5.10), *Path*() function finds the shortest path from substrate node n to m with β subcarriers. After checking the CC-ratio for all substrate candidates of η, the method returns the substrate node (μ) that has the maximum value of CC-ratio. If there is no available substrate node to host η, the algorithm denies the mapping request as in Line 13 of Algorithm 12. Once mapping η onto μ in the SON, the Algorithm 12 in Line 11 updates the sets Γ_S and Γ_V.

Algorithm 15 Find the Substrate Node to Embed

```
1:  procedure CLOSENESS($G_P, \eta, \Gamma_S, \Gamma_V, \zeta, MR$)
2:    initialize maximum CC-ratio as 0 and save it as temp
3:    identify the substrate candidate set of $\eta$ ($\varsigma_\eta$) from $\zeta$
4:    for each substrate node $n \in \varsigma_\eta$ do
5:      identify the substrate candidates of unmapped virtual neighbor(s) of $\eta$ and save them
        as $set_1$
6:      identify the substrate node(s) mapped by virtual neighbor(s) of $\eta$ and save them as $set_2$
7:      find the union of $set_1$ and $set_2$ and save it as $set_0$
8:      calculate CC ratio of $n$ in Eq. (5.10) and save it as $f$
9:      if $f$ is greater than temp then
10:       save $f$ in temp and save $n$ in $\mu$
11:     else if $f$ is equal to temp and $n$ has more CPU than $\mu$ then
12:       save $n$ in $\mu$
13:     end if
14:   end for
15:   return $\mu$
16: end procedure
```

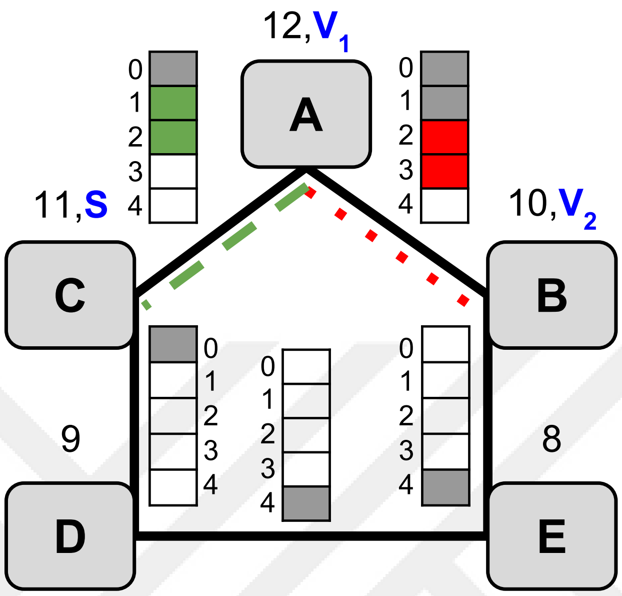
$$CC(n, set_0, B) = \Big(\frac{\sum_{m\in\{set_0-\{n\}\}\subseteq N_P} Path(n,m,B)}{size(set_0-\{n\})}\Big)^{-1} \tag{5.10}$$

5.2.2 Virtual Link Mapping

After mapping a virtual node, in Line 15 of Algorithm 12, the algorithm checks whether there is any mapped neighbor(s) of the current virtual node (η) in order to conduct the corresponding link mapping in the SON. If the virtual node has mapped neighbor(s), the Algorithm 12 in Line 16 tries to find a feasible physical path to map the virtual link between η and the mapped neighbor(s). If IF-VOMTE finds feasible path(s) between mapped virtual neighbors in the SON, the algorithm in Line 18-19 updates each substrate link usage and prunes the current SON. However, if no feasible path can be found, the request will be denied as shown in Line 21 of Algorithm 12.

In IF-VOMTE, the $Candidate()$ and $ImpactFactor()$ functions respectively have the average time complexity of $\mathcal{O}(|N|^2 \log |L|)$ and $\mathcal{O}(|N| * log|L|)$, where $|N|$ and $|L|$ are the number of nodes and links in the graph, respectively. The Closeness-Centrality (CC) function takes $\mathcal{O}(|N||L| + |N|^2 \log |N|)$ computing time [40]. The $Path()$ function uses the Dijkstra's shortest path algorithm and first fit spectrum assignment that has the computing complexity of $\mathcal{O}(|N||L| + |N|^2 \log |N| + |S||L|)$ [46], where $|S|$ is the number of spectrum segments in a SFL. Hence, the IF-VOMTE algorithm can take the average computing time of $\mathcal{O}(|N|^3|L| + |N|^3 \log |N|)$.

5.2.3 An Example of IF-VOMTE Algorithm



(a) The mapping result of IF-VOMTE

Iteration \ Virtual Node	S	V_1	V_2
1	0.31	0.42	0.15
2	0.37	-	0.15

(b) The values of IF

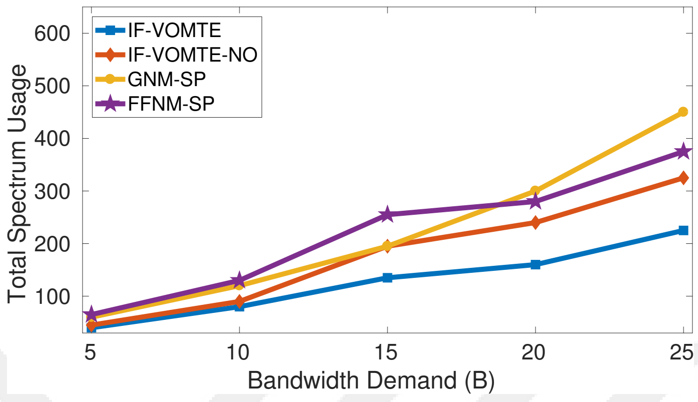
Virtual Node \ Substrate Node	A	B	C	D	E
V_1	0.67	0.67	0.67	0.67	-
S	-	1	1	-	-
V_2	-	1	-	0.5	0.5

(c) The values of CC

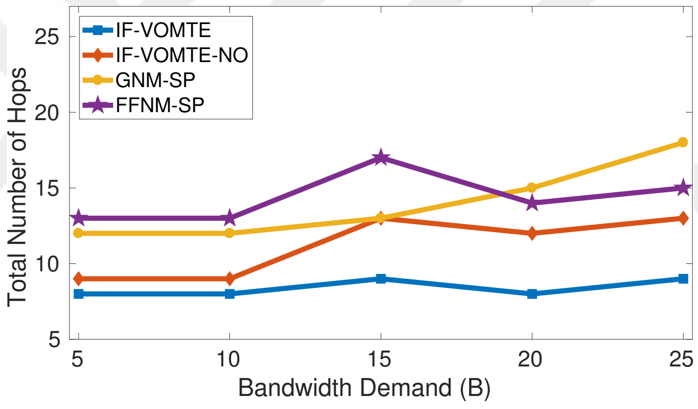
Figure (5.6) The mapping process of IF-VOMTE with $B = 2$

To illustrate how the IF-VOMTE algorithm works, Fig. 5.6a shows the result of the mapping process if the input VOR and SON are the same ones in Fig. 5.5a and 5.5b, respectively. As the requested VOR requires 2 subcarriers (i.e., $B = 2$) in Fig. 5.5a, the IF-VOMTE finds the current most appropriate virtual node (V_1) with the highest IF value at Iteration 1 in Fig. 5.6b. Since all substrate candidate nodes of V_1 have the same CC value as shown in Fig. 5.6c and the node A has more computing resource than other substrate candidates, V_1 is mapped onto the node A. Next, the unmapped virtual node that has the highest IF value at Iteration 2 is the node S, which is then mapped onto the substrate node C that has more computing resource than other substrate candidates of S. Since the mapped virtual node V_1 is the neighbor of the mapped virtual node S, the IF-VOMTE algorithm finds a physical path from substrate node C to A as shown by *the green dashed line* in Fig. 5.6a. Similarly, the IF-VOMTE maps the last unmapped virtual node V_2 onto the substrate node β that has the highest CC value as shown in Fig. 5.6c and creates the physical path between mapped virtual neighbors V_1 and V_2 as shown by *the red dotted line*. As a result, when the virtual network operator sends multicast traffic through $S - V_1$ and $V_1 - V_2$ in the VOR, there is no redundant multicast tranmission in the SON. Note that, the IF-VOMTE in Algorithm 12 cannot totally avoid the redundant multicast transmission. However, one can apply a Redundancy Checking Mechanism (RCM) and make sure $\chi_{ij} + \chi_{ji} \leq 1$ during the subprocess of virtual link mapping to avoid any redundant multicast transmission in the substrate network. When incorporating $\chi_{ij} + \chi_{ji} \leq 1$ or the RCM is employed in Algorithm 1, there is no redundant multicast transmission, which is called as the IF-VOMTE-NO.

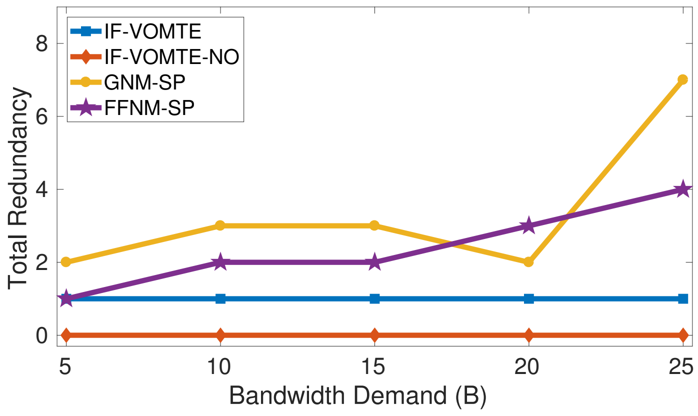
5.2.4 Experimental Results



(a) Spectrum Usage vs B

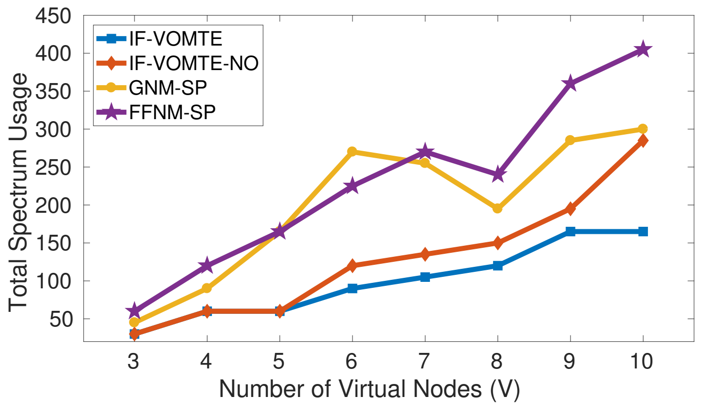


(b) Number of Hops vs B

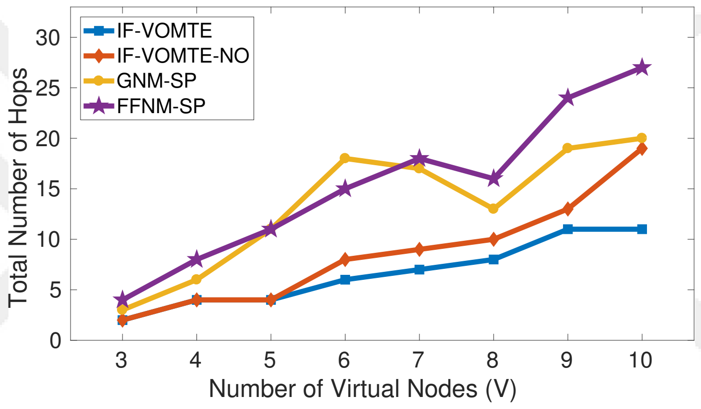


(c) Redundancy vs B

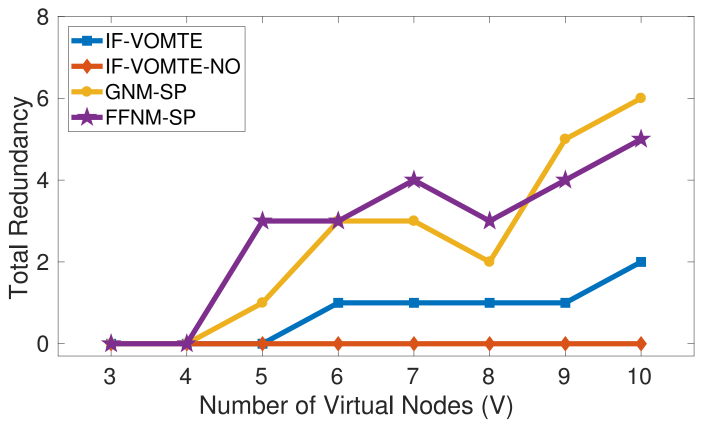
Figure (5.7) Results of spectrum and hop usage, and redundancy while varying B



(a) Spectrum Usage vs V



(b) Number of Hops vs V



(c) Redundancy vs V

Figure (5.8) Results of spectrum and hop usage, and redundancy while varying V

To study the performance of the proposed IF-VOMTE algorithm, we also implement two other heuristics: Greedy Node Mapping (GNM-SP) [12], First-Fit Node Mapping (FFNM-

SP) [25], both of which employ the shortest path strategy to map virtual links [47].

In the simulation, we use the connected graph technique of Erdos-Renyi [37] to create the substrate network with 35 nodes randomly connected by 90 links and to generate a large number of virtual multicast tree requests with $[3, 10]$ virtual nodes. Each substrate node is assigned with random available computing resource in the range of $[5 - 45]$, and each substrate fiber link has the same number of subcarrier (S= 55) with random availability. Similarly, each virtual node has random computing resource demand in the range of $[10, 30]$ and the bandwidth demand (or subcarrier) of each virtual request, B, is in the range of $[5, 25]$. In the experimental series, we obtain the average results by varying B and V the requests.

In Figs. 5.7a-5.7c and 5.8a-5.8c, we show the total spectrum usage, the number of hops, and the total redundancy when the number of requested subcarriers (or spectrums) for the request varies from 5 to 25, the maximum computing demand CPU is 30 and the number of virtual node V is 8 in Figs. 5.7a-5.7c. Similarly, Figs. 5.8a-5.8c show the results while varying the requested number of virtual nodes in a VOR from 3 to 10, setting the maximum bandwidth demand B and the maximum computing demand CPU as 25 and 30, respectively.

As shown in Figs. 5.7a and 5.8a, the proposed IF-VOMTE outperforms the GNM-SP and FFNM-SP heuristics in terms of spectrum usage, particularly when B or V is larger. This is because the proposed IF-VOMTE embeds the virtual nodes onto the SON with dynamic Impact Factor (IF) and Closeness-Centrality (CC) techniques, which effectively optimize the processes of nodes and links mapping. More specifically, the IF technique takes the resource demand by a virtual node and the connected virtual links into the consideration, which enables IF-VOMTE to give the virtual nodes with less number of mapping candidates a higher priority in the mapping process. In the mean time, the CC technique effectively identifies the substrate node that is close to substrate candidates of adjacent multicast members, to minimize the required bandwidth resource by the process of links mapping. The results in Figs. 5.7b and 5.8b further verify the effectiveness of the proposed IF and CC techniques, which can facilitate the proposed IF-VOMTE to employ the shorter physical paths by comparing with GNM-SP and FFNM-SP.

In Figs. 5.7a and 5.8a, we can also see that the IF-VOMTE performs slightly better than IF-VOMTE-NO when B and V are smaller. However, when increasing B beyond 10 or V beyond 5, the performance gap between IF-VOMTE and IF-VOMTE-NO is enlarged significantly. This is because, when B and V are smaller or less resource requested by the VOR, IF-VOMTE and IF-VOMTE-NO can find similar group of nodes to be mapped with less resource usage. However, when B and V are larger, and the subcarrier resource in the SON becomes relatively less, IF-VOMTE-NO may have to find a longer path due to the Redundancy Checking Mechanism (RCM), resulting in more spectrum usage. The higher hop numbers in Figs. 5.7b and 5.8b by IF-VOMTE-NO further verify the impacts of RCM.

In Figs. 5.7c and 5.8c, IF-VOMTE-NO has 0 multicast redundancy due to the employment of the RCM (i.e., $\chi_{ij} + \chi_{ji} \leq 1$). When increasing B or V in a VOR, multicast redundancy for FFNM-SP and GNM-SP increases. This can be explained as follows: when B or V is larger, more virtual links require more subcarrier resource from the substrate network; longer physical paths are employed as shown in Figs. 5.7b and 5.8b, resulting in more multicast redundancy by GNM-SP and FFNM-SP. Interestingly, when increasing B and V, the number of multicast redundancy from IF-VOMTE is almost flat and IF-VOMTE significantly outperforms the algorithms of GNM-SP as well as FFNM-SP. This indicates that the proposed IF and CC techniques in IF-VOMTE can intrinsically avoid multicast redundancy during the node/link mapping process. In other words, even without employing the RCM, IF-VOMTE can effectively map a multicast tree request with much less multicast redundancy due to the fact that the CC and IF techniques map proper virtual nodes onto the substrate nodes that are close to other multicast members.

5.3 Chapter Summary

In this chapter, we have defined a novel problem namely, Virtual Multicast Tree Embedding (VMTE) in IP and flexible optical networks. To heuristically solve the VMTE problem in IP and optical networks, we have proposed the virtual multicast tree embedding based on dynamic impact factor (VMTE-IF) and Impact Factor based Virtual Optical Multicast Tree Embedding (IF-VOMTE) algorithms to take advantage of the node/link closeness-centrality to minimize the required resource and redundant multicast transmission in the substrate network. Extensive simulations and analysis demonstrate that the proposed VMTE-IF algorithm outperforms the traditional approaches Greedy Node Mapping (GNM-SP) and First-Fit Node Mapping (FFNM-SP) by a very large margin.

PART 6

DEGREE-BOUNDED MULTICAST-AWARE VIRTUAL REQUEST EMBEDDING

In our earlier work [43] and [45], we have studied how to efficiently embed a given virtual multicast tree in IP or optical networks while minimizing the resource usage and avoiding the redundant multicast transmission. However, none of the aforementioned work addresses how to efficiently map virtual multicast requests over EONs or Flexgrid networks while satisfying the fanout limitation on the multicast tranmission. To illustrate the multicast transmission with the fanout limitation, we assume that a VOR requires a multicast service from node S to a set of virtual nodes $\{D_1, D_2, D_3\}$, where no virtual network topology is given a prior as shown in Fig. 6.1a. Each virtual node requests a certain amount of computing resource as shown by the number along the node in Fig. 6.1a. In the substrate network shown in Fig. 6.1b, the number beside each substrate node indicates the available computing resource. To map the multicast virtual request in Fig. 6.1a onto Fig. 6.1b, the traditional approaches in [12, 35] will create a list of virtual nodes according to the descending order of computing demands in Fig. 6.1a as well as a substrate node list based on the descending order of the available computing resource in Fig. 6.1b. As shown in Fig. 6.1c, the process of node mapping will embed the virtual nodes S, D_1, D_2, D_3 onto the substrate nodes B, E, F, C, respectively. If there is no multicast fanout limitation, the virtual link $S - D_1$, $S - D_2$, and $S - D_3$ will be created and mapped to substrate link $B - E$, $B - F$, and $B - C$ as shown in Fig. 6.1c, respectively. However, if the multicast fanout limitation of the substrate node is 2, then at most two multicast links can start from any substrate node. In other words, we cannot map 3 virtual links (i.e., $S - D_1$, $S - D_2$, $S - D_3$) simultaneously onto physical links starting from substrate node B. Instead, one may create virtual link $S - D_1$, $S - D_2$, and $D_1 - D_3$, which will be mapped to substrate path $B - E$, $B - F$, and $E - D - C$ as shown in

Fig. 6.1d, respectively. Clearly, the traditional MVONE schemes cannot be directly applied to efficiently solve the embedding of multicast services with fanout limitation.

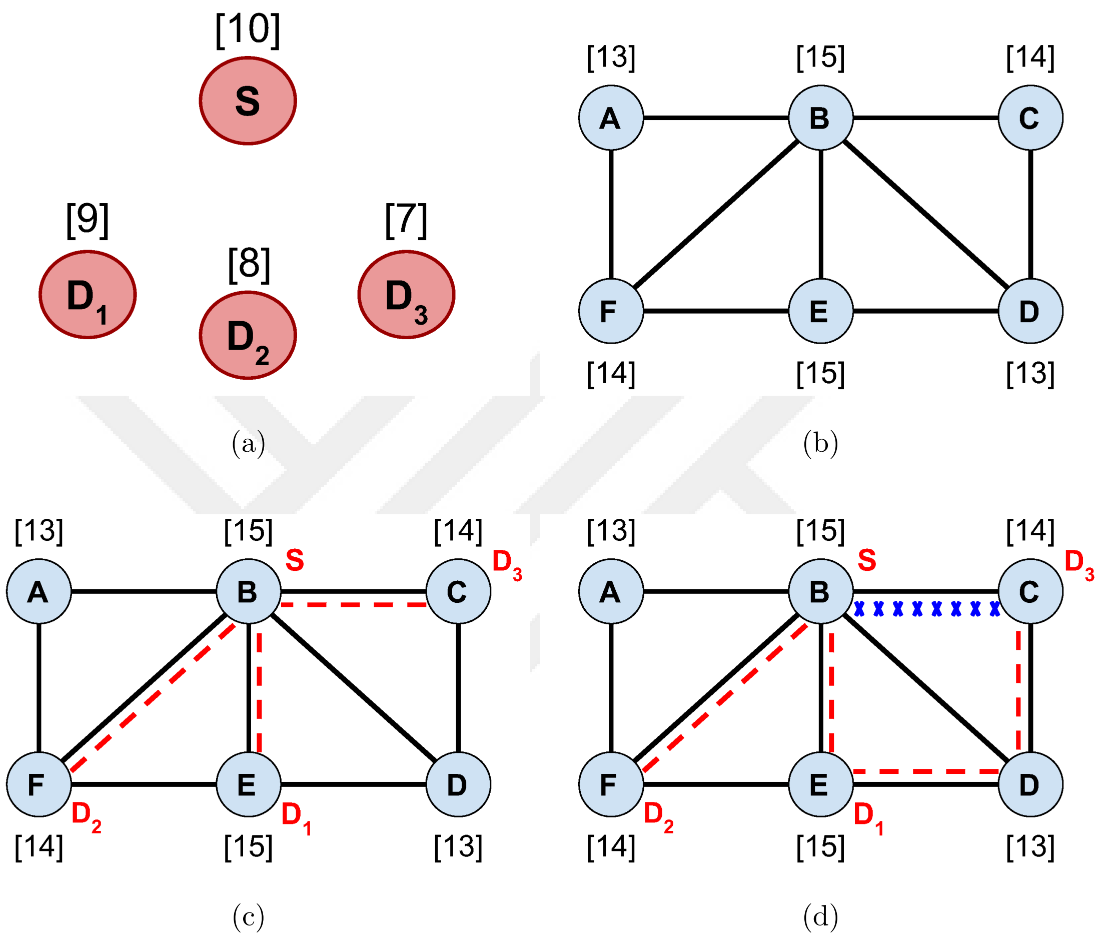


Figure (6.1) Fanout limited multicast service

In this section, for the first time, we investigate how to design efficient algorithms to map a virtual multicast request onto the substrate network while minimizing the required resource and satisfying the fanout limitation. We define a new problem called Multicast Service Embedding in EONs with Fanout Limitation (MSE-FL) [48], which is different from the problem of MVONE. We also propose an efficient algorithm, namely, Centrality-based Degree Bounded Shortest Path Tree (C-DB-SPT), to minimize the required resource of multicast transmission while satisfying the fanout limitation. The proposed C-DB-SPT maps virtual nodes and links simultaneously to reduce the needed resource by using the proposed dynamic centrality metric.

6.1 Substrate Optical Network

We model a Substrate Optical Network (SON) as a bidirectional connected graph $G_S = (N_S, L_S)$ defined as in Section 3. We use $c_S(n)$ and $f(n)$ ($\forall n \in N_S$) to represent the available computing capacity and fanout (splitting/forwarding) capacity of each substrate node, respectively. For a Substrate Fiber Link (SFL), b_{ij} and ω_{ij} ($ij \in L_S, \forall i, j \in N_S$) denote the index list of the available subcarriers and the cost or weight of the SFL, respectively.

6.2 Virtual Optical Request

To model a Virtual Optical Request (VOR), we use a 2-tuple $R = (N_V, B)$, where N_V is the set of virtual nodes, which consists of the virtual source and destination nodes of the multicasting service. In R, $B \in \mathbb{Z}^+$ denotes the bandwidth demand (i.e., the number of subcarriers) by the VOR. Without loss of generality, we consider that all virtual connections for the multicast service require the same amount of subcarriers B, while each virtual node requests a random amount of computing resource denoted as $c_V(v), (\forall v \in N_V)$.

6.3 Multicast Service Embedding in EONs with Fanout Limitation

To map a VOR onto a shared SON with given fanout limitation, the processes of node and link mapping have to be employed.

6.3.1 Node and Link Mapping

The process of node mapping is defined in Section 3, and to connect the mapped virtual nodes in the SON, physical paths consisting of SFL(s) with enough available number of subcarriers have to be identified. We use P^v_{ij} and Z_{ij} to specify whether or not the substrate link $ij \in L_S$ is chosen to reach virtual node v and the link is in the constructed multicast tree, respectively. When P_{ij}^v and Z_{ij} are equal to 1, the requested B consecutive subscarriers will be reserved from the substrate link $ij \in L_S$. Eq. (6.3) represents the flow conservation, and Eq. (6.4) indicates the distinct substrate links that form the constructed multicast tree. Eq. (6.5) represents the total traffic usage for each link in the constructed multicast tree. In Eq. (6.6) and Eq. (6.7), we ensure that the provisioned traffic does not violate the substrate link capacity, and the number connections from substrate node i to its adjacent substrate nodes at most equals to the fanout limitation, respectively.

$$P_{ij}^v = \begin{cases} 1, & \text{if substrate link } ij, (\forall ij \in L_S) \text{ is chosen} \\ & \text{to reach virtual node } v \in N_V \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

$$Z_{ij} = \begin{cases} 1, & \text{if substrate link } ij, (\forall ij \in L_S) \text{ is part} \\ & \text{of the multicast tree} \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

$$\sum_{j:(ij)\in L_S} P_{ij}^v - \sum_{j:(ji)\in L_S} P_{ji}^v = M_i^v - M_i^s, \forall i \in N_S, v \in N_V \setminus \{s\} \tag{6.3}$$

$$Z_{ij} \geq P_{ij}^v, \quad \forall v \in N_V \setminus \{s\}, (ij) \in L_S \tag{6.4}$$

$$t_{ij} = Z_{ij} * B, \quad \forall (ij) \in L_S \tag{6.5}$$

$$t_{ij} \leq b_{ij}, \quad \forall (ij) \in L_S \tag{6.6}$$

$$\sum_{ij\in L_S} Z_{ij} \leq f(i), \quad \forall i, j \in N_S \tag{6.7}$$

During the process of node and link mapping, different embedding strategies will differently map virtual nodes and use different physical paths to connect the mapped virtual nodes, resulting in various amount of resource consumption. Here, we define the cost function in Eq. (6.8), which is proportional to the total cost of the required bandwidth (or number of subcarriers) for a multicast request.

$$min(\sum_{ij \in L_S} t_{ij} * \omega_{ij}), \quad \forall i, j \in N_S \tag{6.8}$$

Definition: **Multicast Service Embedding in EONs with Fanout Limitation (MSE-FL)** Given G_S, an R and the fanout limitation, how to map a VOR onto the SON while (i) satisfying the aforementioned constraints of the node/link mapping, (ii) satisfying fanout limitation for multicast transmission in the shared SON, and (iii) minimizing the required resource.

MSE-FL consists of the subprocesses of node and link mapping with the constraints of spectrum continuity and consecutiveness as in the VONE [34], which is NP-Hard. Thus, we introduce an efficient heuristic algorithm, namely, Centrality-based Degree Bounded Shortest Path Tree (C-DB-SPT) with a dynamic centrality metric in the following part.

Table (6.1) Notations for the C-DB-SPT Algorithm

Notation	Physical Meaning
G_S	substrate network $G_S = (N_S, L_S)$
N_S	set of substrate nodes
L_S	set of substrate optical links
G_P	pruned substrate network $G_P = (N_P, L_P)$
N_P	set of substrate nodes in the pruned network
L_P	set of substrate optical links in the pruned network
R	2-tuple of virtual multicast request $R = (N_V, B)$
N_V	set of virtual nodes in a multicast request
B	bandwidth (or subcarrier) demand of a virtual multicast request
M_V	set of mapped virtual nodes
M_S	set of mapped substrate nodes
T	degree bounded multicast tree
P	physical fiber path
$ANS(T)$	set of adjacent substrate nodes of T
SC	set of all substrate candidates of unmapped virtual nodes
CV	centrality metric
NV	set of virtual nodes that can be mapped onto the closest substrate node

6.4 Centrality-based Degree Bounded Shortest Path Tree

We propose the Centrality based Degree Bounded Shortest Path Tree (C-DB-SPT) algorithm to map a request of virtual optical multicast service onto a shared substrate network while minimizing the needed resource and satisfying the fanout limitation for the multicast transmission. The proposed centrality metric allows the C-DB-SPT to map the appropriate virtual node onto a substrate node that satisfies the computing demand of the virtual node and is close to the substrate candidates of virtual multicast nodes.

Algorithm 16 C-DB-SPT Algorithm

```
1: procedure C-DB-SPT($G_S, R$)
2:     Initialize $M_V, M_S, T, ANS(T)$ as $\emptyset$;
3:     prune $G_S$ by removing links without $B$ unused consecutive subcarriers and save it as $G_P$;
4:     while size($M_V$) < size($N_V$) do
5:         update $SC$;
6:         calculate centrality values ($CV$) of all nodes in $SC$;
7:         if $T$ is $\emptyset$ then
8:             call $NodeMapping()$ and update $M_V$, $M_S$;
9:             $T \leftarrow T \cup \{n\}$;
10:            update $ANS(T)$;
11:        else
12:            if $ANS(T) \cap SC \neq \emptyset$ then
13:                $SC \leftarrow ANS(T) \cap SC$;
14:                call $NodeMapping()$ and update $M_V$, $M_S$;
15:                call $LinkMapping()$ and update $T$, $ANS(T)$;
16:            else
17:                find the substrate node $n$ from $SC$, the closest to $T$;
18:                call $NodeMapping()$ and update $M_V$, $M_S$;
19:                call $LinkMapping()$ and update $T$, $ANS(T)$;
20:            end if
21:        end if
22:        prune the nodes in $T$ that reach the fanout limitation;
23:    end while
24:    return total spectrum usage of $T$
25: end procedure
```

As shown in Algorithm 16, the C-DB-SPT algorithm has the inputs as SON (G_S) and VOR (R). In Line 2, the algorithm initializes the set of mapped virtual nodes M_V, mapped substrate nodes M_S, the degree bounded substrate multicast tree T and adjacent substrate nodes of T, denoted by $ANS(T)$ as empty. Next, the algorithm prunes the substrate network by removing the substrate links that do not have B unused consecutive subcarriers and creates $G_P = (N_P, L_P)$, where N_P and L_P are the set of substrate nodes and links in the pruned SON, respectively. Then, the C-DB-SPT conducts virtual node and link mapping until embedding all virtual nodes in Line 4-23.

For each unmapped virtual node, in Line 5 of Algorithm 16, the C-DB-SPT finds substrate candidate nodes, which satisfy the demand of unmapped virtual node, and add all substrate candidate nodes into the set SC. After creating SC, the algorithm calculates the centrality value CV for each node in SC by using Eq. (6.9), where CV identifies the average distance from the substrate node n to all other substrate nodes in SC. In Eq. (6.9), $distance()$ calculates the shortest path between substrate nodes.

$$CV(SC, n) = \frac{\sum_{m \in \{SC \setminus n\}} distance(n, m)}{|SC| - 1}, \forall n \in SC \tag{6.9}$$

In Line 7-21 of Algorithm 16, the C-DB-SPT constructs the multicast tree T. Line 7 checks whether or not T has any substrate node. If there is no substrate node in T, Line 8 calls $NodeMapping()$ method to find the substrate node n for the virtual node v as shown in Algorithm 17. The C-DB-SPT also updates the set of mapped virtual nodes M_V and substrate nodes M_S by adding v and n, respectively. As shown in Algorithm 17, the $NodeMapping()$ method, in Line 2 of Algorithm 17, finds the substrate candidate n that has the lowest CV value. In other words, Line 2 of Algorithm 17 will identify the substrate node n that is closest to other substrate candidates. After identifying the substrate node n, the method finds the unmapped virtual nodes that can be satisfied by the substrate node n and holds them into the set NV. In Line 4 of Algorithm 17, the methods selects the virtual node $v \in NV$ with the highest computing demand to map onto the substrate node n and returns the virtual node v as well as substrate node n. In Algorithm 17, $NodeMapping()$ method gives higher priority to the virtual node with the highest computing demand. This is because (i) virtual node with high computing demand has less number of substrate candidates in the SON and (ii) a VOR does not specify any given or fixed topology to map.

Algorithm 17 The Process of Node Mapping

```
1: procedure NodeMapping($SC, R, M_V$)
2:     find the substrate node ($n$) in $SC$ with the lowest $CV$;
3:     find the set of virtual nodes ($NV, NV \subseteq \{N_V \setminus M_V\}$) that can be satisfied by $n$;
4:     map the virtual node $v$ in $NV$ with the highest computing demand onto $n$;
5:     return $v$ and $n$
6: end procedure
```

In Line 9-10 of Algorithm 16, the C-DB-SPT adds the mapped substrate node into T, and adds the adjacent substrate nodes of T into the set $ANS(T)$.

In Line 11, the algorithm checks the current multicast tree T whether the multicast tree has at least one substrate node. The C-DB-SPT then employs two subprocesses: (i) $ANS(T)$ has at least one common substrate node within the set of substrate candidate nodes SC as shown in Line 13-15, and (ii) there is no common substrate node between $ANS(T)$ and SC as shown in Line 17-19.

In Line 13 of Algorithm 16, the C-DB-SPT assigns the SC as the intersection of SC with $ANS(T)$. In Line 14, the algorithm calls $NodeMapping()$ process and maps the virtual node v onto the substrate node n while updating M_V and M_S. The C-DB-SPT in Line 15 uses $LinkMapping()$ method to create the physical fiber path from the node n to the substrate node in T and to return the updated T. As shown in Algorithm 18, the $LinkMapping()$ method finds the shortest physical fiber path P from the substrate node n to the closest substrate node in T. When creating P, the method adds P into the multicast tree T in Line 3 of Algorithm 18. In Line 4, the $LinkMapping()$ method updates the degree of each substrate node in T and returns the updated T. When the C-DB-SPT receives the updated T from $LinkMapping()$, the algorithm updates the $ANS(T)$ in Line 15 of Algorithm 16.

In Line 17 of Algorithm 16, the C-DB-SPT finds the closest substrate node n from SC to the multicast tree T. If multiple substrate nodes have the same distance to T, then the

Algorithm 18 The Process of Link Mapping

```
1: procedure $LinkMapping(G_P, n, T)$
2:     find the shortest path from $n$ to any node in $T$ and save it as $P$;
3:     add the substrate fiber path $P$ into $T$;
4:     update the degree of each node in $T$;
5:     return $T$
6: end procedure
```

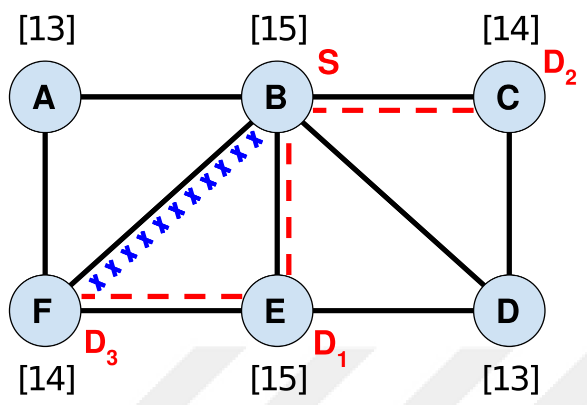
algorithm picks the substrate node with the lowest CV metric.

In Line 22, if there is a substrate node in T that reaches the fanout limitation, the algorithm marks the node from the multicast tree T as unusable for more data splitting and forwarding. In other words, the substrate node reaches the fanout limitation for the multicast service.

When the C-DB-SPT completes the process of virtual node and link mapping, the algorithm returns the total usage of number of subscarriers in Line 24.

In the C-DB-SPT, the process of finding substrate candidates and $NodeMapping()$ method has the average time complexity of $\mathcal{O}(|V| * |N|)$ and $\mathcal{O}(|N| + |V|)$, respectively, where $|V|$ and $|N|$ are the number of virtual nodes in the VOR and substrate nodes in the SON, respectively. The calculation of centrality metric takes $\mathcal{O}(|V| * |N| * \log|N| + |V| * |L|)$ computing time, where $|L|$ is the number of substrate links. The $LinkMapping()$ method uses the Dijkstra's shortest path algorithm that has the computing complexity of $\mathcal{O}(|N|^2 * \log|N| + |N| * |L|)$. Hence, the C-DB-SPT algorithm can take the average computing time of $\mathcal{O}(|N|^3 * \log|N| + |N|^2 * |L|)$.

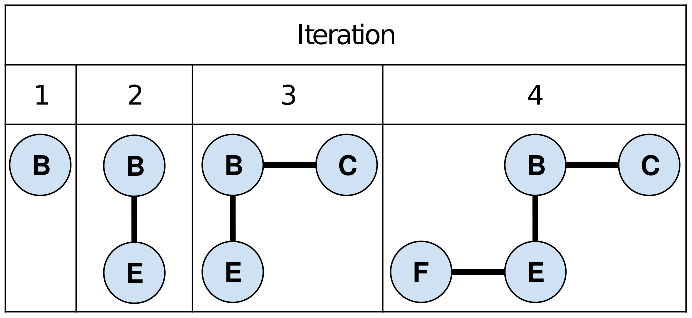
6.4.1 An Example of C-DB-SPT Algorithm



(a) The mapping result of the C-DB-SPT

	CV Values of Candidate Nodes						Sets of *SC* and *NS(T)*	
Iteration	A	B	C	D	E	F	*SC*	*NS(T)*
1	1.6	1	1.6	1.4	1.4	1.4	{A, B, C, D, E, F}	-
2	1.75	-	1.75	1.5	1.5	1.5	{A, C, D, E, F}	{A, C, D, E, F}
3	1.66	-	1.66	1.66	-	1.66	{A, C, D, F}	{A, C, D, F}
4	1.5	-	-	2	-	1.5	{A, D, F}	{A, D, F}

(b) CV Values, the sets of SC and $ANS(T)$ in the C-DB-SPT

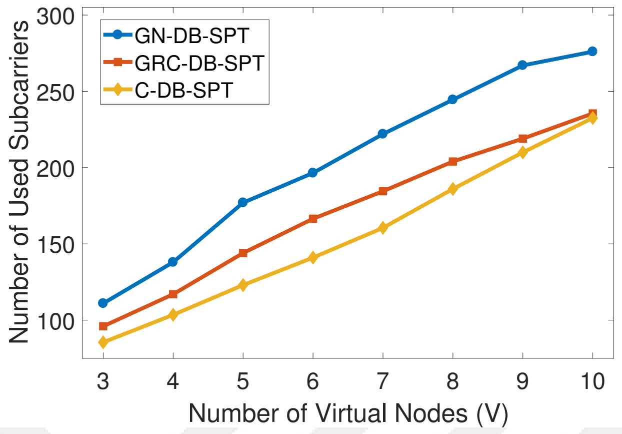


(c) Constructed degree bounded multicast tree

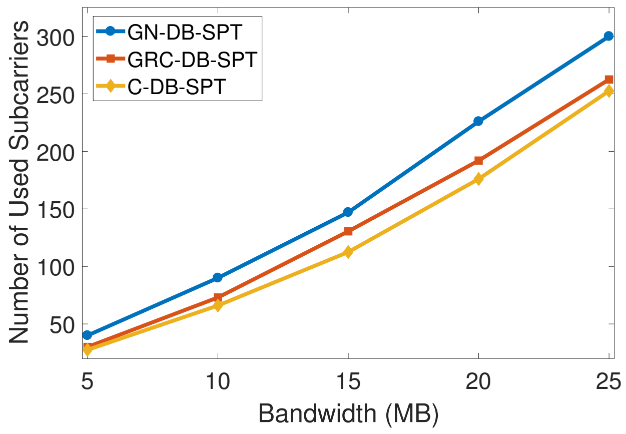
Figure (6.2) The mapping process of the C-DB-SPT

To illustrate how the C-DB-SPT algorithm works, we use Fig. 6.2a to show the mapping results when the input VOR and SON are the same ones as in Fig. 6.1. If the VOR requires 2 subcarriers (i.e., $B = 2$) and the fanout limitation equals to 2 (i.e., $F = 2$ for all substrate nodes), the C-DB-SPT finds a substrate candidate with the lowest centrality value CV to be mapped by a virtual node with the highest computing demand. In the first iteration, the C-DB-SPT identifies the substrate node B having the lowest CV and maps the virtual node S onto B while adding B into T as shown in Iteration 1 of Fig. 6.2b and 6.2c, respectively. For the next iteration, the algorithm identifies common substrate nodes between $ANS(T)$ and SC for unmapped virtual nodes as shown by Iteration 2 in Fig. 6.2b. The substrate node E is selected with higher computing capacity when multiple substrate nodes have the same lowest CV, and then the unmapped virtual node D_1 with higher computing demand is mapped onto the substrate node E. After mapping D_1 onto E, the algorithm has the substrate node B in T and then node E is connected with node B as shown by Iteration 2 in Fig. 6.2c. Therefore, the C-DB-SPT simultaneously finds a physical fiber path to connect the current mapped substrate node with the current tree such that the current mapped D_1 connects with S by using physical link $B - E$. Next, the algorithm chooses the substrate node C during Iteration 3 in Fig. 6.2b and maps virtual node D_2 onto C. The C-DB-SPT then connects D_2 with S by using physical fiber link $B - C$ while adding C into T as shown by Iteration 3 in Fig. 6.2c. For the last iteration, the C-DB-SPT maps the virtual node D_3 onto the substrate node F and connects the mapped virtual nodes by using $E - F$ instead of using the SFL $B - F$ as shown by Iteration 4 in Fig. 6.2c. This is because the substrate node B cannot split and forward data more than the fanout limitation ($F = 2$). As a result, the C-DB-SPT completes the mapping process of virtual node and link while satisfying the fanout limitation to connect the virtual nodes $S - D_1$, $S - D_2$ and $D_1 - D_3$ by using physical fiber paths $B - E$, $B - C$ and $E - F$, respectively.

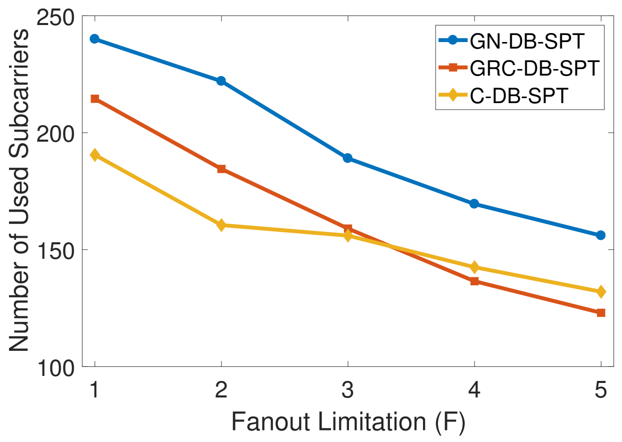
6.5 Experimental Results



(a) Spectrum Usage vs V

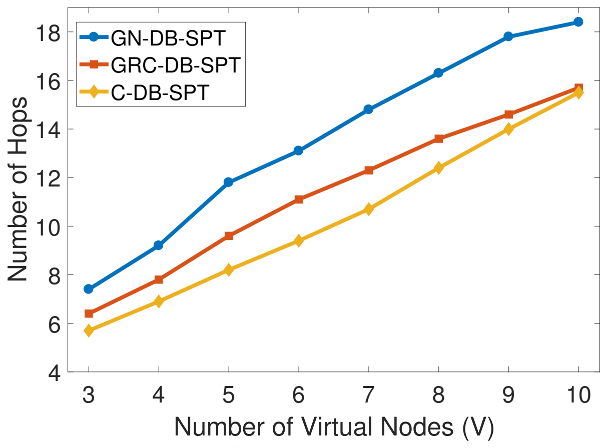


(b) Spectrum Usage vs MB

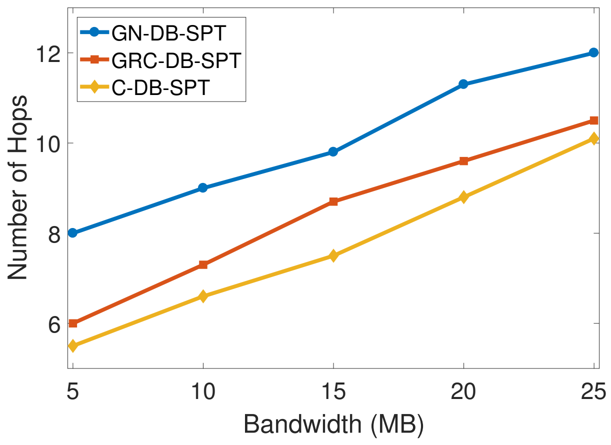


(c) Spectrum Usage vs F

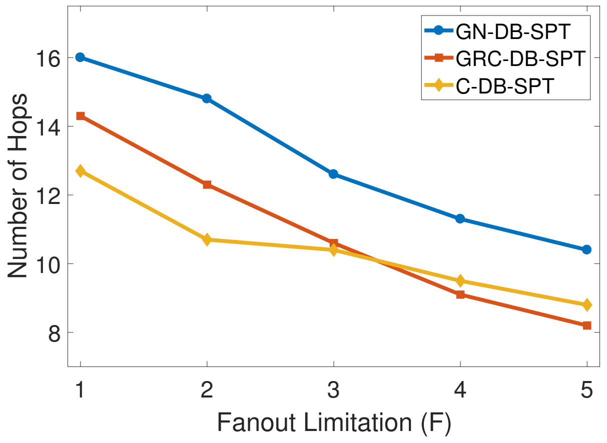
Figure (6.3) Performance results of spectrum usage while varying V, MB and F



(a) Number of Hops vs V



(b) Number of Hops vs MB



(c) Number of Hops vs F

Figure (6.4) Performance results of used hops while varying V, MB and F

In this section, we compare the proposed C-DB-SPT algorithm with the greedy node mapping and degree bounded shortest path tree (GN-DB-SPT), and the global resource capacity based degree bounded shortest path tree (GRC-DB-SPT) algorithms. The GN-DB-SPT uses the node mapping scheme proposed in [11] and the process of node mapping in the GRC-DB-SPT considers node ranking based on the availability of a substrate network proposed in [49]. The link mapping process in both GN-DB-SPT and GRC-DB-SPT is based on the technique of degree bounded shortest path tree.

In the simulation, we model the substrate network as the NSFNET with 14 nodes and 22 links. Each substrate node is assigned with random computing resource (e.g., CPU) in the range of $[5 - 35]$ and the same fanout limitation (F). Each substrate fiber link has the same number of subcarrier ($S = 55$) together with a random availability. Similarly, we create a large number of virtual multicast requests with the number of virtual nodes in the range of $[1, V]$. Each virtual node requests a random computing resource in the range of $[5, C]$. For each multicast request, the bandwidth demand (or number of subcarriers) is in the range of $[5, MB]$. For each series of experiments, we obtain the average results by varying the maximum number of virtual nodes (i.e., V) and the maximum bandwidth demand (i.e., MB) in a multicast service request, and the fanout limitation (i.e., F).

Fig. 6.3a and 6.4a show the total spectrum usage and number of hops while varying V. In Fig. 6.3a, V varies from 3 to 10 when setting $F = 2$, $C = 15$, $MB = 15$. For all three schemes, the total spectrum usage increases with V. This is due to the fact that larger V leads to a larger load of virtual multicast requests coming to the substrate network. The proposed C-DB-SPT algorithm outperforms the GN-DB-SPT and GRC-DB-SPT algorithms by as much as 35% when V is less than 9. Fig. 6.4a further verifies better performance of the C-DB-SPT while identifying shorter trees. This is because the centrality technique in the proposed C-DB-SPT can identify a better (i.e., closer) set of substrate nodes to form the virtual multicast tree. However, when $V \geq 9$, the heavy virtual multicast requests limit the option of C-DB-SPT in searching of efficient node/link mapping. As a result, the curve of C-DB-SPT gets close to (or overlapped with) that of GRC-DB-SPT. In fact, when V is large and there are not many options to map virtual nodes, both C-DB-SPT and GRC-DB-SPT essentially just carry out the link mapping with the degree bounded shortest path tree, resulting similar performance. The GN-DB-SPT uses more substrate resource than the C-DB-SPT and GRC-DB-SPT. This is because the GN-DB-SPT focuses on greedily choosing substrate nodes with higher resource availability.

Similarly, Fig. 6.3b and 6.4b show that the C-DB-SPT algorithm has better performance than the GN-DB-SPT and GRC-DB-SPT algorithms when varying MB from 5 to 25, and setting $V = 5$, $F = 2$, $C = 15$. Specifically, the C-DB-SPT uses as much as 20% fewer subcarriers than the GRC-DB-SPT and as much as 40% fewer subcarriers than the GN-DB-SPT when MB is larger than 10.

Fig. 6.3c and 6.4c show the results of performance comparison when varying F from 1 to 5 and setting $V = 7$, $C = 15$, $MB = 15$. As one can see that the proposed C-DB-SPT algorithm again outperforms the GN-DB-SPT and GRC-DB-SPT algorithms in terms of the total spectrum usage when F is less than 4. This is because the centrality technique can find closer substrate nodes for the multicast members with limited fanout capacity. Interestingly, when F is larger than 3, the GRC-DB-SPT finds slightly better results. This is due to the fact that the GRC-DB-SPT can find the group of substrate nodes that are close to each other when substrate nodes have higher flexibility (i.e., more fanouts) to connect with their neighbors, which is further verified by Fig. 6.4c.

6.6 Chapter Summary

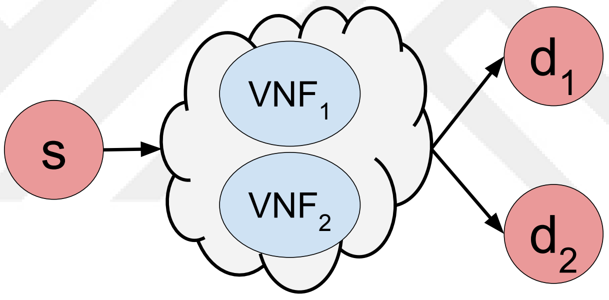
In this chapter, we have defined a new problem, which is called Multicast Services Embedding in Optical Networks with Fanout Limitation (MSE-FL). We have proposed an efficient algorithm, namely, Centrality-based Degree Bounded Shortest Path Tree (C-DB-SPT) algorithm, to take advantage of the centrality technique for multicasting node and link mapping while minimizing the needed resource and satisfying the fanout limitation. Extensive simulations and analysis have demonstrated that the proposed C-DB-SPT algorithm outperforms the schemes directly extended from the traditional approaches by as much as 35% in terms of the total spectrum usage and number of hops.

PART 7

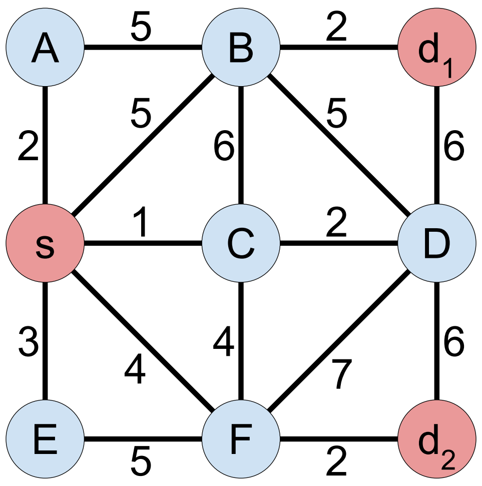
MULTICAST-AWARE SERVICE FUNCTION TREE EMBEDDING

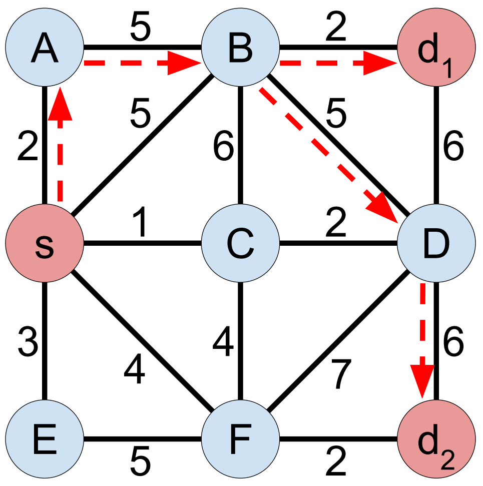
Unlike the traditional/virtual multicast studies on routing data from a source node to multiple destination nodes [43,45,48], NFV-enabled multicast requires constructing multicast transmission tree and embedding the required VNFs (e.g., Deep Packet Inspection (DPI), Network Address Translation (NAT), Intrusion Detection System (IDS), video transcoder (VT)) before reaching the destinations [50–53]. In [50], the authors introduce a dynamic heuristic method to solve centralized function deployment problem while minimizing the required resource. The authors in [51] propose a heuristic method based on path-intersection to place VNF nodes on these intersection nodes. In [52], the techniques of minimum spanning tree and backtracking are used to construct NFV-based multicast tree.

However, none of the aforementioned works addresses how to efficiently map SFC-based virtual multicast request without prior constructed SFC while satisfying the node and link constraints. To illustrate the multicast transmission with the VNF node placement, we assume that a Virtual Request (VR) requires a multicast service from source node s to a set of destination nodes $\{d_1, d_2\}$ and a set of requested service functions $\{VNF_1, VNF_2\}$ as shown in Fig. 7.1a. In this VR, the traffic has to traverse through SFC that provides all VNFs (i.e., VNF_1, VNF_2) before reaching the destination nodes. The VNF nodes (VNF_1, VNF_2) request a certain amount of computing resource and network functionality as $10, 15$ and f_1, f_2, respectively. In the substrate network shown in Fig. 7.1b, the number beside each substrate link indicates the weight (e.g., distance) of a link. The available computing capacity and offered functionality of each substrate node are listed in Table 7.1. To map the multicast virtual request in Fig. 7.1a onto Fig. 7.1b, the traditional approach (e.g., [54]) will create a list of virtual nodes according to the descending order of computing demands in Fig. 7.1a as well as a substrate node list based on the descending order of the available computing resource in Fig. 7.1b. This process will embed the virtual nodes VNF_2, VNF_1 onto the substrate nodes B, A, respectively, as shown in Fig. 7.1c. More specifically, to construct the SFC from s to d_1 and d_2, the virtual link $s - VNF_1$ and $VNF_1 - VNF_2$ will be created and mapped to substrate links $s - A$, $A - B$ as shown in Fig. 7.1c, respectively. When connecting VNF nodes with the source node s, the SFC connects the destinations via the shortest paths as shown $B - d_1$ and $B - D - d_2$ in Fig. 7.1c. The traditional SFC schemes construct the SFC to split/forward transmission through all destinations. In other words, as in Fig. 7.1c, the traffic from source node s will traverse through $VNF_1 - VNF_2$ as a single SFC to reach destination nodes d_1 and d_2. Thus, the schemes cannot be directly applied to efficiently solve the embedding of multicast services with SFC constraint.



(a) Virtual Request





(b) Substrate Network (c) Result of traditional mapping

Figure (7.1) Multicast Service Function Tree

Table (7.1) The Substrate Network

Substrate Node	A	B	C	D	E	F
Available CPU	15	20	15	10	15	10
Offered Functionality	f_1	f_2	f_2	f_1	f_3	f_1

In this dissertation, for the first time, we investigate how to design an efficient algorithm to map a virtual NFV-based multicast request onto the substrate network while minimizing the required resource and satisfying the functionality requirement. We define a new problem called Multicast-aware Service Function Tree Embedding (M-SFTE) [55], which is different from the NFV mapping problem. We propose an efficient algorithm, namely, Minimum Cost Multicast Service Function Tree (MC-MSFT), to minimize the required resource of the multicast transmission. The proposed MC-MSFT maps VNF nodes and links simultaneously to reduce the needed resource by using the proposed level-based SFC construction technique.

7.1 Substrate Network

We model a Substrate Network (SN) as a bidirectional connected graph $G = (N, L)$, where N and L are the set of substrate nodes and links, respectively. In an SN, each node has a certain computing capacity (e.g., CPU) with an offered network functionality, and has splitting/forwarding capability. Each substrate link has a bandwidth capacity and weight metric (e.g. hop or distance). We use $c_S(n)$ and $f_S(n)$ ($\forall n \in N$) to represent the available computing capacity and network function of each substrate node, respectively. For a substrate link, b_{mn} and ω_{mn} ($mn \in L, \forall m, n \in N$) denote the available bandwidth and weight of the link, respectively.

7.2 Virtual Request

To model a Service Function Chain (SFC) based Virtual Request (VR), we use a 4-tuple $R = (s, D, F, \beta)$, where s and D are the given source node and set of destination nodes, respectively. In R, F represents the set of requested Virtual Network Functions (VNFs) between the source node and each destination node, and $\beta \in \mathbb{Z}^+$ denotes the bandwidth demand of the VR. Without loss of generality, we consider that all virtual connections for the multicast service require the same amount of bandwidth β while each VNF node requests a random amount of computing resource denoted as $c_V(v), (\forall v \in F)$. For each VNF, we use $f_V(v), (\forall v \in F)$ to represent the requested functionality.

7.3 Multicast Service Function Tree Embedding (M-SFTE)

To map a VR onto a shared SN, the processes of node and link mapping have to be employed.

7.3.1 Node Mapping

We use M_n^v to denote whether a virtual node $v \in F$ is mapped onto the substrate node $n \in N$ as shown in Eq. (7.1). In Eq. (7.2), we ensure that each virtual node is mapped onto one substrate node. Similarly, Eq. (7.3) denotes that a substrate node can host at most one VNF node from the same VR. Each virtual node is embedded onto only one substrate node with enough computing resource and required functionality as in Eq. (7.4) and (7.5), respectively.

$$M_n^v = \begin{cases} 1, & \text{if } v \in F \text{ is mapped onto } n \in N \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

$$\sum_{n \in N} M_n^v = 1, \quad \forall v \in F \tag{7.2}$$

$$\sum_{v \in F} M_n^v \leq 1, \quad \forall n \in N \tag{7.3}$$

$$\sum_{v \in F} c_V(v) * M_n^v \leq c_S(n), \quad \forall n \in N \tag{7.4}$$

$$\sum_{v \in F} f_V(v) * M_n^v = f_S(n), \quad \forall n \in N \tag{7.5}$$

7.3.2 Link Mapping

To connect the mapped virtual nodes in the SN, physical paths consisting of substrate links with enough available bandwidth have to be identified. We use l_{mn}^{uv} to specify whether or not the multicast service goes through a substrate link from $m \in N$ to $n \in N$ $(mn \in L)$ for a virtual link from u to v $(u, v \in \{s\} \cup D \cup F)$. When l_{mn}^{uv} is 1, the requested β bandwidth will be reserved from the substrate link $mn \in L$.

$$l_{mn}^{uv} = \begin{cases} 1, & \text{if a substrate link from } m \text{ to } n \text{ is used} \\ & \text{for a virtual link from } u \text{ to } v \\ 0, & \text{otherwise} \end{cases} \tag{7.6}$$

During the process of node and link mapping, different embedding strategies will differently map virtual nodes and use different physical paths to connect the mapped VNF nodes, resulting in various amount of resource consumption. Here, we define the cost function in Eq. (7.7), which is proportional to the total cost of the required bandwidth for a multicast request.

$$min(\beta * \sum_{mn \in L, \forall u, v \in \{s\} \cup D \cup F} l_{mn}^{uv} * \omega_{mn}) \tag{7.7}$$

Definition: **Multicast-aware Service Function Tree Embedding (M-SFTE)** Given G and R, how to map a VR onto a shared SN while (i) satisfying the aforementioned constraints of node/link mapping, (ii) satisfying the requested SFC for multicast transmission in the shared SN, and (iii) minimizing the required resource.

Similar to SFCE, M-SFCTE consists of the subprocesses of node and link mapping under the constraints such as bandwidth consumption and computing resource. By adding some links with zero bandwidth demand, one can convert a tree into a mesh network. Accordingly, the M-SFCTE optimization can be converted to the traditional virtual network embedding optimization, which is proved to be NP-Hard [9]. Thus, we introduce an efficient heuristic algorithm for the M-SFTE, called Minimum Cost Multicast-aware Service Function Tree (MC-MSFT) in next section.

7.4 Minimum Cost Multicast Service Function Tree

We propose the Minimum Cost Multicast Service Function Tree (MC-MSFT) algorithm to map a Virtual Request (VR) onto a shared Substrate Network (SN) while minimizing the needed resource. We propose a technique of level-based path splitting to map the closest (or minimum cost) unsatisfied functionality in a source-destination pair for multicast transmission.

As shown in Algorithm 19, the MC-MSFT algorithm has the inputs as an SN (G) and a VR (R). In Line 2, the algorithm initializes the set of constructed Service Function Tree (SFT) as empty and the Service Function Chain (M_{SFC}) matrix to hold satisfied functions for source-destination pairs in SFT as 0. Next, the algorithm prunes the substrate network by removing the substrate links that do not have β available bandwidth and creates $G_P = (N_P, L_P)$, where N_P and L_P are the set of substrate nodes and links in the pruned SN, respectively. In the pruned SN (G_P), the algorithm finds the Minimum Spanning Tree (MST), which connects the source node (s) with all destination nodes (D), and updates SFT. In Line 5, the MC-MSFT algorithm creates $Path = path_{sd_1}, ..., path_{sdm}, (\forall d_i \in D, \forall path_{sd_i} \subseteq SFT, m = size(D))$ to hold all substrate paths from source node to each destination node in

Table (7.2) Notations for the MC-MSFT Algorithm

Notation	Physical Meaning
G	substrate network $G = (N, L)$
N	set of substrate nodes
L	set of substrate links
G_P	pruned substrate network $G_P = (N_P, L_P)$
N_P	set of substrate nodes in the pruned network
L_P	set of substrate links in the pruned network
R	4-tuple of virtual multicast request $R = (s, D, F, \beta)$
s	source node in a multicast request
D	set of destination nodes in a multicast request
F	set of requested VNFs for an SFC in a multicast request
β	bandwidth demand of a virtual request
SFT	substrate service function tree
M_{SFC}	matrix to hold satisfied functionality from s-D pairs
C_f	set of candidate substrates for VNF f

SFT. In Line 6 and 7, the algorithm finds all substrate candidates of Virtual Network Functions (VNFs) that have enough available computing capacity with the requested functionality as $Candidate = \{C_{f_1}, ..., C_{f_t}\}, (\forall f_i \in F, t = size(F), \forall C_{f_i} \subseteq N_P)$, and updates M_{SFC} with satisfied VNFs that are already in $Path$ for each source-destination pair as shown in Eq. (7.8). Line 8-15 outlines the VNF node and link mapping until satisfying SFCs for each source-destination pair, which are further elaborated in the following sections.

7.4.1 Level-based Path Splitting

For each unmapped VNF node, in Line 8 of Algorithm 19, the $CheckFunctionality()$ function returns a VNF (f) that has the highest FV value as shown in Eq. (7.9), where

Algorithm 19 Minimum Cost MSFT Algorithm

```
1: procedure MC-MSFT($G$, $R$)
2:     Initialize $SFT$ as empty and $M_{SFC}[D, F]$ as 0;
3:     Prune $G$ by removing links without $\beta$ available bandwidth and save it as $G_P = (N_P, L_P)$;
4:     Find the Minimum Spanning Tree (MST) between $s$ and $D$ pairs and save it as $SFT$;
5:     Find all $s$-$D$ paths in $SFT$ and save them as $Path$;
6:     Find all substrate candidate nodes in $N_P$ for each $f \in F$ and save all candidate sets as $Candidate$;
7:     Update $M_{SFC}$ for the satisfied functionality in $Path$;
8:     while $f = CheckFunctionality()$ is not empty do
9:         Initialize $min$ as MAX_INTEGER;
10:        Find destinations in $M_{SFC}$ without $f$ and save them as $DList$;
11:        Call $ReceivePathLevels()$ and save it as $Levels$;
12:        Call $SelectPairs()$ and update current $SFT$;
13:        Find all $s$-$D$ paths in $SFT$ and save them as $Path$;
14:        Update $M_{SFC}$ for the satisfied functionality in $Path$;
15:    end while
16:    Prune $SFT$ by removing duplicate functions in $Path$;
17:    return total bandwidth usage of $SFT$
18: end procedure
```

the ratio of unsatisfied VNF for all destinations over the number of substrate candidates for C_f. After selecting VNF f, the algorithm finds all destination nodes that do not have f in $path_{sd}, (\forall d \in D)$, and save them as $DList$ in Line 10. If there is no unsatisfied VNF node for all source-destination pairs (i.e., $FV = 0$ for all VNFs), the function returns empty.

$$M_{SFC}[d][f] = \begin{cases} 1, & \text{if } f \in F \text{ is in the } path_{sd} \\ 0, & \text{otherwise} \end{cases} \tag{7.8}$$

$$FV(M_{SFC}, f, C_f) = \frac{size(D) - \sum_{d \in D} M_{SFC}[d, f]}{size(C_f)} \tag{7.9}$$

In Line 11 of Algorithm 19, the technique of level-based path splitting, namely, $ReceivePathLevels()$ method, finds the levels of all source-destination paths ($path_{sd}, \forall d \in DList$). In other words, the method selects each node pairs (or link between adjacent nodes)

Algorithm 20 Returns the levels in path for a particular DList

```
1: procedure ReceivePathLevels($G_P$,$Path$,$DList$,$C_f$)
2:     Initialize $Levels$ as empty;
3:     for each node pair $p$ of $path_{sd_i}$ in $DList$ do
4:         Initialize $count$ as 1;
5:         for each node pair $p'$ of $path_{sd_j,(i<j)}$ in $DList$ do
6:             if $p$ equals to $p'$ then
7:                 Increase $count$ by 1 and remove $p'$ from $path_{sd_j}$;
8:             end if
9:         end for
10:        Call $ConnectFunction()$ and save it as $cost$;
11:        Append $count$ and $cost$ to $p$ and save $p$ in $Levels[i]$;
12:    end for
13:    return $Levels$
14: end procedure
```

starting from source to each destination in $DList$ and save them into $Levels$, which separately holds the node pair (or link) with their usage for different $path_{sd}$. In Algorithm 20, the $ReceivePathLevels()$ method chooses node pairs for each $path_{sd}$ in Line 3 by ordering their hop counts from the source node to the destination node. In Line 5-9 of Algorithm 20, the method finds the number of the node pair usage in all source-destination pairs. If the node pair is used for multiple $path_{sd}$, the method increases the count of the node pair and remove it from other source-destination pairs in Line 7 of Algorithm 20. In Line 10 of Algorithm 20, the $ConnectFunction()$ method finds the closest substrate candidate of VNF f to all node pairs (p) in the current level via the shortest path. In Line 11 of Algorithm 20, the $ReceivePathLevels()$ appends all node pairs p with their usage and the cost of VNF f connection, and save them into the $Levels$ until completing all destination nodes in $DList$.

7.4.2 Minimum Cost Pair Selection

In Line 12 of Algorithm 19, the $SelectPairs()$ method finds the minimum cost of connecting f to the current SFT. For each level of $Levels$, in Line 2-17 of Algorithm 21, the $SelectPairs()$ method finds a node pair in $Levels$ that has the minimum cost to add VNF f into SFT. In Line 7 of Algorithm 21, the method checks whether the pair is used

Algorithm 21 Returns node pair(s) with minimum cost

```
procedure SelectPairs($PList$, $Levels$, $index$)
    for each pair $p$ in $Levels[index]$ do
        Set start node of $p$ as $start$;
        Set end node of last pair in $PList$ as $end$;
        if $start$ is not equal to $end$ then
            Add $p$ into $PList$ and save its count and cost as $count$, $cost$, respectively;
            if $count = size(DList)$ and $cost < min$ then
                Assign total cost of $PList$ to $min$ and save $PList$;
            else
                Increase $index$ by 1;
                call $SelectPairs(PList, Levels, index)$;
            end if
        else
            Increase $index$ by 1;
            call $SelectPairs(PList, Levels, index)$;
        end if
    end for
    return $PList$
end procedure
```

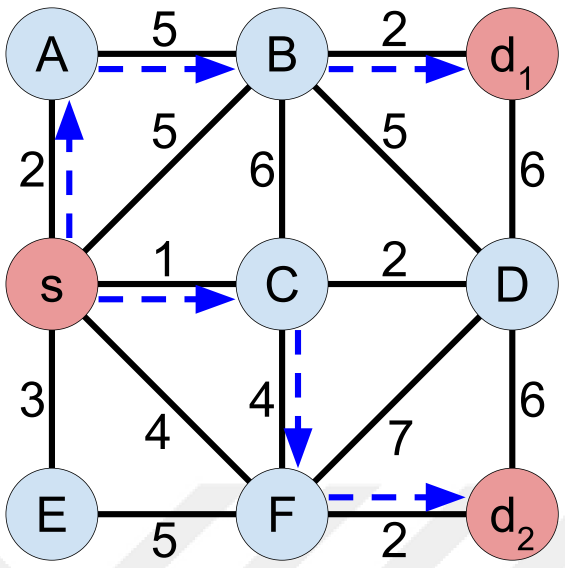
by all $path_{sd}, (d \in DList)$. In other words, if the node pair is used for all destinations ($count = size(DList)$), the method can add VNF node to that pair by satisfying all source-destination pairs. Otherwise, the method, in Line 11 and 15, recursively calls another level until satisfying all source-destination pairs, and saves the pairs with minimum addition cost as shown in Line 8 of Algorithm 21. After finding the minimum cost to add the current VNF node to the current SFT, in Line 12 of Algorithm 19, the $SelectPairs()$ method returns the selected node pairs and updates the current path list.

In Line 13 and 14 of Algorithm 19, the MC-MSFT algorithm updates all paths for source-destination pairs and M_{SFC}. After constructing SFT, in Line 16 and 17, the algorithm prunes the current SFT by removing the duplicate functions that are used multiple times in the same $path_{sd}, (\forall d \in D)$, and returns the total bandwidth usage of SFT.

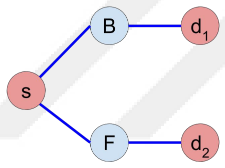
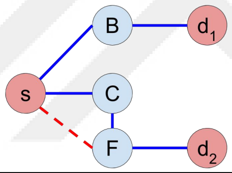
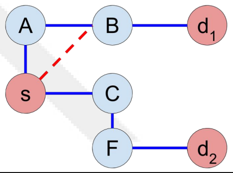
In the MC-MSFT, the process of $CheckFunctionality()$ and $ReceivePathLevels()$ have the average time complexity of $\mathcal{O}(|F| * |D|)$ and $\mathcal{O}(|D| * \log |N|)$, where $|F|$, $|D|$ and $|N|$ are the number of VNF and destination nodes in the VR, and substrate nodes in the SN, respectively. The $ConnectFunction()$ method uses the Dijkstra's shortest path algorithm that has the computing complexity of $\mathcal{O}(|N|^2 * \log|N| + |N| * |L|)$. Hence, the MC-MSFT algorithm can take the average computing time of $\mathcal{O}(|F| * |N|^2 * \log|N| + |F| * |N| * |L|)$.

7.4.3 An Example of MC-MSFT

To illustrate how the proposed MC-MSFT algorithm works, we use Fig. 7.2 to show the mapping results when the input VR and SN are the same ones as in Fig. 7.1a and 7.1b, respectively. If the VR requires two VNF nodes (i.e., $F = \{VNF_1, VNF_2\}$) and the substrate links satisfies the bandwidth demand, the MC-MSFT finds the MST in SN and the VNF node with the highest FV value to be mapped onto the closest substrate candidate of VNF node. In the first iteration, the MC-MSFT updates M_{SFC} and identifies the VNF node VNF_2 having the highest FV value for the path from s to d_2 (i.e., $path_{sd_2}$). For the second iteration, the algorithm identifies the substrate node C that is the closest substrate candidate of VNF_2 to be connected with $path_{sd_2}$ and updates the multicast tree SFT, functionality matrix M_{SFC} and the set of paths $Path$. Therefore, the algorithm simultaneously finds a physical path to connect the current mapped substrate node with the current tree such that the current mapped C connects with s and F by using physical links $s - C$ and $C - F$. In the last iteration, the algorithm chooses the VNF node VNF_1 to be mapped for s-d_1 path and maps VNF_1 onto the substrate candidate A. The MC-MSFT then connects A with the path $s - d_1$ (i.e., $path_{sd_1}$) by using physical links $s - A$ and $A - B$ while updating SFT as shown in Fig. 7.2a. As a result, the MC-MSFT completes the mapping process of VNF node and link while satisfying SFC for each $s - D$ pairs as $s - VNF_1 - VNF_2 - d_1$ and $s - VNF_2 - VNF_1 - d_2$ by using physical paths $s - A - B - d_1$ and $s - C - F - d_2$, respectively.



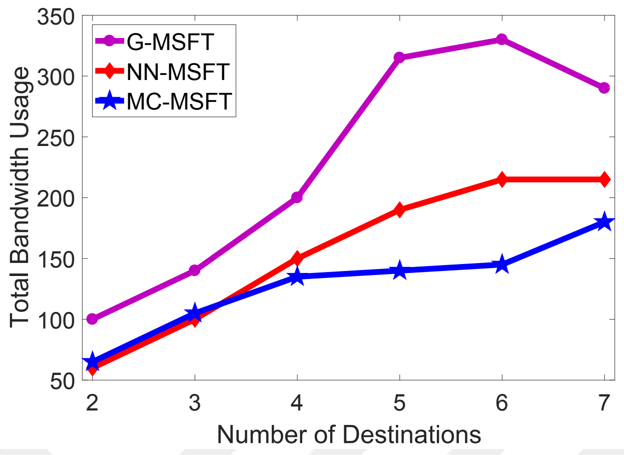
(a) The result of MC-MSFT

	Iteration 1	Iteration 2	Iteration3
SFT			
SFC	s - VNF_2 - d_1 s - VNF_1 - d_2	s - VNF_2 - d_1 s - VNF_2 - VNF_1 - d_2	s - VNF_1 - VNF_2 - d_1 s - VNF_2 - VNF_1 - d_2
Path	$path_{sd1}$ = { s-B, B-d_1 } $path_{sd2}$ = { s-F, F-d_2 }	$path_{sd1}$ = { s-B, B-d_1 } $path_{sd2}$ = { s-C, C-F, F-d_2 }	$path_{sd1}$ = { s-A, A-B, B-d_1 } $path_{sd2}$ = { s-C, C-F, F-d_2 }
M_{SFC}	$\begin{array}{ccc} D \backslash VNF & VNF_1 & VNF_2 \\ d_1 & 0 & 1 \\ d_2 & 1 & 0 \end{array}$	$\begin{array}{ccc} D \backslash VNF & VNF_1 & VNF_2 \\ d_1 & 0 & 1 \\ d_2 & 1 & 1 \end{array}$	$\begin{array}{ccc} D \backslash VNF & VNF_1 & VNF_2 \\ d_1 & 1 & 1 \\ d_2 & 1 & 1 \end{array}$
FV	VNF_1 = 0.33 VNF_2 = 0.50	VNF_1 = 0.33 VNF_2 = 0.00	VNF_1 = 0.00 VNF_2 = 0.00

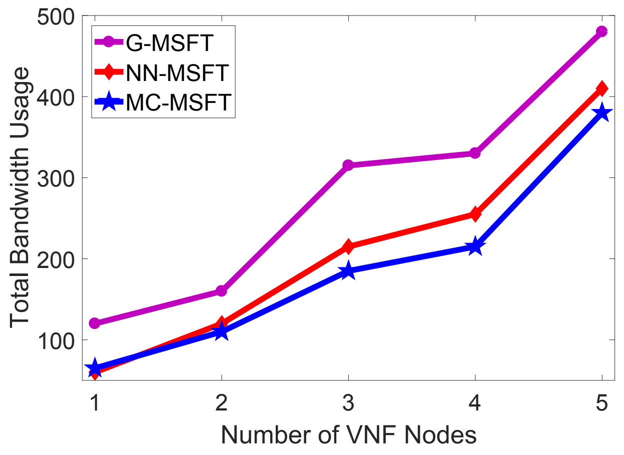
(b) SFT, SFC, $Path$, M_{SFC} and FV values in the MC-MSFT

Figure (7.2) The mapping process of MC-MSFT

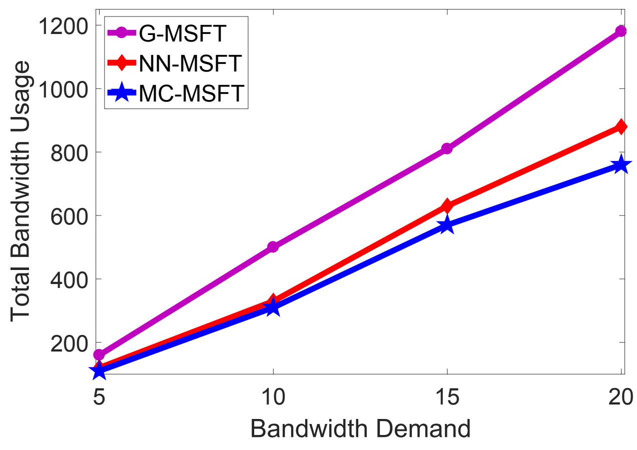
7.5 Experimental Results



(a) Bandwidth Usage vs D

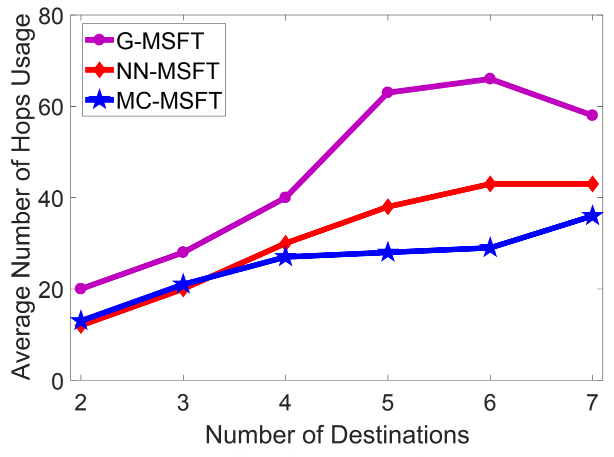


(b) Bandwidth Usage vs F

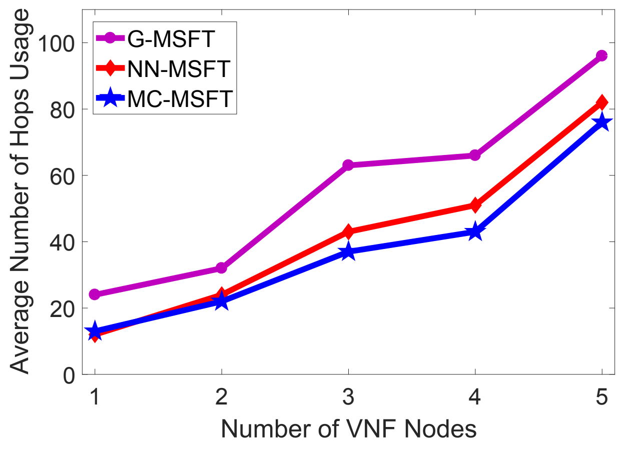


(c) Bandwidth Usage vs B

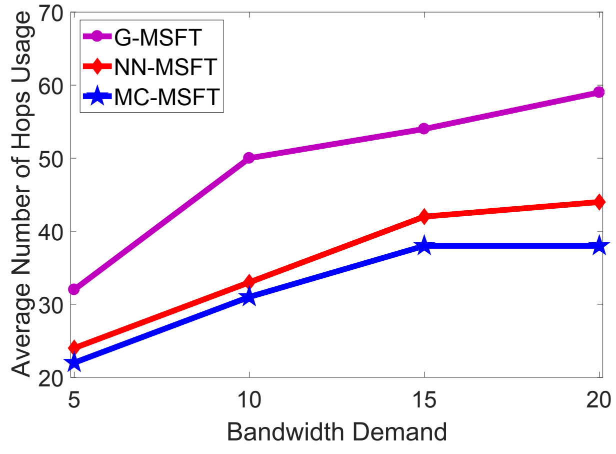
Figure (7.3) Performance results of bandwidth usage while varying D, F and B



(a) Number of Hops vs D



(b) Number of Hops vs F



(c) Number of Hops vs B

Figure (7.4) Number of hops while varying D, F and B

In this section, we compare the proposed MC-MSFT algorithm with the Greedy node mapping and Multicast Service Function Tree (G-MSFT), and the nearest neighbor MCSFT (NN-MSFT) algorithms. The G-MSFT uses the node mapping scheme proposed in [54] and the process of node mapping in the NN-MSFT considers node ranking based on the closeness of a substrate candidates to the current Service Function Chain (SFC). The link mapping process in both G-MSFT and NN-MSFT is based on the technique of shortest path SFC.

In the simulation, we model the substrate network as the USNET with 24 nodes and 43 links. Each substrate node is assigned with random computing resource (e.g., CPU) in the range of $[5 - 35]$ and a single offered functionality (f). Each substrate link has random bandwidth availability in the range of $[5, 45]$. Similarly, we create a large number of virtual multicast requests with the number of Virtual Network Function (VNF) and destination nodes in the range of $[1, V]$ and $[2, D]$, respectively. Each VNF node requests a random computing resource in the range of $[5, 25]$. For each multicast request, the bandwidth demand is in the range of $[5, B]$. For each series of experiments, we obtain the average results by varying the maximum number of VNF and destination nodes (i.e., V and D), and the maximum bandwidth demand (i.e., B) in a multicast service request.

Fig. 7.3a and 7.4a show the total bandwidth usage and number of hops while varying D. In Fig. 7.3a, D varies from 2 to 7 when setting $F = 2$ and $B = 5$. For all three schemes, the total bandwidth usage increases with D. This is due to the fact that larger D leads to a larger load of virtual multicast requests coming to the substrate network. The proposed MC-MSFT and NN-MSFT algorithms outperform the G-MSFT algorithm by as much as 35% when D is less than 4. When D is larger than 4, the MC-MSFT has better performance than the NN-MSFT algorithm. This is because the level-based path splitting technique for VNF node embedding in the proposed MC-MSFT can identify a better (i.e., closer) set of substrate nodes to form the multicast service tree. However, when $D > 6$, the heavy virtual multicast requests limit the option of MC-MSFT in searching of efficient node/link mapping. In fact, when D is large, there are not many options to map virtual nodes, both MC-MSFT and NN-MSFT essentially just carries out the link mapping with the multicast service tree.

As a result, the curve of MC-MSFT gets close to that of NN-MSFT. The G-MSFT uses more substrate resource than the MC-MSFT and NN-MSFT. This is because the G-MSFT greedily chooses substrate nodes with higher resource availability.

Similarly, Fig. 7.3b and 7.4b show that the MC-MSFT algorithm has better performance than the NN-MSFT and G-MSFT algorithms when varying F from 1 to 5, and setting $D = 4$ and $B = 5$. When $F \leq 2$, both MC-MSFT and NN-MSFT algorithms have similar performance. This is because both MC-MSFT and NN-MSFT selects closer VNF nodes to destinations. When F is larger than 2, the MC-MSFT uses as much as 20% fewer bandwidth than the NN-MSFT and much less than the G-MSFT.

Fig. 7.3c and 7.4c show the performance comparison when varying B from 5 to 20 and setting $D = 4$ and $F = 2$. Interestingly, when B is less than 10, both MC-MSFT and NN-MSFT find similar results. This is due to the fact that both MC-MSFT and NN-MSFT can find the group of substrate nodes that are close to each other when substrate nodes have higher flexibility (i.e., more available links) to connect with their neighbors, which is further verified by Fig. 7.4c. As one can see that the proposed MC-MSFT algorithm again outperforms the NN-MSFT and G-MSFT algorithms in terms of the total bandwidth usage. This is because the technique of level-based path splitting can find closer substrate nodes for the multicast request.

7.6 Chapter Summary

In this chapter, we have defined a new problem of Multicast-aware Service Function Tree Embedding (M-SFTE). We have proposed an efficient Minimum Cost Multicast Service Function Tree (MC-MSFT) algorithm. The MC-MSFT algorithm takes advantage of the proposed level-based path splitting technique to map the closest (or minimum cost) missed Virtual Network Function (VNF) nodes to minimize the needed resource and construct the Service Function Chain (SFC) for each source-destination pair in the multicast request. Extensive simulations and analysis have demonstrated that the proposed MC-MSFT algorithm outperforms the schemes directly extended from the traditional approaches by as much as 35% in terms of the total bandwidth usage.

PART 8

CONCLUSION

Software-Defined Networking (SDN) decouples the network control and forwarding functions to enable the programmable network control such that the underlying infrastructure can offer flexible network services. SDN offers an architecture that is cost-effective, agile and adaptable for today's emerging high-bandwidth applications. With network virtualization in SDN, the networking connectivity and services can be abstracted and decoupled from the underlying infrastructure. As a result, multiple tenants can share the same physical infrastructure to advance the utilization of physical resource and minimize the service CAPEX/OPEX. The process of Virtual Network Embedding (VNE) deals with how to map Virtual Network (VN) requests onto a shared Substrate Network (SN) with limited resource.

The traditional VNE includes two important mapping processes: node mapping and link mapping. For the node mapping, each virtual node that requires some specific computing resource (such as CPU) needs to be mapped onto a substrate node with enough computing resource. The link mapping will find an appropriate substrate path and reserve enough bandwidth along the path in the substrate network for each virtual link. The VNE optimization problem is known as NP-hard [9]. Hence, many researchers focus on efficient heuristic and meta-heuristic approaches to solve the VNE problem.

Recently, with emerging cloud applications and services unlike the unicast (point-to-point) tranmission, Internet applications (e.g. live stock quotes, IPTV, video-conferencing) require the same data packet to be transmitted through the group of destinations. We have studied how to efficiently embed a given virtual multicast tree in IP or optical networks while minimizing the resource usage and avoiding the redundant multicast transmission, and map virtual multicast requests over EONs or Flexgrid networks while satisfying the fanout limitation on the multicast tranmission.

In this dissertation, we have studied how to design efficient algorithms to map a virtual network (in the form of virtual multicast network/tree) request onto the substrate network while minimizing the required resource and multicast transmissions. We define a novel problem, namely, Virtual Multicast Tree Embedding (VMTE), which is different from traditional MVNE and VNE problems. In MVNE, we are given a set of virtual nodes and need to map these nodes onto the substrate network for multicast services, which can lead to the redundant transmission. Similarly, in traditional VNE, one is given a virtual network graph and the VNE results may not be efficient in handling multicast traffic, while encountering similar inefficiencies. We note that VMTE is also different from IP multicast routing, whereas IP multicast routers are fixedly located and mapping multicast routers (or virtual nodes) is not needed. Accordingly, we propose efficient algorithms, called Closeness-Centrality based Multicast-aware VNE (CC-MVNE), Virtual Multicast Tree Embedding based on dynamic Impact Factor (VMTE-IF), and Impact Factor based Virtual Optical Multicast Tree Embedding (IF-VOMTE) algorithms, to minimize the required resource and multicast transmissions in IP and flexible optical networks.

Furthermore, we have defined new problems, which are called Multicast Services Embedding in Optical Networks with Fanout Limitation (MSE-FL) and Multicast-aware Service Function Tree Embedding (M-SFTE). To solve these problems, we have proposed efficient algorithms, namely, Centrality-based Degree Bounded Shortest Path Tree (C-DB-SPT) algorithm and Minimum Cost Multicast Service Function Tree (MC-MSFT) algorithm. The Centrality-based Degree Bounded Shortest Path Tree (C-DB-SPT) algorithm takes advantage of the centrality technique for multicasting node and link mapping while minimizing the needed resource and satisfying the fanout limitation. The Minimum Cost Multicast Service Function Tree (MC-MSFT) algorithm takes advantage of the proposed level-based path splitting technique to map the closest (or minimum cost) missed Virtual Network Function (VNF) nodes to minimize the needed resource and construct the Service Function Chain (SFC) for each source-destination pair in the multicast request.

REFERENCES

[1] B. A. A. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.

[2] N. Feamster, J. Rexford, and E. Zegura, "The road to sdn: An intellectual history of programmable networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 2, pp. 87–98, Apr. 2014.

[3] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[4] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 3–14, Aug. 2013.

[6] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *Communications Magazine, IEEE*, vol. 53, no. 2, pp. 90–97, 2015.

[7] B. Yi, X. Wang, K. Li, S. k. Das, and M. Huang, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, pp. 212 – 262, 2018.

[8] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba, "Net-

work function virtualization: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[9] D. G. Andersen, "Theoretical approaches to node assignment," *Computer Science Department at Carnegie Mellon University*, p. 86, 2002.

[10] J. M. Kleinberg, "Approximation algorithms for disjoint paths problems," Ph.D. dissertation, Massachusetts Institute of Technology, 1996.

[11] Y. Zhu and M. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2006, pp. 1–12.

[12] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, Mar. 2008.

[13] M. Chowdhury, F. Samuel, and R. Boutaba, "Polyvine: Policy-based virtual network embedding across multiple domains," *ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures*, pp. 49–56, Feb. 2010.

[14] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM Transactions on Networking*, vol. 20, no. 1, pp. 206–219, Feb. 2012.

[15] Q. Hu, Y. Wang, and X. Cao, "Resolve the virtual network embedding problem: A column generation approach," in *Proceedings of IEEE INFOCOM*, Apr. 2013, pp. 410–414.

[16] X. Liu, L. Gong, and Z. Zhu, "Design integrated rsa for multicast in elastic optical networks with a layered approach," in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2013, pp. 2346–2351.

[17] Q. Wang and L. K. Chen, “Performance analysis of multicast traffic over spectrum elastic optical networks,” in *OFC/NFOEC*, Mar. 2012, pp. 1–3.

[18] L. Gong, X. Zhou, X. Liu, W. Zhao, W. Lu, and Z. Zhu, “Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks,” *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 8, pp. 836–847, Aug. 2013.

[19] K. Walkowiak, R. Goścień, M. Tornatore, and M. Woźniak, “Impact of fanout and transmission reach on performance of multicasting in elastic optical networks,” in *OFC/NFOEC*, Mar. 2015, pp. 1–3.

[20] S. Ayoubi, C. Assi, K. Shaban, and L. Narayanan, “Minted: Multicast virtual network embedding in cloud data centers with delay constraints,” *IEEE Transactions on Communications*, vol. 63, no. 4, pp. 1291–1305, Apr. 2015.

[21] M. Zhang, C. Wu, M. Jiang, and Q. Yang, “Mapping multicast service-oriented virtual networks with delay and delay variation constraints,” in *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, Dec. 2010, pp. 1–5.

[22] Y. Miao, Q. Yang, C. Wu, M. Jiang, and J. Chen, “Multicast virtual network mapping for supporting multiple description coding-based video applications,” *Computer Networks*, vol. 57, no. 4, pp. 990 – 1002, 2013.

[23] Y. Wei, J. Wang, and Z. Gong, “A novel gossip-based multirate overlay multicast strategy,” in *2014 21st International Conference on Telecommunications (ICT)*, May 2014, pp. 196–200.

[24] D. Liao, G. Sun, V. Anand, H. Yu, and K. Xiao, “Opportunistic provisioning for multicast virtual network requests,” in *Proceedings of IEEE Globecom Workshops*, 2014, pp. 133–138.

[25] S. Ayoubi, K. Shaban, and C. Assi, “Multicast virtual network embedding in cloud data centers with delay constraints,” in *IEEE Cloud Computing (CLOUD)*, Jun. 2014, pp. 482–489.

[26] D. Liao, G. Sun, V. Anand, and H. Yu, “Survivable provisioning for multicast service oriented virtual network requests in cloud-based data centers,” *Optical Switching and Networking*, vol. 14, no. Part 3, pp. 260 – 273, 2014, sI: Optimization and Application in Converged Optical and Data Center Networks.

[27] J. Duan, Z. Guo, and Y. Yang, “Cost efficient and performance guaranteed virtual network embedding in multicast fat-tree dcns,” in *2015 IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2015, pp. 136–144.

[28] A. M. Ghaleb, T. Khalifa, S. Ayoubi, and K. B. Shaban, “Surviving link failures in multicast vn embedded applications,” in *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, Apr. 2016, pp. 645–651.

[29] J. Zhang, Y. Ji, M. Song, H. Li, R. Gu, Y. Zhao, and J. Zhang, “Dynamic virtual network embedding over multilayer optical networks,” *IEEE/OSA Journal of Optical Communications and Networking*, vol. 7, no. 9, pp. 918–927, Sep. 2015.

[30] S. Shakya and X. Cao, “Transparent virtual network embedding in elastic optical networks,” in *IEEE Sarnoff Symposium*, Sep. 2016, pp. 71–76.

[31] A. Pages, J. Perello, S. Spadaro, and G. Junyent, “Strategies for virtual optical network allocation,” *IEEE Communications Letters*, vol. 16, no. 2, pp. 268–271, Feb. 2012.

[32] L. Gong, W. Zhao, Y. Wen, and Z. Zhu, “Dynamic transparent virtual network embedding over elastic optical infrastructures,” in *IEEE International Conference on Communications (ICC)*, June 2013, pp. 3466–3470.

[33] J. Zhao, S. Subramaniam, and M. Brandt-Pearce, “Virtual topology mapping in elastic

optical networks," in *2013 IEEE International Conference on Communications (ICC)*, Jun. 2013, pp. 3904–3908.

[34] L. Gong and Z. Zhu, "Virtual optical network embedding (vone) over elastic optical networks," *Journal of Lightwave Technology*, vol. 32, no. 3, pp. 450–460, Feb. 2014.

[35] X. Gao, Z. Ye, W. Zhong, C. Qiao, X. Cao, H. Zhao, H. Yu, and V. Anand, "Multicast service-oriented virtual network mapping over elastic optical networks," in *IEEE International Conference on Communications (ICC)*, Jun. 2015, pp. 5174–5179.

[36] X. Gao, Z. Ye, J. Fan, W. Zhong, Y. Zhao, X. Cao, H. Yu, and C. Qiao, "Virtual network mapping for multicast services with max-min fairness of reliability," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 7, no. 9, pp. 942–951, Sep. 2015.

[37] P. Erdős and A. Rényi, "On the strength of connectedness of a random graph," *Acta Mathematica Academiae Scientiarum Hungarica*, vol. 12, no. 1, pp. 261–267, 1964.

[38] H. V. Tran and S. H. Ngo, "Improved heuristics for online node and link mapping problem in network virtualization," *ICCSA'13*, pp. 154–165, Jun. 2013.

[39] E. Guler, G. Luo, K. Koneru, and X. Cao, "Closeness-centrality based multicast-aware virtual network embedding," in *Proceedings of IEEE Globecom Workshops*, Dec. 2016.

[40] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215–239, 1978.

[41] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*, 3rd ed. The MIT Press, 2009.

[42] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[43] E. Guler, D. Zheng, G. Luo, L. Tian, and X. Cao, "Embedding virtual multicast trees in software-defined networks," in *IEEE International Conference on Communications (ICC)*, May 2017, pp. 1–6.

[44] A. Pages, J. Perello, S. Spadaro, J. A. Garcia-Espin, J. F. Riera, and S. Figuerola, "Optimal allocation of virtual optical networks for the future internet," in *International Conference on Optical Network Design and Modelling (ONDM)*, Apr. 2012, pp. 1–6.

[45] E. Guler, D. Zheng, G. Luo, L. Tian, and X. Cao, "Virtual multicast tree embedding over elastic optical networks," in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2017.

[46] F. S. Abkenar and A. G. Rahbar, "Study and analysis of routing and spectrum allocation (rsa) and routing, modulation and spectrum allocation (rmsa) algorithms in elastic optical networks (eons)," *Optical Switching and Networking*, vol. 23, Part 1, pp. 5 – 39, 2017.

[47] Q. Ma and P. Steenkiste, "On path selection for traffic with bandwidth guarantees," in *International Conference on Network Protocols*, Oct. 1997, pp. 191–202.

[48] E. Guler, D. Zheng, G. Luo, L. Tian, and X. Cao, "Embedding multicast services in optical networks with fanout limitation," in *IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6.

[49] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *IEEE Conference on Computer Communications (INFOCOM)*, Apr. 2014, pp. 1–9.

[50] S. Q. Zhang, Q. Zhang, H. Bannazadeh, and A. Leon-Garcia, "Routing algorithms for network function virtualization enabled multicast topology on sdn," *IEEE Transactions on Network and Service Management*, vol. 12, no. 4, pp. 580–594, Dec. 2015.

[51] M. Zeng, W. Fang, J. J. P. C. Rodrigues, and Z. Zhu, "Orchestrating multicast-oriented nfv trees in inter-dc elastic optical networks," in *IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[52] B. Yi, X. Wang, M. Huang, and A. Dong, "A multi-stage solution for nfv-enabled multicast over the hybrid infrastructure," *IEEE Communications Letters*, vol. 21, no. 9, pp. 2061–2064, Sep. 2017.

[53] M. Jalalitabar, E. Guler, D. Zheng, G. Luo, L. Tian, and X. Cao, "Embedding dependence-aware service function chains," *J. Opt. Commun. Netw.*, vol. 10, no. 8, pp. C64–C74, Aug. 2018.

[54] M. Jalalitabar, E. Guler, G. Luo, L. Tian, and X. Cao, "Dependence-aware service function chain design and mapping," in *IEEE Global Communications Conference (GLOBECOM)*, Dec. 2017, pp. 1–6.

[55] E. Guler, S. Devaraju, G. Luo, L. Tian, and X. Cao, "Multicast-Aware service function tree embedding," in *IEEE International Conference on High Performance Switching and Routing (HPSR)*, Xi'An, P.R. China, May 2019.