RELIABLE SERVICE CHAIN ORCHESTRATION FOR SCALABLE DATA-INTENSIVE COMPUTING AT INFRASTRUCTURE EDGES

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

---

by

DMITRII YURIEVICH CHEMODANOV

Dr. Prasad Calyam, Thesis Supervisor

July 2019

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled:

RELIABLE SERVICE CHAIN ORCHESTRATION FOR
SCALABLE DATA-INTENSIVE COMPUTING AT INFRASTRUCTURE EDGES

presented by Dmitrii Yurievich Chemodanov,
a candidate for the degree of Doctor of Philosophy and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Prasad Calyam

Dr. Kannappan Palaniappan

Dr. Zhihai He

Dr. Flavio Esposito

Dr. Isa Jahnke

DEDICATION

I dedicate this thesis to my late father: *you gave it your all to provide me a better life!*

ACKNOWLEDGMENTS

This work has been partially supported by the National Science Foundation awards CNS-1647084, CNS-1647182, the Coulter Foundation Translational Partnership Program and by RFBR according to the research project 16-07-00218a and the public tasks of the Ministry of Education and Science of the Russian Federation (2.974.2017/4.6). We thank the researchers who provided the public datasets in [1, 2, 3] and code [4] that we used for testing and evaluating the work in this thesis.

I would also like to acknowledge the faculty members and other students who contributed to this work and thank them for their support: Flavio Esposito, Kannappan Palaniappan, Ronald McGarvey, Zakariya Oraibi, Brittany Morago, Rengarajan Pelapur, Huy Trinh, Jon Patman and all VIMAN Lab students. Thanks for all your encouragement!

A very special gratitude goes out to two my best advisors not only in my research but also in my life – Prasad Calyam and Andrei Sukhov – for "optimally routing" me along this incredible life path.

And finally, last but by no means least, I am grateful to my parents Yurii Chemodanov and Lyudmila Chemodanova, who have provided me through moral and emotional support in my life. I am also grateful to my other family members and friends who have supported me along the way.

TABLE OF CONTENT

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

ABSTRACT

In the event of natural or man-made disasters, geospatial video analytics is valuable to provide situational awareness that can be extremely helpful for first responders. However, geospatial video analytics demands massive imagery/video data 'collection' from Internet-of-Things (IoT) and their seamless 'computation/consumption' within a geo-distributed (edge/core) cloud infrastructure in order to cater to user Quality of Experience (QoE) expectations. Thus, the edge computing needs to be designed with a reliable performance while interfacing with the core cloud to run computer vision algorithms. This is because infrastructure edges near locations generating imagery/video content are rarely equipped with high-performance computation capabilities. This thesis addresses challenges of interfacing edge and core cloud computing within the geo-distributed infrastructure as a novel 'function-centric computing' paradigm that brings new insights to computer vision, edge routing and network virtualization areas. Specifically, we detail the state-of-the-art techniques and illustrate our new/improved solution approaches based on function-centric computing for the two problems of: (i) high-throughput data collection from IoT devices at the wireless edge, and (ii) seamless data computation/consumption within the geo-distributed (edge/core) cloud infrastructure. To address (i), we present a novel deep learning-augmented geographic edge routing that relies on physical area knowledge obtained from satellite imagery. To address (ii), we describe a novel reliable service chain orchestration framework that builds upon microservices and utilizes a novel 'metapath composite variable' approach supported by a constrained-shortest path finder. Finally, we show both analytically and empirically, how our geographic routing, constrained shortest path finder and reliable service chain orchestration approaches that compose our function-centric computing framework are superior than many traditional and state-of-the-art techniques. As a result, we can significantly speedup (up to 4 times) data-intensive computing at infrastructure edges fostering effective disaster relief coordination to save lives.

Chapter 1

Introduction

"Computer Science is no more about computers

than astronomy is about telescopes"

— E.W. Dijkstra

Computer science advances go much beyond creating a new piece of hardware or software. Latest advances are increasingly fostering the development of new algorithms and protocols for a more effective/efficient use of cutting-edge technologies within new computing paradigms. What follows is an introduction to a novel function-centric computing paradigm that allows traditional computer vision applications to scale faster over geo-distributed (edge/core) cloud infrastructures. We discuss salient challenges in realizing this paradigm and show its benefits through a prism of disaster-incident response scenarios. Based on innovative data collection and computation solutions, we illustrate how geospatial video analytics enabled by function-centric computing can help co-ordinate disaster relief resources to save lives.

1.1 A Novel Computing Paradigm for Disaster-Incident Situational Awareness via Geospatial Video Analytics

In the event of natural or man-made disasters, timely and accurate situational awareness is crucial for managing first responders in disaster relief co-ordination. To this aim, imagery data such as videos and photographs can be collected from numerous disaster-incident scenes using surveillance cameras, civilian mobile devices, and aerial platforms. This imagery data processing can be essential for first responders to: (a) provide situational awareness for law enforcement officials (e.g., by using online face recognition to re-unite lost citizens [12]), and (b) inform critical decisions for allocating scarce relief resources (e.g., medical staff/ambulances or search-and-rescue teams [13]). Building dynamic 3-dimensional reconstruction of incident scenes can increase situational awareness in a theater-scale setting ($\sim$2 city blocks) of the incident scene. This can be achieved by fusing crowd-sources and data-intensive surveillance imagery [14]. Furthermore, tracking objects of interest in wide area motion imagery (WAMI) can provide analytics for planning wide-area relief and law enforcement activities at the regional-scale of incident scenes ($\sim$tens of city blocks) [15]. We term *geospatial video analytics* to refer to such an integration of computer vision algorithms implemented for different scales and types of imagery/video data collected from Internet-of-Things (IoT) devices, and processed through geo-distributed (edge/core) cloud infrastructures.

To be effective for users (i.e., incident commanders, first responders), geospatial video analytics needs to involve high-throughput data *collection* as well as seamless data *computation* and *consumption* of imagery/video. For instance, the user Quality of Experience (QoE) expectations demand that the geospatial video analytics is reliable with respect to any (wireless) edge network connectivity issues for data collection, while also delivering low-latency consumption of results (e.g., in real-time) to end-users by computing large amounts of visual data using complex computer vision algorithms [16]. Resources available at infrastructure edges can augment cloud resources as well as services closer to end-

user IoT devices to allow computing anywhere within the IoT-to-cloud continuum. This is known as edge (aka fog) computing, and providing applications with options for it reduces cloud service latencies by enabling computing closer to the IoT data sources at the infrastructure edge.

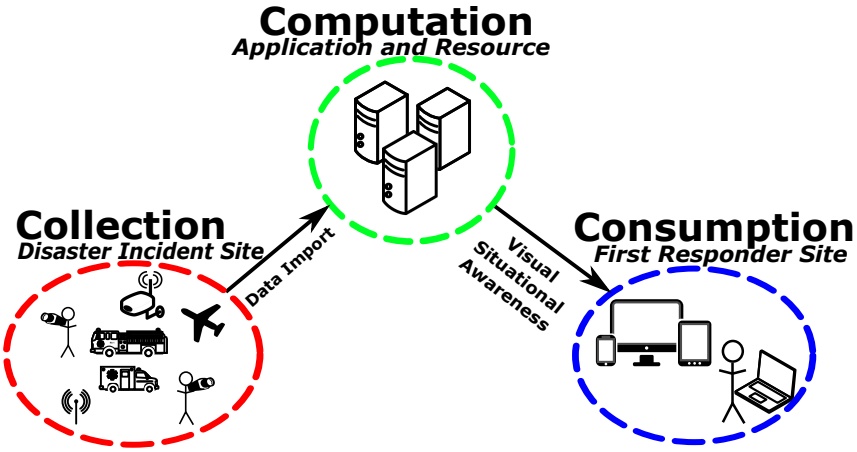


Figure 1.1: Illustrative example of a data-intensive computing at infrastructure edges (i.e., at fog) that needs to span geographically-dispersed sites of data collection, computation and consumption: edge resources here are linked with cloud computing platforms.

Figure 1.1 shows how geo-distributed edge/core cloud resources are used for visual data processing to realize a collection, computation and consumption (3C) pipeline that is common for any geospatial video analytics use case. In the stage of data collection, it is possible that the (wireless) network edge could have lost infrastructure, e.g., loss of cellular base stations in a disaster scene and/or intermittently accessible IoT devices (e.g., sensors, wearable heads-up display devices, bluetooth beacons). Similarly, in the stage of data computation/consumption, it is quite likely that infrastructure edges are rarely equipped with high-performance computation capabilities to run computer vision algorithms. Moreover, cloud-processed data needs to be moved closer to users through content caching at infrastructure edge resources for thin-client consumption and interactive visual data exploration. Thus, adoption of edge computing in conjunction with cloud computing platforms for relevant compute, storage and network resource provisioning and management requires a new computing paradigm.

This thesis describes techniques and methodologies designed to enable scalable data-intensive computing at infrastructure edges. Specifically, to address both the data collection and data computation/consumption problems in geospatial video analytics, we prescribe a novel 'function-centric computing' paradigm that integrates computer vision, edge routing and computer/network virtualization areas. To this aim, we start by addressing data *collection* challenges at the wireless edge in the potential scenario of a lost infrastructure in a disaster scene and design a novel deep learning-augmented edge routing protocol. We then address compute, storage and network resource provisioning and management challenges (that becomes an NP-hard problem to solve [17]) for data *computation/consumption* within a geo-distributed edge/core cloud infrastructure and propose a novel reliable service chain orchestration framework. However, before we discuss limitations of existing techniques and methods that can serve needs of the function-centric computing, we first motivate its need by describing our computer vision application case studies that can compose a common geospatial video analytics providing situational awareness to first responders.

1.2 Computer Vision Application Case Studies and Function-Centric Computing Motivation

In this section, we consider three common computer vision applications that operate on different data scale (e.g., theater-scale vs. regional-scale) and have different latency and geo-location requirements. Particularly, we describe benefits of function-centric computing for the following application case studies: (a) real-time patient triage status tracking with "Panacea's Cloud" Incident Command Dashboard [18] featuring Face Recognition, (b) reconstruction of dynamic visualizations from 3D Light Detection and Ranging (LIDAR) scans [19], and (c) tracking objects of interest in wide-area motion imagery (WAMI) [13]. Recall that we distinguish between theater-scale and regional-scale applications based on the geographical coverage of the incident and the nature of the distributed visual data - with

theater-scale being small area (∼2 city blocks) around a disaster incident site, and regional-scale being large areas (∼tens of city blocks) distributed across multiple disaster incident sites.

1.2.1 Patient Tracking with Face Recognition Case Study

Application's 3C pipeline needs. Following medical triage protocols of hospitals during response co-ordination at natural or man-made disaster incident scenes is challenging. Especially when dealing with several patients with trauma, it is stressful for paramedics to verbally communicate and track patients. In our first case study application viz., "Panacea's Cloud" [18], the function-centric computing needs to support a incident command dashboard shown in Figure 1.2 that can handle patient tracking using real-time IoT device data streams (e.g., video streams from wearable heads-up display or smartphones, geolocation information from virtual beacons) from multiple incident scenes. Proper aggregation of the IoT data sets and intuitive user interfaces in the dashboard can be critical for efficient coordination between first responder agencies, e.g., fire, police, hospitals, etc. [20]. The dashboard supports real-time videoconferencing of the paramedics with the incident commander for telemedical consultation, medical supplies replenishment communication, or coordination of ambulance routing at a *theater-scale* incident site. It can also facilitate patient triage status tracking through secure mapping of the geolocation information with a face recognition application, where the latter is used to recognize the person and if possible pull his/her relevant medical information [21]. Such a mapping is essential to ensure that the relevant patients at the incident sites are provided the necessary care, and follow-ups on the care can be scheduled with e.g., new responders or new location of the patient at the incident site. The face recognition capability can also be used in the dashboard for "lost person" use cases, where first responders can detect children in attempt to reunite them with their guardians [22], or identify bad actors in crowds that might dangerously escalate public safety resource allocation decisions.

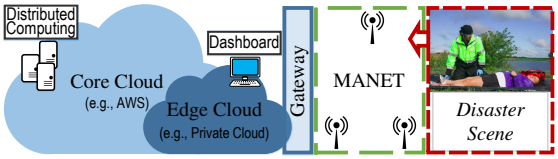


Figure 1.2: Illustrative example of the Panacea's Cloud setup: IoT device data sets generated on-site (e.g., near disaster scenes) need to be collected through a MANET at the edge cloud for further processing in conjunction with the core cloud.

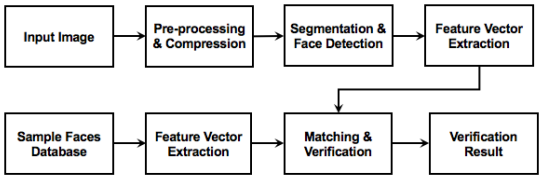


Figure 1.3: Overview of visual data processing stages in a facial recognition application used in patient triage status tracking.

Face recognition pipeline details. The visual data processing pipeline steps for face recognition are shown in Figure 1.3 and can be divided into two main classes according to the nature of the inherent functions such as:

- **Small function processing**: Compression, pre-processing and results verification
- **Large function processing**: Segmentation and face detection, features extraction and matching

Small function processing is mainly focused on pure pixel level information. In contrast, large function processing is focused on both pixel and object level information. To work effectively, large functions typically require data pre-processing stages. Figure 1.3 shows facial recognition steps that are used for "Panacea's Cloud" patient tracking and involve the digital image at the client side and a larger image sample dataset at the server side. A pre-processing step is performed to first detect a human face within a small amount of time (i.e., with low latency). During this step, all input images are compressed to $1/4^{th}$ of their original size. Subsequently, the image is fed into a pre-trained face classifier. This classifier is provided by Dlib [23] (an open-source library) and is based on [24] for facial recognition tasks. Training is done using a "very deep" Convolutional Neural Network which is comprised from a long sequence of convolutional layers which have recently shown state-of-the-art performance in related tasks.

1.2.2 3D Scene Reconstruction from LIDAR Scans

Application's 3C pipeline needs. 3D scene reconstructions have been proven to be useful for a quick damage assessment by public safety organizations. Such an assessment is possible through highly accurate LIDAR scans at incident scenes that provide evidence at relevant locations from multiple viewpoints [25]. As Figure 1.4 shows, 3D models of a scene can be created, e.g., by fusing a set of 2D videos and LIDAR scans. This visual data can be obtained from civilian mobile devices as well as from surveillance cameras near (or at) incident scenes. In our second case study application, function-centric computing registers 2D videos with 3D LIDAR scans collected at the theater-scale. As a result, first responders can view sets of videos in an intuitive (3D) virtual environment [19]. This simplifies the cumbersome task of analyzing several disparate 2D videos on a grid display. However, commonly-used data from LIDAR scans can be large in size – a typical resolution of about 1 cm for data collected at a range of up to 300 m with 6 mm accuracy. Thus, when collected from large-area incident scenes, this data can be computationally expensive to process. To help with this processing, function-centric computing can help take advantage of this rich source of information and quickly provide the situational awareness.

Figure 1.4: Illustrative example of a 3D scene reconstruction with use of LIDAR scans: a 2D video frame (*top left*) is first projected onto LIDAR scan (*bottom left*) to then reconstruct a 3D scene (*right*).

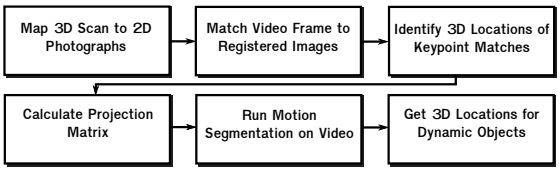


Figure 1.5: Overview of 3D scene reconstruction stages with 2D videos and LIDAR scans.

3D scene reconstruction pipeline details. Fusing a video with a LIDAR scan requires calculation of camera poses for 2D video with respect to the 3D point space. Figure 1.5 illustrates steps of matching a 2D video frame to LIDAR scans with known 3D correspondences, calculation of projection matrix for the camera, and segmentation of moving

objects in the 3D space. The visual data processing pipeline steps in this case study application can be classified as follows:

- **Small function processing**: Metadata data pre-processing, 2D video frame background registration, rendering of 3D
- **Large function processing**: Projection matrix calculation, segmentation of motion, positioning of dynamic objects

To better understand what each function computes, we describe one possible pipeline of the 3D scene reconstruction using LIDAR scan and 2D videos. We start by estimation of the 2D-3D relationship between the LIDAR scan and its photographs. To this end, we map 2D pixels to 3D points during a pre-processing step using standard computer vision techniques described in [26]. The entire point space is projected onto each image exactly once. Subsequently, the 2D-3D mappings can be stored at a remote cloud server location. Identification of moving objects in the 2D video is performed using the Mixture of Gaussians (MOG) approach [27] that yields a binary image featuring the motion segmented from the background.

1.2.3 Tracking Objects of Interest in WAMI

Application's 3C pipeline needs. Tracking objects of interest is crucial for (intelligent) search-and-rescue activities after a large-area disaster incident. Object tracking has long shown potential for city-wide crowd surveillance by providing a hawk-eye view for public safety organizations. Our third case study application that relates to search-and-rescue type of activities, operates on the *regional-scale* by identifying and tracking objects of interest in WAMI. This in turn can assist incident managers in studying the behaviors of particular vehicles [13]. The tracker's input is in the form of bounding boxes that indicate targets as shown in Figure 1.6. Novel sensor technologies allow to capture high resolution imagery data (ranging between 10 cm and 1 m) for wide-area surveillance. The processing is

performed by our Likelihood of Features Tracking (LOFT) framework that utilizes WAMI frames with a high-resolution of 25 cm Ground Sampling Distance (GSD) coming at about one to four frames per second [28]. Utilizing function-centric computing for tracking such imagery data can be challenging due to several reasons. First of all, the objects of interest are usually small with a (relatively) large motion displacement due to the inherent low frame rate. Secondly, WAMI imagery itself is challenging for automated analytics due to multiple reasons including (but not limited to) variations in illumination, oblique camera viewing angles, tracking through shadows, occlusions from tall structures, blurring and stabilization artifacts (e.g., due to atmospheric conditions). LOFT uses a set of imagery features including gradient magnitude, histogram of oriented gradients, median binary patterns [29], eigenvalues of the Hessian matrix for shape indices and intensity maps.

Figure 1.6: Illustrative example of WAMI imagery ecosystem: tiled (TIFF) aerial images of ≈80 MB size and with a resolution of 7800x10600 pixels.

Figure 1.7: Overview of object tracking stages in a typical WAMI analysis pipeline.

Object tracking pipeline details. Typical object tracking in wide-area motion imagery is comprised of several visual data processing stages tested on (large-area) aerial data [30, 31]. Figure 1.7 outlines these stages that can be classified according to the tracker functionality as follows:

- **Small function processing**: Raw data compression, data storage, metadata preprocessing, geo-projection, tiling and stabilization
- **Large function processing**: Initialization of objects of interest, their detection, tracking and analysis

A small function processing mainly operates on pure pixel level information, whereas a large function processing deals with information on both pixel as well as object levels. We

remark that for most of large functions to work effectively, we need the pre-processing stages while applying the function-centric computing approach. For example, common object trackers benefit from imagery stabilization, and hence registration becomes a crucial during data pre-processing. Our LOFT case study utilizes a track-before-detect approach to significantly reduce the search space which is good to have especially in large wide-area motion imagery as most of the objects in WAMI look similar [13, 32]. However, a proper synchronization between large tracking and small pre-processing functions is needed for this approach. To this aim, latency constraints have to be imposed to avoid bottlenecks in the tracking pipeline. When the tracker is fully consolidated on a single server, it has a processing rate of 3-to-4 frames per second. Thus, it is also important to take into account bandwidth demands because even a single WAMI frame can be ≈ 80 MB in size; and at 4 frames per second, the maximum throughput required is up to ≈ 2.5 Gbps.

1.3 Data Collection Challenges and Edge Routing

Providing technologies, e.g., *data collection* at wireless infrastructure edge, in response to a natural or man-made disaster is challenging. This is due to traditional infrastructure assumptions that may fail given the potential loss of infrastructure, e.g., loss of LTE base stations and given intermittently available and mobile IoT devices, e.g., sensors, wearable heads-up display devices, bluetooth beacons, etc [16]. For example, tablets, wearable heads-up display or smartphone devices can be used for the real-time video conferencing with the incident commander featuring face recognition of disaster victims [18, 21], or to detect children in attempt to reunite them with their guardians [22], whereas virtual beacons can be mainly used to track their location.

To cope with the potential loss of infrastructure near e.g., a disaster scene, a mobile ad-hoc wireless network (MANET) needs to be operational for collecting media-rich visual information from this scene as quickly as possible at the edge cloud gateway. These data

are then can be processed at the edge cloud in conjunction with the core cloud to enable seamless data-intensive computing and provide e.g., real-time visual situational awareness to first responders (see Figure 1.2).

For function-centric computing to be operational in such edge network scenarios, we need to steer user traffic dynamically within MANETs to satisfy its throughput and latency requirements. To this aim, we cannot adopt full-fledged routing solutions from core networks due to mobility as well as power constraints of IoT devices. Instead, lightweight routing approaches such as those based on Geographic routing are more suited. However, there is a lack of traffic steering techniques in MANETs that can provide sustainable high-speed delivery of data to a geo-distributed cloud infrastructure gateway component [33]. Specifically, there is a need to design a better performant greedy-forwarding solutions that does not suffer from the *local minimum problem* in presence of (non-arbitrary) node failures and mobility. This problem is caused by the lack of global routing knowledge of the greedy forwarding algorithm [34, 35, 36, 37] that may deliver packets to nodes that do not have neighbors closer to the destination than themselves.

Existing algorithms provide only partial solutions to the local minimum problem and can be classified into stateful or stateless solutions. Existing stateless greedy forwarding solutions may fail to find a path even if it exists [35, 36], or they stretch such paths significantly [34] by visiting almost all possible nodes to "desperately" find a way. Existing stateful greedy forwarding algorithms instead (e.g., those relying on the network topology knowledge including spanning trees [38, 39] or partial paths [40]) are sensitive to frequent node failures and mobility [40, 41, 42], a typical scenario within regions and infrastructures hit by a natural or man-made disaster. Consequently, such algorithms lead to poor or unacceptable performance in incident-supporting applications that need constant high-speed data transfer to provide crucial real-time situational awareness.

1.4 Data Computation Challenges and Reliable Service Chain Orchestration

In the stage of data computation/consumption, it is quite likely that infrastructure edges are rarely equipped with high-performance computation capabilities to run computer vision algorithms. Moreover, cloud-processed data needs to be moved closer to users through content caching at fog resources for thin-client consumption and interactive visual data exploration. To overcome such infrastructure limitations that are especially likely in specific cases of natural or man-made disaster-incidents, the function-centric computing interfaces edge computing [43] in conjunction with the core cloud platforms.

Figure 1.8 illustrates how this interfacing is done within function-centric computing paradigm. The pre-processing and Human-Computer Interaction (HCI) analysis functions (i.e., "small function processing") can be placed for a low-latency access on edge servers by using edge computing. For processing large-scale visual data sets using e.g., object tracking computer vision algorithms, the function-centric computing involves the placement of compute-intensive tracking functions (i.e., "large function processing") on a cloud server.

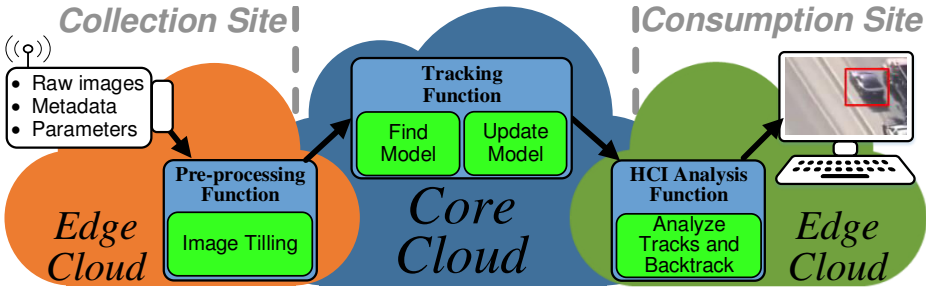


Figure 1.8: Function-centric computing paradigm: computer vision tracking application execution is scaled and accelerated by first decoupling it into a set of services (functions) forming a chain; next, inter-process communication (including latency and geo-location) constraints are added to the service chain to form a request; finally, we manage such chain request with a service chain orchestration and provision resources at both core and edge clouds.

However, interfacing geo-distributed edge/core cloud resources poses new challenges for the function-centric computing. This is because function-centric computing implementation requires refactoring computer vision applications that are typically developed with

codes that are tightly coupled and are not designed for function portability [16]. For this, principles from recent advent of microservices used in Amazon Web Services Lambda [44], Google Cloud Functions [45], Microsoft Azure Functions [46] and IBM OpenWhisk [47] can be used. More specifically, microservices can mitigate mitigate application scalability limitations by its code decoupling (e.g., performed via RESTful APIs) to manage load balancing, elasticity, and server instance types independently for each application function.

Furthermore, optimal placement of computer vision functions and their connection via network as shown in Figure 1.8 is a known (NP-hard) "Service Function Chaining" problem in the Network Function Virtualization (NFV) area [17]. Particularly in NFV, network operators dynamically *place* virtualized network functions (e.g., firewalls, load-balancers, etc.), *chain* them for a service flow routing and allocate corresponding compute/network resources in a cloud infrastructure by utilizing SFCs [48]. Recently, areas such as Microservices [49] and Mobile Edge Computing [50, 51] have also shown benefits of adopting the service chain technology. Thus, service chains can be now composed from both core and edge cloud resources forming geo-distributed chains to satisfy geo-location and latency requirements of their functions [50, 51].

However, before we discuss limitations of existing orchestration approaches for geo-distributed latency-sensitive service chains (that can serve needs of the function-centric computing), we first give new insights to a role and limitations of the constrained shortest path management in the service function chaining in particular and the network virtualization area in general.

1.4.1 Constrained Shortest Path and Network Virtualization

Successful virtual network service composition and maintenance in both management plane and data plane mechanisms requires *scalable* and *flexible* 'constrained shortest path management' for the following reasons. The constrained shortest path problem is the NP-hard problem of finding the shortest path that satisfies an arbitrary number of end-to-end (or

path) constraints [52], and its existing exact solutions are commonly based on branch-and-bound exhaustive search algorithms [52, 53, 54, 55] or integer programming. These techniques have exponential complexity, and hence, limit *scalability* of the constrained shortest path management. Such scalability limitations are exacerbated by the complexity of virtual network service management algorithms as well as the potentially large scale of substrate networks. Although some heuristics or approximation algorithms can find constrained shortest paths in polynomial time (at expense of optimality [56]), these algorithms support only a specific number of constraints, or a specific cost to optimize [57, 58, 59]. Thereby, they limit the *flexibility* of the constrained shortest path management.

By leveraging constrained shortest paths, in this thesis we show how we can enhance network utilization and energy efficiency of common virtual network services by both analytical and empirical means. Reasons for the observed benefits are as follows: Firstly, by using a constrained shortest path, we can minimize a provider's cost associated with flows allocation subject to the application Service Level Objective (SLO) constraints in traffic engineering [6, 7]. Such costs potentially hinder long-term infrastructure providers' revenue maximization; this can be understood using a simple example: a path comprising of two physical links to maintain a single flow has a higher allocation cost than an alternative solution that uses only one physical link. For example, if the flow SLO demand is 10 Mbps, a path composed by a single physical link would need to provision only 10 Mbps, while a path composed by two links would require 20 Mbps. Secondly, a more scalable and flexible constrained shortest path approach can be also beneficial for service function chaining [60, 61, 62] when virtual links between services are subject to an arbitrary number of constraints such as bandwidth, latency and loss rate. Other types of constraints can also be imposed by optimization methods, such as the column generation approach, typically used to speed-up integer programming [53, 61, 62]. *This is because in the aforementioned cases, constrained shortest paths can be part of the optimal solution, i.e., such paths can best improve the objective value under arbitrary constraints.*

1.4.2 Reliable Service Chain Orchestration

Having the constrained shortest path management for the network virtualization area discussed, we now show a lack of reliable orchestration approaches for geo-distributed latency-sensitive service chains that can serve needs of the function-centric computing.

Geo-distributed service chains can be subject to node failures and congested network paths leading to their frequent Quality of Service (QoS) demands violations [63]. In some specific cases of natural or man-made disaster-incidents, they can be subject to severe infrastructure outages [3]. Moreover, computation and network QoS demands of service chains can fluctuate themselves [64]. Thus, their reliable orchestration is needed to cope both proactively upon their *composition* as well as reactively during their lifespan *maintenance* with potential service chain demand fluctuations [64] as well as with possible infrastructure outages [63, 3].

At the same time, providing a continuously available and reliable service chain orchestration is hard [65]. First of all, this is because the optimal service chain composition has known approximation guarantees only in some special cases. For instance, when chaining of service functions (e.g., satisfying bandwidth requirements between two consequent functions) [66] and/or their ordering (i.e., ensuring that function A precedes function B) [67, 68] are omitted. In the general case however, it requires solving of the NP-hard integer *multi-commodity-chain flow* problem to align flow splits with supported hardware granularity [7]. It is also necessary to support cases when service functions or their associated flows are non-splittable. This problem has no known approximation guarantees and has been previously reported as the integer NFV service distribution problem [69]. Moreover, its complexity can be further exacerbated by incorporated reliability and geo-location/latency aware mechanisms. The former aims to cope proactively with both possible infrastructure outages as well as service chain demand fluctuations, whereas the latter is needed to satisfy QoS demands of geo-distributed latency-sensitive service chains.

Secondly, due to scale and nature of geo-distributed service chains, having their orches-

tration with a single point of failure or congestion (i.e., centralized) is too risky. On the one hand, having a SFC *maintenance* which is based on a distributed control plane system is crucial. On the other hand, distributed control plane-based Software Defined Networking (SDN) systems such as ONOS [70] require consistency guarantees to avoid various violations [71]. Examples of such violations tailored to the SFC orchestration can be double assignment of services, looping paths, QoS constraints violations and others. A common approach to guarantee consistency of a distributed control plane is by establishing a consensus. The latter can be done by running known consensus protocols such as Raft [72], etc. To the best of our knowledge, the only distributed service chain orchestration scheme that builds upon consensus literature is 'Catena' [73, 74]. This algorithm ensures a consistency by running consensus protocols based on specified policies and can be used safely within distributed control plane. However, using consensus protocols for every service placement or its chaining in a service chain request is expensive.

1.5 Thesis Contributions and Organization

This thesis addresses challenges of interfacing edge computing with core cloud platforms as a function-centric computing paradigm that brings new insights to computer vision, edge routing and network virtualization areas. In detail, the contributions of my thesis can be summarized as follows:

- ***Theoretical contributions.*** The **first** theoretical contribution lays in my edge routing results pertaining to a geographic routing model that guarantees a local minimum avoidance as well as the shortest path approximation in (mobile) ad-hoc networks (see Chapter 3). Specifically, this stateless greedy forwarding approach builds upon electrostatics principles such as the Green's function [75] to model packets as charges immersed in an electrostatic potential field from the source to the destination to steer their route by charging regions containing obstacles accordingly, hence guaranteeing

local minimum avoidance. Moreover, when packets are forwarded using gradient descent on the Green's function potential field (along lines of electrical force), a ***3.291*** path stretch approximation bound is shown. To my knowledge, this is the first greedy forwarding algorithm that has theoretical guarantees on the path stretch length without any strong assumptions on wireless ad-hoc network such as symmetrical links or unit disk graphs.

The **second and main** theoretical contribution lays in an asymptotic complexity of my novel general constrained shortest path algorithm $viz.$, 'Neighborhoods Method' (NM) [52]. The NM is based on a novel double-pass search space reduction technique that has a ***quadratically*** lower complexity upper-bound (halved exponent) than recent state-of-the-art branch-and-bound exhaustive search methods [53, 54, 55] (see Chapter 4).

- ***Algorithmic contributions.*** The **first** algorithmic contribution lays in my two novel edge routing algorithms, $viz.$ *Attractive Repulsive Greedy Forwarding* (ARGF) and *Attractive Repulsive Pressure Greedy Forwarding* (ARPGF). Both algorithms use the notion of *electrostatic repulsion* to enhance greedy forwarding (see Chapter 3). ARGF does not theoretically guarantee the local minimum avoidance due to the complexity of computing the exact theoretical potential field on multiple obstacles of arbitrary shape and due to discrete node distribution. For this reason, we extend ARGF with a known pressure recovery technique (ARPGF) to guarantee delivery at expense of a small path stretch. The numerical simulations with asymmetrical connectivity and obstacles of complex convex shape show how both ARGF and ARPGF outperform related stateless greedy forwarding solutions such as Greedy Forwarding [36], Greedy Perimeter Stateless Routing (GPSR) [35] (*face routing* representative), and Gravity Pressure Greedy Forwarding (GPGF) [34] (*pressure forwarding* representative). Particularly, ARPGF outperforms GPGF under practical Time To Live (TTL) constraints. Moreover, I show that when the packet TTL

≤ 128 and under legitimate assumptions, ARPGF data could fit in the available IP packet header space, hence it has minor overhead. At the same time, it was found that GPSR performs worse than the ARGF algorithm due to the former's unrealistic assumptions on the underlaying network graph, i.e., unit disk or planar graph. Finally, my NS-3 [76] event-driven simulations also confirm superior ARGF (and hence ARPGF) goodput performance compared to GPSR and other stateful reactive routing protocols, such as Ad-hoc On-demand Distance Vector (AODV) [77] and IEEE 802.11s standard Hybrid Mesh Network Protocol (HWMP) [78], especially in challenging disaster response conditions of severe node failures and high mobility. The source code of both simulations is publicly available under a GNU license at `https://github.com/duman190/AGRA`.

The **second** algorithmic contribution is my NM algorithm that synergizes dynamic programming with a branch-and-bound exhaustive-search (see Chapter 4). NM also uses a novel infeasibility pruning technique $viz.$, 'Look Back', which benefits from NM's double pass design to further ease the constrained shortest paths finding in practice. The NM is solving an NP-hard problem, so it is exponential in its general form. However, when it is synergistically used with existing search space reduction techniques [53, 54, 55], my scalability evaluation results indicate how NM is faster by an ***order of magnitude*** than recent branch-and-bound exhaustive search methods, and hence, *scales* better [52].

The **third and main** algorithmic contribution lays in my novel (and the first to my knowledge [17]) practical and near optimal reliable service chain composition approach in the general case of joint service function placement and chaining in a geo-distributed cloud infrastructure that also admits end-to-end network QoS constraints such as latency, packet loss, etc. Thus, it can serve for needs of function-centric computing (see Chapter 5). To ensure reliability upon service chain composition (i.e., proactively), I compose geo-distributed latency-sensitive service chains with capac-

ity chance-constraints (that handle both service chain demand fluctuations as well as infrastructure outages uncertainties) and with backup policies. To this end, the (master) NP-hard integer *Multi-Commodity-Chain Flow* problem is formulated which has been previously adopted in NFV literature [69]. To cope with this master problem solution intractabilities, I propose a novel first-of-its-kind *metapath composite variable* approach that is similar to other composite variable solutions [79] in terms of its nature that aggregates multiple decisions within a single binary variable. Specifically, I aggregate feasible mapping decisions for each single-link service chain segment as a set of k-constrained shortest metapaths generated with the NM algorithm [52]. Service chain segments are then assigned to their associated metapaths either optimally by using generalized assignment problem (GAP) [80] or suboptimally by using its (polynomial) Lagrangian relaxation counterpart [81]. Using trace-driven simulations of real US Tier-1 ($\sim$300 nodes) and regional ($\sim$600 nodes) infrastructure providers' topologies, I first show how my service chain composition approach achieves 99% optimality on average. In addition, I show that it only takes time on the order of seconds for practically sized problems in contrast with the master problem solution that takes several hours. By recreating challenging disaster-incident scenarios as in [3], I lastly show how my approach can compose up to 3 times more sequentially incoming service chain requests w.r.t. path (or column) generation approach [61, 62] and up to 2 times more such requests w.r.t. isomorphism-based heuristics [82]. Note that all these solutions are suboptimal w.r.t. the optimal (master) service chain composition, but scale better in practice.

The **forth** algorithmic contribution lays in a new distributed metapath-based service chain *maintenance* algorithm that augments my reliable service chain composition algorithm [17] to ensure service chain reliability reactively and form a *reliable service chain orchestration* framework. I design this service chain maintenance algorithm for a distributed control plane with the latter's consistency guarantees without use of

expensive consensus protocols. To this aim, I prove the algorithm eventual correctness property to qualify for a use of Simple Coordination Layer [71]. The evaluation results show how my service chain maintenance is superior than Catena [73, 74] in terms of optimality by utilizing metapath composite variables and in terms of number of control messages due to avoidance of expensive consensus protocols (see Chapter 5).

- ***Design contributions.*** The **first** design contribution lays in my AI-augmented Geographic Routing Approach (AGRA) that utilizes a knowledge about physical obstacles presence in a geographic area obtained by deep learning [83, 84] from the satellite imagery (maps) available at the system's dashboard (see Chapter 3). In addition to widely adopted geographic coordinates, the obtained geographical obstacle knowledge is then used to build a conceptually different greedy forwarding approach that avoids the *local minima problem* while supporting high-speed data delivery, e.g., of high-definition data streams and multi-modal data, across large disaster incident scene areas.

  The **second and main** design contribution lays in my reliable service chain orchestration prototype that serves needs of function-centric computing and publicly available at [11]. The reliable service chain orchestration prototype builds upon Docker containers to allocate compute resources for application functions as well as Software-Defined Networking system to manage function communications between edge and core cloud sites (see Chapter 6). Using an exemplar case study of an object tracking application (commonly utilized in geospatial video analytics) [13], I show how current computer vision applications can be decomposed into chains of functions. Lastly, I conduct evaluation experiments using an edge computing testbed in the GENI infrastructure [85]. The experiment results demonstrate how my service chain orchestration prototype can speed-up the exemplar case study application by up to 4 times in comparison to common cloud computing over IP networks.

Thesis Organization. The remainder of this thesis is organized as follows: Chapter 2 provides a review of literature related to the work in this thesis. Chapter 3 details a novel edge routing approach and how it can be used for data collection at the wireless infrastructure edge. Chapter 4 discusses how flexible and scalable constrained shortest path algorithm is essential for virtual network service management. Chapter 5 covers details on the reliable service chain orchestration that can serve needs of function-centric computing. Chapter 6 details the function-centric computing prototype implementation and Chapter 7 summarizes this thesis.

Chapter 2

Literature Review

Our work on scalable data-intensive computing at infrastructure edges utilizing reliable service chain orchestration mainly contributes to a novel approach of 'function-centric computing' by combining synergies from the fields such as edge routing and IoT, edge computing and Network Virtualization such as NFV service chaining. The literature pertaining to these sub-areas is vast, and we only focus here on a few works that are most relevant to discuss our contributions. The recent survey on serverless and edge computing can be found in [86], and surveys on Network Virtualization including NFV service chaining are discussed in [48, 65, 87, 88].

2.1 Data Collection and Edge Routing

2.1.1 Edge routing and IoT

Recent advances in the IoT have brought challenges in storage, networking and computation management and under several scenarios, including mobile edge computing [16, 89], wireless sensor networking [90, 91] or cognitive networking [92, 93]. Among the most severe IoT challenges for data marshalling, we have seen scarce energy, high mobility and

frequent failures [40, 92]. To overcome some of these challenges in bridging IoT devices with the gateway, edge and core clouds, recent advances in the Software-Defined Networking (SDN) and Network Function Virtualization (NFV) have been adopted [16, 89]. For example, a control plane has been used to dynamically find low-latency and high bandwidth paths that satisfy IoT-based application demands [16, 73]. Existing solutions that leverage data transfer to a gateway commonly require knowledge of the network topology such as spanning trees [38, 39] or network clusters [91]. In this thesis, we consider a special case of data delivery from the disaster incident scenes to the gateway over MANETs, that is in contrast to typical wireless sensor network scenarios. Because of the high node mobility (i.e., due to use of MANETs) and sever node failures (e.g., due to intermittent energy supply), solutions that rely on a logically centralized network control [16, 89] or on the network topology knowledge [39, 40, 41, 91] can be inadequate. Our approach copes with the above limitations in disaster incident scenarios via a stateless greedy forwarding protocol that finds high bandwidth and low-latency paths by improving packets delivery and minimizing path stretch. Moreover, our approach does not directly address energy scarceness of the IoT devices, but copes with it in a best effort manner by using stateless greedy forwarding that avoids usage of routing protocols that can drain a battery of IoT devices relatively fast.

2.1.2 Physics in computer networks

Applying physics laws to solve computer network problems is not a novelty. The first successful attempt, to our knowledge is the popular result by Shannon, who created the basics of information theory relying on the entropy definition from physics [94]. To justify network effects new models such as the "small world" effect, cluster models, network correlation, random graph model, network growth model and many others have been developed. All these models rely on physics to some extent. A survey of these models can be found in [95].

Narrowing our attention to routing and forwarding schemes using potential fields, we found a few routing and forwarding schemes using potential fields similar to our approach [96, 97, 98]. Their solution is aimed at balancing the network load by the natural property of electrostatic lines of force to be geo-spatially dispersed. In [98], authors use the aforementioned property to select a path trajectory so that a greedy forwarded packet can reach the destination without facing a local minimum. In [97], authors use numerical calculations to optimize network load for one-to-many and many-to-many communication patterns. However, both schemes do not address the local minimum problem due to presence of obstacles directly and can benefit from using our approach. In [96], the authors use a potential field to repulse traffic in excess from heavy loaded sensors to reduce congestion. Our proposed approach can be used similarly to directly deal with congestions and other disaster incident-supporting geographic routing problems.

2.1.3 Geographic routing and MANET

The literature on geographic routing and greedy forwarding is also vast, and here we focus on the most valuable works that help to highlight our contributions. Many geographic routing algorithms that can recover from a local minimum have been proposed. One of the first geographic routing solutions which guarantees delivery were Greedy Perimeter Stateless Routing (GPSR) [35] and GFG [99]. To recover from a local minimum, both protocols use face routing which requires strong assumptions such as unit disk and planar graphs. However, planar graphs can be disconnected when graphs have arbitrary shapes, nodes are mobile and real physical obstacles appear [100]. Kim $et\ al.$ [100] propose a solution which overcomes planar graph limitations in practice by introducing Cross-Link Detection Protocol (CLDP) complication. As later works have show [39, 41], CLDP requires expensive signaling to detect and remove crossed edges. Authors in [39] proposed Greedy Distributed Spanning Tree Routing (GDSTR) which requires less expensive distributed spanning tree construction (to maintain one or several spanning trees) to guarantee delivery and recover

from a local minimum. GDSTR is also extended to a 3D case [101]. Kleinberg $et\ al.$ [38] use spanning trees for greedy embedding, i.e., for an assignment of virtual coordinates to greedy forward a packet without facing a local minimum. More recent works [40, 41, 42] show that spanning trees are sensitive to dynamic topologies and mobility. Moreover, most of the aforementioned solutions were designed for static sensor networks which are limited in dynamics. Our approach obtains better path stretch results, also works in 3D spaces, but does not require the time and space complexity of a spanning trees construction.

More recent protocols such as MTD and WEAVE [40, 41] can cope with topology dynamics to some extent. For example MTD requires construction of Delaunay triangulation (DT) graphs for local minimum recovery. When topology changes, nodes may loose their Delaunay neighbors which are needed for recovery from a local minimum. Contrary to ours, all of the aforementioned protocols are stateful — i.e., they rely on global or partial topology knowledge and therefore their performance degrades under node mobility or failures — common for disaster scenarios. Moreover, all these algorithms build around greedy forwarding and hence, they can benefit from using our *repulsive* field to proactively avoid local minima created by obstacles. To our knowledge, we are the first to introduce a theoretical solution to the local minimum avoidance that approximates with a bound the shortest path in ad-hoc networks by creating conceptually different forwarding decision rules.

2.1.4 Geographic routing and the Internet

Geographic routing has been also proposed for the Internet [34, 42, 102] that is less dynamic than wireless ad-hoc networks. The authors in [34] build upon the work of Kleinberg $et\ al.$ [38] and show that due to inaccurate greedy embedding caused by topology dynamics, packets can get stuck in a local minimum. To this aim, they propose the Gravity Pressure Greedy Forwarding (GPGF) [34] protocol which is shown to have guaranteed packet delivery on graphs of an arbitrary shape. To recover from local minimum, GPGF

counts the number of node visits (storing that information in packet headers) to press packets from local minima until greedy forwarding can resume. The key idea beyond pressure recovery is a greedy forwarding gradient descent property — once a packet reaches a location closer to the destination, there is no way how the packet can be forwarded back to the previous location of a local minimum. However, such a recovery needs expensive packet header space [42] and can stretch path significantly to undesired levels.

2.1.5 AGRA: AI-augmented geographic routing

In this thesis, we use a conceptually different *repulsive* field for geographic routing to benefit from a static physical obstacle knowledge obtained by using deep learning-based detectors [83, 84] over any available edge satellite maps. The proposed approach based on the electrostatic potential of Green's function theoretically guarantees avoidance of a local minimum as well as the shortest path approximation. The electrostatic field guiding packets has a gradient descent property, with minimum at the destination. This means that greedy forwarding can be also complemented with the gravity pressure mode of GPGF for a local minimum avoidance. Our proposed algorithms, i.e. Attractive Repulsive Greedy Forwarding (ARGF) and Attractive Repulsive Pressure Greedy Forwarding (ARPGF) use both repulsive and attractive fields to greedy forward a packet (in 2D or 3D Euclidean spaces). As shown in our simulations, such a greedy forwarding synergy enhances the path stretching property of GPGF and hence the delivery ratio (limited by the packet's Time To Live), making ARPGF suitable for the incident-supporting wireless edge (i.e., ad hoc) networks. Due to the static obstacle knowledge, proposed algorithms can cope better with high mobility and severe node failures, which results in an overall greater goodput during disaster scenarios, crucial for most of the incident-supporting situational awareness applications. In the absence of obstacles knowledge or with obstacle location miscalculations by deep learning, the performance of such proposed algorithms degrades with respect to their respective predecessor performances, i.e., ARGF to GF [36] and ARPGF to GPGF [34].

2.2 Network Virtualization and Reliable Service Chaining

2.2.1 On Constrained Shortest Path Problem for Virtual Network Service Management

The problem of providing a (shortest) path with multiple (e.g., virtual network service SLO) constraints is NP-hard [53], and its complete survey can be found in [56]. Herein, we mention a few representative solutions that help us present our novel contributions. Most heuristics group multiple metrics into a single function reducing the problem to a single constrained routing problem [103], and then solve the routing optimization separately, e.g., using Lagrangian relaxation [104]. The exact pseudo-polynomial algorithm proposed by Jaffe $et\ al.$ [57] offers a distributed path finder solution limited to a two-path constraints problem. Wang $et\ al.$ [105] use an extended version of Dijkstra algorithm (EDijkstra), where all links with infeasible hop-to-hop constraints are excluded. EDijkstra runs in polynomial time but may omit any (path hop count) minimization, desirable in network virtualization to optimize the physical network utilization. To minimize the path hop count under a single path constraint (e.g., delay) an iterative modification of Bellman-Ford (IBF) algorithm was proposed in [58]. Our approach is not limited to a single path constraint and can be adapted to subsume both EDijkstra and IBF as we discuss in Section 4.3.

The authors in [55] propose an exact algorithm for the constrained shortest path problem, and apply several search space reduction techniques such as dominated paths (pruning by dominance and bound) and the look-ahead (pruning by infeasibility) notion for the exponential complexity exhaustive search, utilizing the k-shortest path algorithm. We also apply a similar technique to reduce the constrained path search space, though without any look-ahead, since it is computationally expensive. Instead, our design uses a more efficient "Look-Back" pruning technique (see Section 4.2.2). In [106], the authors propose an Exhaustive Breath-First Search (EBFS) based approach to solve the constrained path finder problem, focussing on delay. Another more recent work [54] also uses EBFS with

a dominant path space reduction technique to find multi-criteria Pareto-optimal paths. Alternatively, an exhaustive Depth-First Search (EDFS) can be used as a branch-and-bound algorithm. For example, the authors in [53] proposed the "pulse" algorithm that uses EDFS with dominated paths and look-ahead search space reduction techniques. Both of EBFS and EDFS algorithms have exponential worst case time complexity. Our solution however quadratically reduces the worst case complexity of these algorithms (see Section 4.3).

2.2.2 Reliable Service Chaining

Service chaining is traditionally used in NFV to place a set of middleboxes and chain relevant functions to steer traffic through them [65]. Existing service chaining solutions either separate the service placement from the service chaining phase [66, 67, 68], or jointly optimize both the two phases [64, 69].

Service Chain Optimality. In some special cases the optimal service chaining is shown to have approximation guarantees [66, 67, 68]. For instance, Cohen $et\ al.$ [67] and Sang $et\ al.$ [68] provide near optimal approximation algorithms for the service chaining problem without chaining and ordering constraints. Tomassilli $et\ al.$ [66] propose the first service chaining solution with the approximation guarantees which admits ordering constraints, but still omits chaining constraints. Guo $et\ al.$ [107] show approximation guarantees for service chains with both ordering and chaining constraints, but only under assumptions that available service chaining options are of polynomial size. In the general case however, when service functions need to be jointly placed and chained in a geo-distributed cloud infrastructure with a corresponding compute/network resource allocation, possible service chain compositions are of exponential size. Thus, it becomes a linear topology Virtual Network Embedding [50, 60] and can be formulated as the (NP-hard) multi-commodity-chain flow problem with integrality constraints with no known approximation guarantees [69]. Thus, Feng $et\ al.$ [69] propose a heuristic algorithm whose preliminary evaluation results in a small-scale network settings (of $\sim$10 nodes) shows promise for providing efficient

solutions to the integer multi-commodity-chain flow problem in practical settings.

In this thesis, we propose the first to our knowledge practical and near optimal service chain composition approach in the general case of joint service function placement and chaining in a geo-distributed cloud infrastructure that also admits end-to-end network QoS constraints such as latency, packet loss, etc. To this aim, we propose a novel *metapath composite variable approach* which reduces a combinatorial complexity of the (master) integer multi-commodity-chain flow problem. As a result, our approach achieves 99% optimality on average and takes seconds to compose service chains for practically sized problems of US Tier-1 ($\sim$300 nodes) and regional ($\sim$600 nodes) infrastructure providers' topologies, where master problem solution takes hours using a High Performance Computing cloud server.

Service Chain Reliability. With the advent of edge networking and growing number of latency sensitive services, recent works also consider problems of geo-distributed [108] and edge service chaining [51]. Although these works mainly focuses on the new load balancing and latency optimization techniques, they omit an important reliability aspect of geo-distributed latency-sensitive service chains. The closest works related to ours are [64] and [82]. Fei $et\ al.$ [64] propose a prediction-based approach that proactively handles service chain demand fluctuations. However, their approach does not account for network/infrastructure outages that mainly cause service function failures [63]. At the same time, Spinnewyn $et\ al.$ [82] propose a service chaining solution that ensures a sufficient infrastructure reliability, but neither proactively nor reactively handles service chain demand fluctuations.

In contrast to [64] and [82], our reliable composition scheme uniquely ensures reliability of geo-distributed latency-sensitive service chains by using of chance-constraints and backup policies to cope with both service chain demand fluctuations and infrastructure outages [17]. It then also *maintains* them during their lifespan to ensure reliability of already composed service chains (i.e., reactively) by utilizing a distributed control plane to avoid a

single point of failure or congestion. We also found only one service chain orchestration approach that guarantees a distributed control plane consistency during service chain orchestration — Catena [73]. In contrast to Catena that uses expensive consensus protocols, we prove our metapath-based maintenance algorithm *eventual correctness* to use Simple Coordination Layer [71].

Chapter 3

Data Collection and Edge Routing

In this chapter, we propose a novel edge routing solution that can address data collection challenges within lost infrastructure regions to allow function-centric computing.

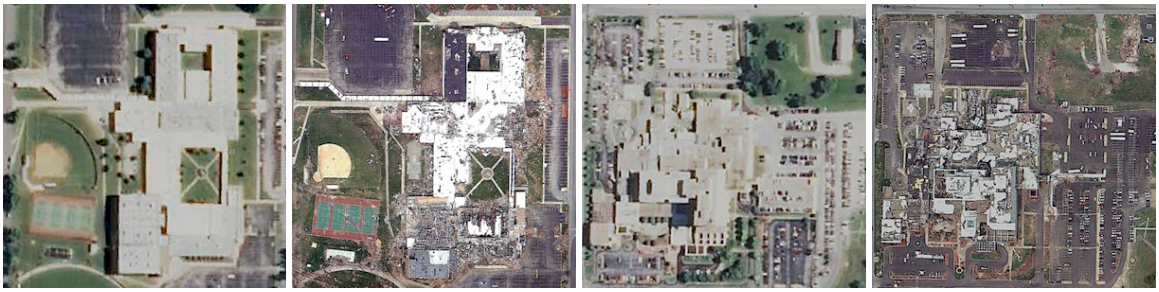
3.1 Problem Motivation and Artificial Intelligence Relevance

In the highly mobile and frequent node failure conditions during a disaster-incident response, we cannot rely on the network topology knowledge such as routing tables, spanning trees, etc. Thus, most of the geographic routing solutions designed for static sensor networks and available for MANETs today are poorly applicable for the disaster-incident case. On the other hand, local minimum of the geographic routing often appears near large physical obstacles (especially, of concave shapes) such as man-made (e.g., buildings) or natural (e.g., close to lakes or ponds). Figures 3.1a and 3.1b illustrate these potential physical obstacles.

(a)

(b)

Figure 3.1: Satellite imagery examples of various physical obstacles which can be used for training purposes including both man-made e.g., buildings (a) and natural e.g., lakes or ponds (b).



(a) (b) (c) (d)

Figure 3.2: Satellite imagery of Joplin, MO area including Joplin High School (*top row*) and Joplin Hospital (*bottom row*) buildings before (a,c) and after (b,d) tornado damages on May 22^{nd}, 2011: we can see how captured on satellite images information about *size* and *location* of buildings (and other physical objects) was slightly impacted by the disaster incident.

The information about physical obstacles can be obtained directly from the satellite maps (commonly available for geospatial cloud analytics) which contain information about the disaster-incident area. Figure 3.2 shows maps of Joplin, MO area before and after tornado damages that occured on May 22^{nd}, 2011. The tornado response imagery of Joplin, MO is available at [109]. We can see how information (e.g., size and location) of the Joplin High School (see Figures 3.2a and 3.2b) and the Joplin Hospital (see Figures 3.2c and 3.2d) buildings (and other physical objects) are slightly impacted by the tornado disaster incident. Thus, we can benefit even from currently available maps of the disaster incident area by addressing the following problems - (i) how to extract information about potential physical obstacles (e.g., buildings, lakes, etc) from the available maps of the disaster scene; and (ii) how to use this knowledge within geographic routing to improve its goodput (i.e., application layer throughput) sufficient for the *real-time visual situational awareness*.

Note that in this thesis, we do not focus on the first problem of finding the best (i.e., the most accurate) approach for the obstacle detection on the satellite maps. Instead, we address the second problem — how to utilize obtained physical obstacle knowledge in the geographic routing to support real-time visual situational awareness as well as seamless data-intensive processing at the infrastructure edge under challenging disaster-incident conditions. Our AGRA architecture is discussed in Section 6.1.

3.2 Repulsive Field Model

In this section, we describe a theoretical solution that can be used to incorporate the physical obstacle's knowledge extracted from the maps within the geographic routing. We will use an approximation of this theory as a design principle for our algorithms described in subsequent sections.

Let us consider a wireless network with nodes uniformly distributed over the continuous $\mathbb{R}^2$ (or $\mathbb{R}^3$ in 3-dimensional case) plane, with limited radio range. Let us also assume that our wireless network is static and does not have any obstacles (voids). In a greedy forwarding algorithm, nodes need to be aware of the (euclidean) coordinates of the destination as well as of all their neighbors [34, 35, 40, 41, 39]. Packets are then forwarded to the neighbor closest to the destination.

We model greedy forwarding based on an analogy from the electrostatics literature: specifically, we model a packet as a positive electric charge (or test charge) and its movement from the source with an electrostatic field created by the destination (with point negative charge). Packets are forwarded towards lines of force of the electrostatic field till they reach the destination. The potential field φ generated by the negative charge at the destination on the test charge is modeled as:

$$\varphi = -\frac{Q}{r} \tag{3.1}$$

where r is a distance between the node which currently holds a packet and the charge located at the destination, and Q represents the intensity of such a charge. Following the laws of electrostatics, each node forwards its packets to the neighbor with the lowest electrostatic energy i.e., the node with the lowest electrostatic potential. When there are no obstacles between the source and destination, nodes always forward packets to the node closer to the destination. In this thesis, we refer to this potential field generated with Equation 3.1 as the *attractive* field.

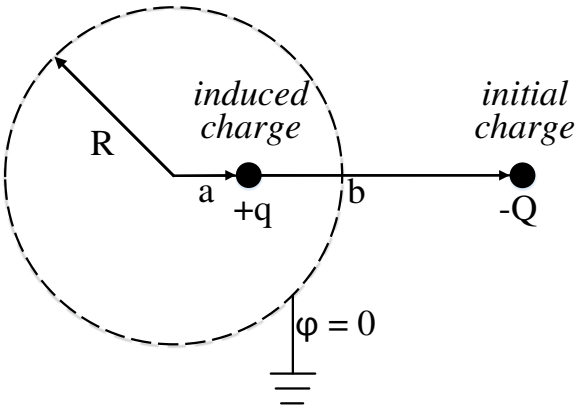


Figure 3.3: Electrical images method: a single negative point charge induces a single positive point charge (image) inside a grounded sphere making its surface equipotential (zero potential).

In presence of obstacles, to generate the repulsive potential, we first ground each obstacle region making its surface equipotential (zero potential) by generating additional charges within this region. We then sum to the main potential described in Equation 3.1, an additional potential created by each induced charge. In the rest of this section we first show how nodes would compute a potential for a single grounded spherical obstacle region. We then extend our computation to a network with multiple obstacles of arbitrary shapes. We finally show how our proposed *repulsive* field model approximates the shortest path.

3.2.1 Electrical Images Solution for a single Spherical Obstacle

To "ground" a single spherical obstacle we use the electrical images method [75]. In particular, we induce a single positive point charge inside the obstacle as shown in Figure 3.3. The potential field generated by such point charge is modeled by the following equation:

$$\varphi = -\frac{Q}{r} + -\frac{q}{|\vec{r} - \vec{b} + \vec{a}|} \tag{3.2}$$

where $\vec{b}$ is a vector modeling the distance between the center of the sphere and the destination node, and $\vec{a}$ is a vector modeling the distance between the center of the sphere and the induced charge. The value of the induced charge q is then: $q = \frac{R}{b}Q$, where R is the radius

of the sphere. The final location of the induced charge $\vec{a}$ is instead: $\vec{a} = \frac{R^2}{b^2}\vec{b}$.

3.2.2 Green's Function Solution for Obstacles of Arbitrary Shape

Let us now consider a more general case of a network with multiple obstacles of arbitrary shape. To model the positive potential generated by such scenario, we use the Green's function [75]:

$$G = -\frac{Q}{r} + \chi \tag{3.3}$$

where χ is the potential induced by the charges on the obstacles within the network graph. We assume the origin of the coordinate system at the destination node. The effect of such a positive potential is then summed to the negative potential field generated by the point negative charge at the destination to generate the line of force, and hence the trajectory of the packet. In this thesis, we refer to the potential field generated with Equation 3.3 (or its approximation) as the *repulsive* field. Thus, we have the following result:

Fact 1. *Given a network with a system of grounded conductors and a single point charge at the destination, we can emulate a unique potential field without a local minimum.*

Proof. Let us assume by contradiction that we have a local minimum created by the Green's function potential field for several grounded obstacle surfaces and a negative point charge located at the destination. Moreover, let us assume by contradiction that this potential field is also not unique. Having a local minimum implies that all lines of force have to be directed inwards of the local minimum, which is possible if and only if there is a point or a surface that creates an additional negative potential field besides the destination (see Figure 3.4). From the Green's Reciprocation Theorem [75] we derive the closed-form expression for a potential between any point a located on the grounded surface and a single point charge located at the destination as follows:

$$\varphi(a) = -\oint_S \varphi_S \vec{\nabla} G(a)\, dS \tag{3.4}$$

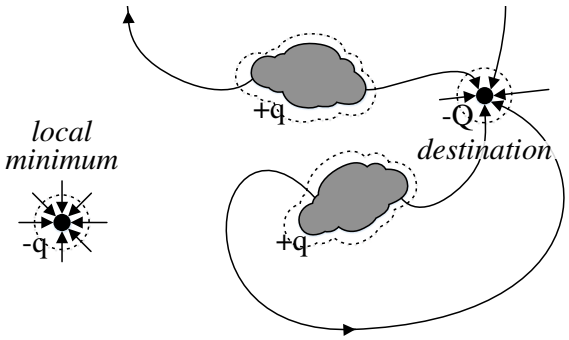


Figure 3.4: A potential field has a local minimum if there is a point or a surface which creates an additional negative potential field somewhere else besides the destination, so that the lines of force are directed inwards the local minimum.

where φ_S is the potential on the surface S and φ is the potential generated by a point charge located at the destination. Note that Equation 3.4 can be rewritten in a discrete form as: $\varphi(a) = -\sum_{j=1}^{n} \varphi_{jS} q_{jS}$. From the Equation 3.4 and its discrete form we conclude that the potential field created by some obstacle of arbitrary shape has always an opposite sign of the potential field created by the point charge. This means that an additional potential field is always positive when the point charge located at the destination is negative. In addition, based on the Green's function Uniqueness Theorem [75] such solution is unique. ■

Corollary 1. *Repulsive greedy forwarding always avoids local minimum.*

Proof. Within a repulsive field, each node forwards packets to the neighbor with minimum emulated potential energy. Based on Fact 1, the line of force of such potential function can lead only to the destination, or to infinity in case of a disconnected network. Hence, if a path between the source and the destination exists, then packet delivery by *repulsive* greedy forwarding is guaranteed. ■

3.2.3 Path Stretch Approximation Bound

The goal of this subsection is to show a bound on the path stretch obtained using *repulsive* greedy forwarding. To this aim, we first show that the maximum path stretch arises only

in presence of a single spherical obstacle; we then estimate such stretch upper bound using properties of the electrostatic potential field.

Fact 2. *The maximum path stretch of repulsive forwarding arise when both source and destination are located at the two extreme points of a single obstacle's diameter.*

Proof. Since the potential field created by an obstacle is inversely proportional to the distance r to this obstacle (Equation 3.2), the closer the shortest path lies to an obstacle region, the greater the strength of the obstacle's potential field on such packet. This means that the path stretch is as high as the path gets closer to the obstacle. In the worst-case, the shortest path length equals to half of the obstacle's perimeter, e.g., when the source and the destination are located on the opposite sides of the obstacle's diameter. From the Green's function [75] we know that the potential field generated by the grounded obstacle is directly proportional to its volume. This means that the greater is the obstacle's volume (or area) and the shorter is its perimeter, the higher is the possible path stretch. Finally, the more obstacles we have in the forwarding region, the weaker will be their potential fields due to secondary induced charges, since the obstacle potential fields weaken each other. Due to this fact and based on the isoperimetric property of a sphere (circumference) [110], we conclude that *repulsive* greedy forwarding shows a maximum path stretch only in the presence of a single spherical obstacle, when both source and destination are located at opposite sides of the obstacle's diameter. ■

Corollary 2. *Repulsive forwarding has a* 3.291 *path stretch approximation bound with respect to the shortest path.*

Proof. Based on Fact 2, the *repulsive* greedy forwarding maximum path stretch arise when a single spherical obstacle has both source and destination located at opposite sides of its diameter. The Green's function (see Equation 3.3) of this obstacle is $G = \frac{q}{r_1} - \frac{Q}{r_2}$ as shown in Figure 3.5a, where Q is a point charge located at the destination; $q = \frac{RQ}{R+\frac{l}{2}}$ is a point induced charge located within the obstacle (see Equation 3.2), and l is the distance between

them. Since $l \ll r$, using the cosine theorem and the Taylor series expansion on l we have:

$$\begin{cases} r_1 = \sqrt{r^2 + \frac{l}{4}^2 - rlcos\phi} = r - \frac{l}{2}cos\phi + O(l^2) \\ r_2 = \sqrt{r^2 + \frac{l}{4}^2 + rlcos\phi} = r + \frac{l}{2}cos\phi + O(l^2) \end{cases} \tag{3.5}$$

Similarly, q can be written as $q = Q - \frac{l}{2R}Q + O(l^2)$, and hence, using Equation 3.5 the Green's function G in its first l approximation can be rewritten in polar coordinates as:

$$G(r, \phi) = \frac{Q - \frac{l}{2R}Q}{r - \frac{l}{2}cos\phi} - \frac{Q}{r + \frac{l}{2}cos\phi} = \frac{Ql(cos\phi - \frac{r}{2R})}{r^2} \tag{3.6}$$

Note that when $r \ll R$, the potential field in Equation 3.6 becomes a potential field of a point dipole. Moreover, on the obstacle's circumference (e.g., when $r = 2Rcos\phi$), Equation 3.6 equals to 0, which is also expected since a grounded conductor (an obstacle) has a zero-potential surface.

Initially, packets are repelled away from the obstacle along the source-destination line until they reach the minimum potential point, which we can find by differentiating Equation 3.6 with respect to r and subsumed $\phi = 0$ as following:

$$\frac{dG(r, 0)}{dr} = -\frac{2Ql}{r^3} + \frac{Ql}{2Rr^2} = 0 \tag{3.7}$$

Solving Equation 3.7 gives us $r = 4R$, and hence, initially packets are repelled away from the obstacle on $L_0 = 2R$ distance. If we model a path with the closed-form of packet trajectory obtained by *repulsive* greedy forwarding, we can find its residual length from the gradient symmetry of the electrostatic potential field [75]:

$$\frac{dr}{\nabla_r G(r, \phi)} = \frac{rd\phi}{\nabla_\phi G(r, \phi)} \tag{3.8}$$

Further, by subsuming Equation 3.6 to Equation 3.8 we obtain the following differential

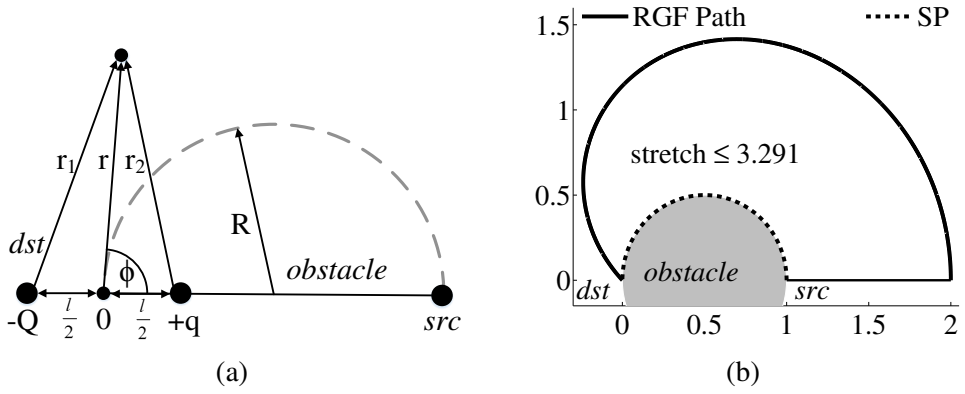


Figure 3.5: (a) Potential field in presence of a single grounded sphere in polar coordinate system: when the destination locates very close to the surface of a sphere, the induced charge creates a potential field of a point dipole. (b) Using electrostatic potential field properties and numerical calculations we estimated the worst-case *repulsive* greedy forwarding (RGF) stretch of the shortest path (SP) as $\approx 3.2907221 \pm 10^{-7}$.

equation:

$$\dot{r} + r\frac{1}{2Rsin\phi} - 2ctg\phi = 0 \tag{3.9}$$

where $\dot{r} = \frac{dr}{d\phi}$. Moreover, $r(0) = 4R$ (see Equation 3.7).

As both shortest path and *repulsive* greedy forwarding path lengths are proportional to the obstacle's size, without loss of generality we can assume unitary obstacle diameter. Hence, solving Equation 3.9 with $R = \frac{1}{2}$ gives us the following packet trajectory closed-form:

$$r = \frac{2\phi}{tg\frac{\phi}{2}} - 2 \tag{3.10}$$

The maximum path stretch is then equal to $\frac{L}{\pi R} = \frac{2L}{\pi}$ where $L = \int_{\phi_1}^{\phi_2}(r^2 + \dot{r}^2)^{\frac{1}{2}}d\phi + L_0$ is a length of the found trajectory. Note that $\phi_1 = 0$, and $\phi_2 \approx \frac{3\pi}{4}$ which can be found by subsuming $r = 0$ to Equation 3.10, i.e., by solving $\phi = tg\frac{\phi}{2}$.

Solving the integral using numerical calculations, we estimated the length of the trajectory to be $L = 5.1690541 \pm 10^{-7}$. The maximum path stretch is then $\approx 3.2907221 \pm 10^{-7}$ (see Figure 3.5b). We then conclude that - *repulsive* greedy forwarding is an approximation algorithm for the shortest path. ■

In the rest of this chapter, we first present a practical algorithms which incorporates

repulsive field model. Using numerical and event-driven simulations, we then show how a path stretch minimization with proposed algorithms improves an overall network goodput.

3.3 Practical Repulsive Forwarding

In the previous section, we have shown how to use physical obstacle information within geographic routing to avoid the local minima by greedy forwarding packets as if they are hypothetically immersed in a potential field generated by a point charge at the destination. Obstacles were modeled as conductors with zero potential surface resulting in additional repulsing fields. Herein, we describe how we approximate the Green's function to capture the potential field generated by multiple randomly shaped obstacles. We first compute our approximation assuming global knowledge of all obstacles at every node. Next, we extend our solution to a local knowledge case: nodes are only aware of obstacles present at some specified distance called "repulsion zone". Then, we discuss how our forwarding algorithm can be coupled with existing stateless local minimum recovering scheme called "Pressure" proposed in [34] for a guarantee delivery [1].

Why do we need an approximation? Our repulsion field builds upon the continuous $\mathbb{R}^2$ (or $\mathbb{R}^3$) plane; a discrete node distribution instead creates discontinuity of potential fields which can form artificial local minima. For example, nodes may be scattered in a way of creating artificial voids (i.e., not radio-covered space without physical obstacles). Moreover, to find a closed-form of the potential induced on every node on the network, the Green's function requires an integration over the obstacles' surface with respect to every other obstacle, including the impact of secondary charges. Solving such an integral (at every forwarding decision) may be unfeasible, due to complex obstacle shapes or by the dynamic nature of the network. The mirror effect further complicates the computation of the repulsive potential field induced by the obstacles: when an object is located between

[1] As we show later guaranteed delivery is possible only if there is no path length limit such as e.g., Time To Live (TTL).

two mirrors, infinite number of images (induced charges) appear. Similarly, secondary charges recursively induce progressively weaker additional fields, to be included in the Green's function solution.

Next, we show the impact of the secondary charges by approximating any obstacle shape to a circle (a sphere) with a point charge located on its center of mass. For our approximation of the obstacle electrostatic field we require:

Sign: the potential induced by each obstacle region has to be positive, i.e., each obstacle is replaced with a positive point charge to repulse a packet.

Direction: the potential induced has to be greater inside, and smaller outside than a potential field created by a negative point charge located at the destination. This is necessary to correctly drive the packet in direction of the destination.

Intensity: the total potential filed must be equal to 0 at ∞.

The above requirements allow the greedy forwarding algorithm to use a gradient descent on the repulsion field to converge to the destination by forwarding packets to the neighbor whose electrostatic energy is minimum. We describe the local minima recovering mechanism in Section 24.

Obstacles' shape approximation. We approximate the potential field of an obstacle by circumscribing it to a circle (or sphere) j to capture the worst path stretch it can cause (see Fact 2). Having the set of obstacle's pixels the detector (see Section 3.1) computes the center and the radius of the circumscribing circle. We locate the center of the circle with the center of mass of such an obstacle. To locate the coordinates of the center of mass C_j we average the coordinates of the N pixels S_i of the obstacle as follows:

$$\begin{cases} x_{C_j} = \frac{\sum_{i=1}^{N} x_{S_i}}{N} \\ y_{C_j} = \frac{\sum_{i=1}^{N} y_{S_i}}{N} \end{cases} \tag{3.11}$$

We then assign to the radius of the circumscribing circle R_j, the distance between the center

C_j and the furthest (border) pixel, which can be computed as follows:

$$R_j = \max_{i=1,N} dist(C_j, S_i). \tag{3.12}$$

Remark: *Note that instead of using an AI-based obstacle detectors over satellite imagery at edge clouds, the proposed Repulsive field can be used in conjunction with existing dynamic obstacle detection algorithms (e.g., localization techniques based on node's signal strength) running in MANETs [111]. To this aim, Equations 3.11 and 3.12 can use coordinates of obstacle's border nodes. The cons of a dynamic obstacle detection is the overhead as well as the dependency on the network awareness, which can be aggravated by the geographic routing performance under node failure and mobility conditions.*

Obstacle's Potential Approximation. To neglect the impact of self-induced charges we approximate the potential field of the obstacle with a decaying potential whose strength magnitude is a parameter n. To approximate the potential field of the circumscribed circle, we place a positive point charge q_j in the obstacle center C_j. We approximate the potential field generated by the obstacle with a potential whose strength diminishes with the r_j^n:

$$\varphi_j = \frac{q_j l^{n-1}}{r_j^n} \tag{3.13}$$

where l^{n-1} is a normalizing constant with units of length elevated to the power of $n-1$. To minimize the impact of the secondary induced charges in presence of several obstacles, we could use $n \geq 1$. Note how a potential of a negative point charge located at the destination always depends merely on the distance r_d ($n = 1$).

3.3.1 Computing Electrostatic Potential with Global Knowledge of Obstacles' Location

Let us assume that all nodes are aware of both the center and the radius of each obstacle j, $< C_j, R_j >$. This assumption is suitable for routing schema in which the global knowledge

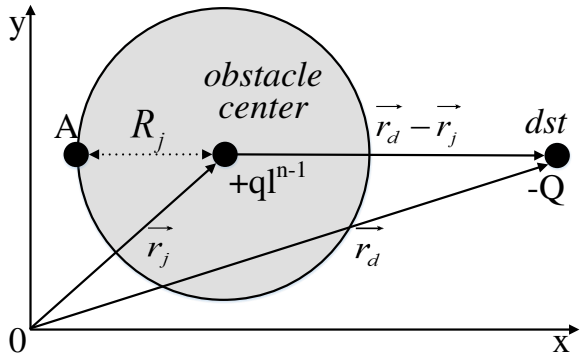


Figure 3.6: Use of border conditions in A to find values $q_j l^{n-1}$.

of the topology is available, or in the case of a static known man-made or natural obstacle, such as a building or a pond. With such information, we can compute the electrostatic potential at any node e as:

$$\varphi_e = -\frac{Q}{|\vec{r_d} - \vec{r_e}|} + \sum_{j=1}^{M} \frac{q_j l^{n-1}}{|\vec{r_j} - \vec{r_e}|^n} \tag{3.14}$$

where $\vec{r_d}$ is a radius vector directed towards the destination, $\vec{r_j}$ is a radius vector directed towards the obstacle's center C_j, and $\vec{r_e}$ is a radius vector directed towards the node e. To compute $q_j l^{n-1}$ we equate the electrical fields created by the obstacle j and the destination d on the obstacle's border, so that they are equal in magnitude and opposite in direction, i.e., $\vec{\nabla}\varphi_d = -\vec{\nabla}\varphi_j$, where:

$$\begin{cases} \vec{\nabla}\varphi_d = -\frac{Q}{|\vec{r_d} - \vec{r_A}|^3}(\vec{r_d} - \vec{r_A}) \\ \vec{\nabla}\varphi_j = \frac{n q_j l^{n-1}}{|\vec{r_j} - \vec{r_A}|^{n+2}}(\vec{r_j} - \vec{r_A}) \end{cases} \tag{3.15}$$

From Equation 4.15 we derive the dependence of $q_j l^{n-1}$ by Q (see Figure 3.6):

$$q_j l^{n-1} = \frac{Q R_j^{n+1}}{n(|\vec{r_d} - \vec{r_j}| + R_j)^2} \tag{3.16}$$

3.3.2 Computing Electrostatic Potential without Global Knowledge of Obstacles' Location

In this section we describe a simple method to compute forwarding decisions based solely on local knowledge of obstacles' location. This is motivated by the need to reduce network overhead and to optimize memory and storage needed to deal with the propagation and storage of obstacles information (especially in case of large-scale networks). In our evaluation section, we quantify how such local knowledge is enough for a performant greedy forwarding strategy with minimum path stretch. Note how, since the electrostatic field intensity diminishes with distance, when computing the potential, forwarding nodes do not need to consider obstacles "far away".

To find the distance at which the obstacle's contribution to the total potential field is still significant, i.e., the obstacle's *repulsion zone* R_z, we first decompose the second term of Equation 3.14 with a Taylor series expansion obtaining:

$$f(R) = \frac{q_j l^{n-1}}{R^n} - \frac{n q_j l^{n-1}}{R^{n+1}}(r - R) + O((r - R)^2) \tag{3.17}$$

and then we neglect the first two terms of higher order and substitute R_z to r in Equation 4.16. The radius of repulsion zone is therefore:

$$R_z = (1 + \frac{1}{n})R \tag{3.18}$$

Thus, nodes need only to store obstacles located closer than R_z. Once a node moves further away from an obstacle's R_z, it can remove its states concerning that obstacle. On contrary, when a new node arrives within the obstacle's repulsion zone R_z, the information about such obstacle should not be used. As we show in Section 3.4, such information can be exchanged among neighbors during their regular position beaconing communication with negligible additional overhead (i.e., using the same packet).

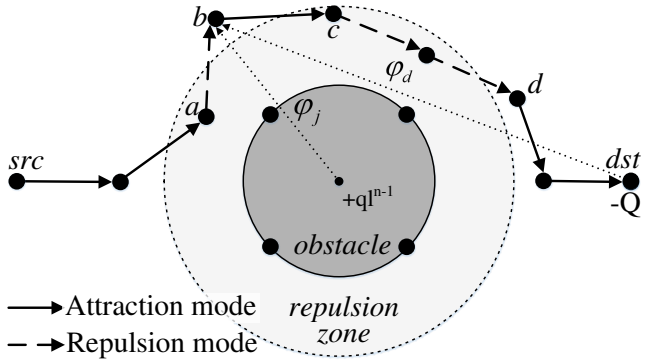


Figure 3.7: Attractive/Repulsive Greedy Forwarding (ARGF) example: ARGF starts in *Attraction* mode until it reaches the *repulsion zone*; at node b (located outside the repulsion zone), ARGF returns back to its *Attraction* mode. ARGF can again switch to the *Repulsion* mode with guaranteed progress towards destination. At node d, ARGF returns to the *Attraction* mode and continue forwarding in this mode to destination.

3.3.3 Greedy Forwarding Algorithms

Note how, due to discrete node distribution and since we approximate the Green's function of the real obstacles to a potential of spherical regions, Equations 3.14 and 3.16 do not guarantee avoidance of a local minimum. To this end, we first propose a solution that alternates forwarding in both attractive and repulsive fields, *viz.* Attractive Repulsive Greedy Forwarding (ARGF), to increase the chance of escaping or avoiding all local minima and deliver a packet with the minimum path stretch. To guarantee 100% packet delivery, we then extend ARGF with stateless Pressure local minimum recovery scheme proposed in [34]. The resulting Attractive Repulsive Pressure Greedy Forwarding (ARPGF) applies recovery when in pressure mode to both attractive and repulsive fields (details in Section 24).

Attractive/Repulsive Greedy Forwarding

Algorithm 1 outlines how each node forwards packets using the Attractive Repulsive Greedy Forwarding (ARGF) strategy, when either local or global information about obstacles are known. Figure 3.7 illustrates the following forwarding process: upon receiving a packet to be forwarded, node e checks if the packet should proceed further in *Repulsion* mode or in *Attractive* mode. To this end, ARGF first check that e in *repulsion zone* R_z and that previously found potential in *Repulsion* mode attached to the packet P_φ_{rep} (which

is initially set to max value) is greater than current potential of e (line 3). This statement is important to ensure that packets progress in *Repulsion* mode towards the destination without possibility of returning to previous local minimum of repulsive field. Thus, if $P\_\varphi_{rep} > \varphi_{total}(e, P, e.\vec{C}, e.\vec{R})$, e stores its potential in packet header (line 4) and computes neighbors $Nbrs$ potential φ_{total} (line 5) using its information about known obstacle centers $e.\vec{C}$ and their radius $e.\vec{R}$ via Equations 3.14 and 3.16.

If no neighbor n found has a potential φ_{total} lower than node e potential, or if e is not in R_z, e switches to an *Attraction* mode, where it computes its neighbors' potential φ_d using Equation 3.1 (line 12). If both *Repulsion* and *Attractive* modes are unavailable to find next hop, ARGF returns a potentially detected obstacle condition and terminates.

Algorithm 1: Attractive/Repulsive Greedy Forwarding

```
   /* Upon receiving a packet P at node e                                    */
1  if e ≠ dst then
2      next ← NIL
3      if e ∈ R_z and P_φ_rep > φ_total(e, P, e.C⃗, e.R⃗) then
           /* Repulsion mode                                                 */
4          P_φ_rep ← φ_total(e, P, e.C⃗, e.R⃗)
5          n ← arg min_{n∈Nbrs(e)} φ_total(n, P, e.C⃗, e.R⃗)
6          if φ_total(n, P, e.C⃗, e.R⃗) < φ_total(e, P, e.C⃗, e.R⃗) then
7              next ← n
8              forward(P, next)
9          end
10     end
11     if next == NIL then
           /* Attraction mode                                                */
12         n ← arg min_{n∈Nbrs(e)} φ_d(n, P)
13         if φ_d(n, P) < φ_d(e, P) then
14             next ← n
15             forward(P, next)
16         else
17             exception ("ARGF faced a local minimum")
18             alert ("Potentially unknown obstacle detected")
19             terminate
20         end
21     end
22 else
23     terminate
24 end
```

Attractive/Repulsive/Pressure Greedy Forwarding

Although empirically (as we show in Section 3.4) our ARGF outperforms traditional Greedy Forwarding in terms of packet delivery, it does not theoretically guarantee 100% packet delivery. To this end, we devised Attractive Repulsive Pressure Greedy Forwarding (ARPGF), which builds upon a known *Gravity-Pressure* scheme that has been shown to provide guarantee packet delivery [34].

Algorithm 2: Attractive/Repulsive Pressure Forwarding

```
   /* Upon receiving a packet P at node e                                    */
1  if e ≠ dst then
2      next ← NIL
3      if e ∈ R_z and P_φ_rep > φ_total(e, P, e.C⃗, e.R⃗) then
           /* Repulsion mode                                                 */
4          P_φ_rep ← φ_total(e, P, e.C⃗, e.R⃗)
5          n ← arg min_{n∈Nbrs(e)} φ_total(n, P, e.C⃗, e.R⃗)
6          if φ_total(n, P, e.C⃗, e.R⃗) < φ_total(e, P, e.C⃗, e.R⃗) then
7              next ← n
8              forward(P, next)
9          end
10     end
11     if next == NIL and P_φ_attr > φ_d(e, P) then
           /* Attraction mode                                                */
12         P_φ_attr ← φ_d(e, P)
13         n ← arg min_{n∈Nbrs(e)} φ_d(n, P)
14         if φ_d(n, P) < φ_d(e, P) then
15             next ← n
16             forward(P, next)
17         else
18             alert ("Potentially unknown obstacle detected")
19         end
20     end
21     if next == NIL then
           /* Pressure mode                                                  */
22         visits_min ← min_{n∈Nbrs(e)} P_visits(n)
23         Candidates ← {n ∈ Nbrs(e) and P_visits(n) == visits_min}
24         n ← arg min_{n∈Candidates} φ_total(n, P, e.C⃗, e.R⃗)
25         P_visits(n) ← P_visits(n) + 1
26         next ← n
27         forward(P, next)
28     end
29 else
30     terminate
31 end
```

Algorithm 2 outlines how each node forwards packets using Attractive Repulsive Pressure Greedy Forwarding (ARPGF): the algorithm also starts by alternating *Repulsive* and

Attractive fields in ARGF, when it needs to forward packets. However, similarly to the last known potential of *Repulsion* field φ_{rep}, it also saves the last known potential in the *Attractive* field variable φ_{attr} (line 12) to ensure in a possibility of the progress in *Attractive* mode e.g., after resuming from *Pressure* mode. When both *Attractive* and *Repulsive* forwarding fail to find next node for forwarding, ARPGF switches to *Pressure* mode (line 21).

The key idea behind recovery in *Pressure* mode is to forward packet to the closest to the destination neighbor among the least visited neighbors (line 23).

3.4 Performance Evaluation

In this section, we establish the practicality of our electrostatics-based approach by evaluating its performance in several scenarios that result into the following salient findings:
$(i.a)$ *Local obstacles knowledge is enough.* Our repulsive greedy forwarding approach is not affected by a lack of global knowledge on obstacles' position.
$(i.b)$ *Local obstacles introduce negligible storage and no network overhead.* To maintain a local knowledge on obstacles, our routing protocols only requires < 0.25 KB of storage space, and hence, that information can be piggybacked and propagated with the keep-alive beaconing message to update their position at no (or negligible) network overhead.
(ii) *Our ARPGF outperforms related stateless greedy forwarding solutions [35, 34] in terms of delivery ratio, and the required information to run it, which can fit in available IP packet header space with* 99% *probability (*$i.e$*, its overhead is extremely low).*
(iii) *The repulsive field (and hence both ARGF and ARPGF) improve network's goodput in challenged disaster incident wireless edge networks*[2]. By reducing a path stretch due to a physical obstacles knowledge, ARGF (and hence ARPGF) results into a higher network throughput than related solutions. The first two results emerge from our numeric simulations, while we found our third result analyzing using more realistic ns-3 event-driven

[2]Improvement were observed when both coordinates and radius of physical obstacles are known (see Section 3.4.2).

simulations (see Subsections 3.4.1 and 3.4.2).

3.4.1 Performance Tuning under Static Obstacles of Complex Concave Shapes

Simulation Settings. Our Java-based simulation environment is composed by an Ubuntu OS GNU/Linux $x86\_64$ machine with an *Intel(R) Xeon(R)* processor with CPU 2.1 GHz and 1GB RAM. We generate a 1 km^2 area and place nodes into each 10x10 m cell (for a total of 10K nodes). To remove the "Unit Disk" graph assumption, we set the radio range of each node from 50 to 40 m, unless stated differently. We then applied the random graph generation model $G(n, 1-p)$ [34, 41] with probability $p = 0.05$. With this parameters when two nodes are within the reciprocal radio range, there is a 5% probability that one of these nodes is not detected by the other. We refer to this condition as lack of symmetrical link assumption, that in turn leads to a network asymmetrical connectivity.

We generate circular obstacles with a radius ranging from 10 to 100 m in random locations. When overlapping, such obstacles create complex concave shapes, which stress greedy forwarding to the limit [40]. We run our simulations with 0, 10, 30, 50 and 100 obstacles that occupy $\approx 0\%$, 10%, 25%, 40% and 60% of the available routing space, respectively.

Remark: *Note that some of the recent similar solutions demonstrate valuable performance degradation after only* 30% *of obstacles occupancy [41]. In a disaster scenario, this would be common and such performance degradation unacceptable.*

After setting up the environment, we attempt to deliver traffic among 1000 random pairs $< src, dst >$. In this scenario, our main goal is to stress our greedy forwarding algorithms with obstacles of complex concave shapes, and hence, we do not use node mobility, as it leads to a frequent network partitioning under high obstacle occupancy which hides greedy

forwarding algorithms' potential.[3] For the same reason, we do not generate obstacles at the area edges. All our results show 99% confidence interval over 50 trials, and our randomness lies in both the source-destination pairs and the formed network topologies.

Comparison Methods and Metrics. To empirically evaluate which potential field best approximates an obstacle of arbitrary shape, we tested the performance of both ARGF and ARPGF under different potential field attenuation orders n. We then leverage our finding ($n = 2$ for ARGF and $n = 1$ for ARPGF) in our other experiments.

We compare our ARGF and ARPGF algorithms with three other stateless greedy forwarding exiting approaches, that is, the original Greedy Forwarding (GF) algorithm (also known as compass routing [36]), a *face routing* algorithm, GPSR [35] and a *gravity pressure forwarding* algorithm GPGF [34]. In addition, we also compare ARGF algorithm coupled with GPSR using a face routing recovery policy, which we call ARPSR. The related solutions are compared across two metrics: the packet delivery success ratio, i.e., the number of delivered packets divided by the total number of attempted delivery (counting only cases in which a path for $< src, dst >$ pair exists), and the average path stretch, calculated as the average path length ratio of delivered packets and the shortest paths computed with a simple Breadth-First Search algorithm. Finally, we also compare packet header sizes, which are dynamic for ARPGF and GPGF.

(i.a) Local obstacle knowledge is enough. During the potential field formation w.r.t. the different attenuation order n, we found that based on local obstacles knowledge, both ARGF and ARPGF perform similarly (aside from a better path stretch for $n \leq 2$) to the case when all obstacles are known a priori (Figures 3.8a,3.8b,3.8c and 3.8d). A path stretch difference under $n \leq 2$ is due to a potential field approximation inaccuracy: we ignore an obstacle's mutual influence by disregarding the secondary induced charges (creating artificial local minima), while local obstacles knowledge mitigates this problem. We can see

[3]Note how mobility does not affect stateless greedy forwarding under the assumption that $t_1 \ll t_2$, where t_1 is time needed to greedy forward a packet to the next hop, and t_2 is time needed for the node to move out of its neighbor radio range. To convince the reader that this is a practical assumption, we later apply (high) mobility in our event-driven simulations.

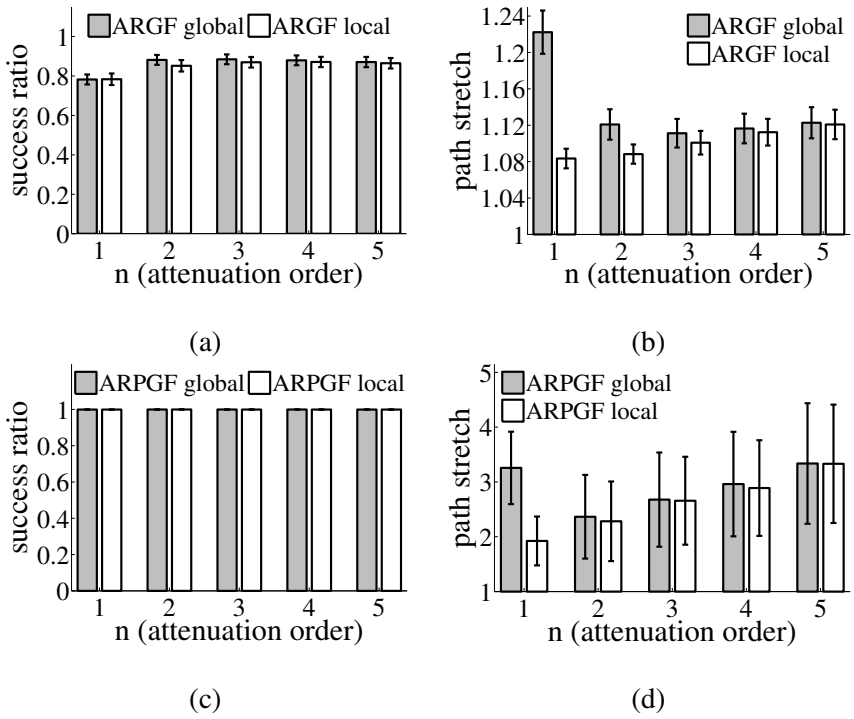


Figure 3.8: Ignoring the obstacle mutual electrostatic influence during a potential field computation leads to a lower (a, c) success ratio and/or (b, d) higher path stretch (attenuation order $n < 2$). Performance of both ARGF (first row) and ARPGF (second row) does not improve for attenuation order $n \geq 2$.

how both global and local versions of ARGF show the lowest path stretch and the highest success ratio for $n = 2$, which match with the attenuation order of a point dipole, and therefore in line with our theoretical model (see Equation 3.6). At the same time, the local version of ARPGF shows the lowest path stretch when $n = 1$, as in this case we gain the best ratio between secondary induced charges mitigation and the radius of *repulsion zone*, which achieves maximum length decreasing the number of Attractive/Repulsive field alternations during the *Pressure* recovery mode. We use $n = 2$ for ARGF and $n = 1$ for ARPGF results for the rest of experiments.

(i.b) Local obstacles introduce no network overhead. To maintain information about local obstacles for our repulsive field, we need to store them and periodically exchange with the neighbors. The (3D) GPS coordinates of the obstacle location in the worst case (i.e., without converting them to grid coordinates) takes no more than 12 bytes (4 bytes for each coordinate). To store obstacle radius we need no more than 4 bytes. Finally, to

distinguish obstacles we can also store its id which takes no more than 2 bytes (up to 65K unique obstacles). To exchange that information we can use the following packet payload structure shown in Table 3.1. Figure 3.9a shows how maintenance of the local knowledge for both ARGF (with $n = 2$) and ARPGF (with $n = 1$) on obstacles requires low amount of the node storage space, i.e., even in rear cases of storing 7 unique obstacles as shown in Figure 3.9b we need $7 \cdot 18 < 256$ B (or < 0.25 KB). Thus, local obstacle information can be exchanged during a periodical node's neighbors beaconing to update their position at no additional network overhead (i.e., within a single packet).

Table 3.1: Obstacle Data Exchange Packet Payload

Obstacle ID	Center coordinates	Radius
2·n bytes	12·n bytes	4·n bytes

(ii) ARPGF can fit its data in the available IP header space. Our simulations show how all *pressure forwarding* algorithms (i.e., GPGF and ARPGF) have a guaranteed packet delivery when there is no path length restrictions, such as a set time to leave (TTL) — see Figure 3.9c. However, both ARPGF and GPGF lead to large path stretches when obstacles occupy most of the available space, i.e., the ARPGF and GPGF average path lengths are ≈ 3 and 6 times larger than a shortest path, respectively (Figure 3.9d.) Although *Repulsive* field usage allows ARPGF to have a halved path stretch than GPGF (paths are 2x shorter), these path stretches may force large end-to-end delays and network congestions that may jeopardize applications usage. We can also see how the packet delivery of the *face routing* algorithms (i.e., GPSR and EPSR) degrades due to asymmetrical links and variation of the nodes' radio range, leading to disconnected planar graphs (these results are in line with previous works [100]. Surprisingly, ARGF without a local minimum recovery outperforms GPSR with the local minimum recovery. These results are in line with our event-driven simulations in ns-3 (Section 3.4.2).

In the last two simulated scenarios, where 100 obstacles are present on $\approx 60\%$ of the

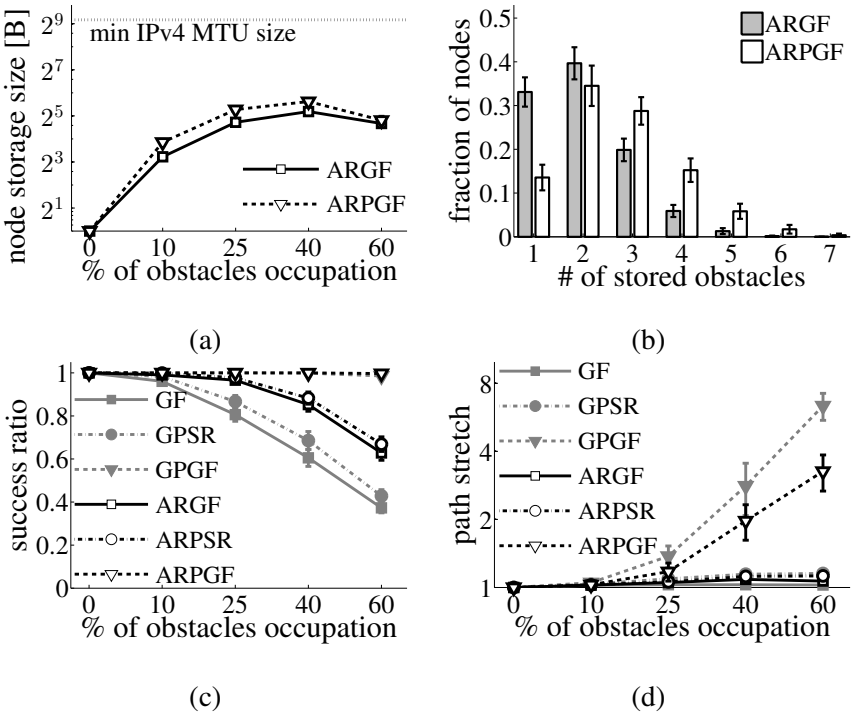


Figure 3.9: Local knowledge on obstacles maintenance for both ARGF (with $n = 2$) and ARPGF (with $n = 1$ - the largest *repulsion zone*) requires low storage (a); even for $\leq 1\%$ of nodes for the (worst-case) 40% obstacles occupation (b) which store up to 7 unique obstacles the storage space needed is $7 \cdot 18 < 256$ B. (c) All *pressure forwarding* algorithms (i.e., GPGF and ARPGF) have a guaranteed packet delivery when there is no TTL policy. Due to the asymmetrical links and no Unit Disk graph guarantees, ARGF (without local minimum recovery) outperforms GPSR — known *face routing* algorithm. These results are confirmed also by our event-driven simulations. (d) Recovering from a local minimum in GPGF and ARPGF may stretch path significantly; however, applying a *Repulsion* field to GPGF (using ARPGF) shows a halved path stretch.

area, we first limit the path length setting different TTL policies (for a maximum of 256 path length) having fixed the average node degree (nodes' radio range ranging from 50 to 40 m). We then set TTL to 128, and we vary the average node degree by reducing an interval of node's radio range distribution by 10 m, until all nodes have a minimal network connectivity radio range of 10 m. As expected, the *repulsion* field usage allows ARPGF to achieve the best packet delivery $\approx 90\%$ (Figure 3.10a) that gradually decreases as the network become less dense (see Figure 3.10c). We can also see how in 99% of the cases, both GPGF and ARPGF fits their required data for greedy forwarding in the available IP packet header space, as long as the TTL ≤ 128 (see Figures 3.10b and 3.10d). We computed the packet header sizes under the following assumptions:

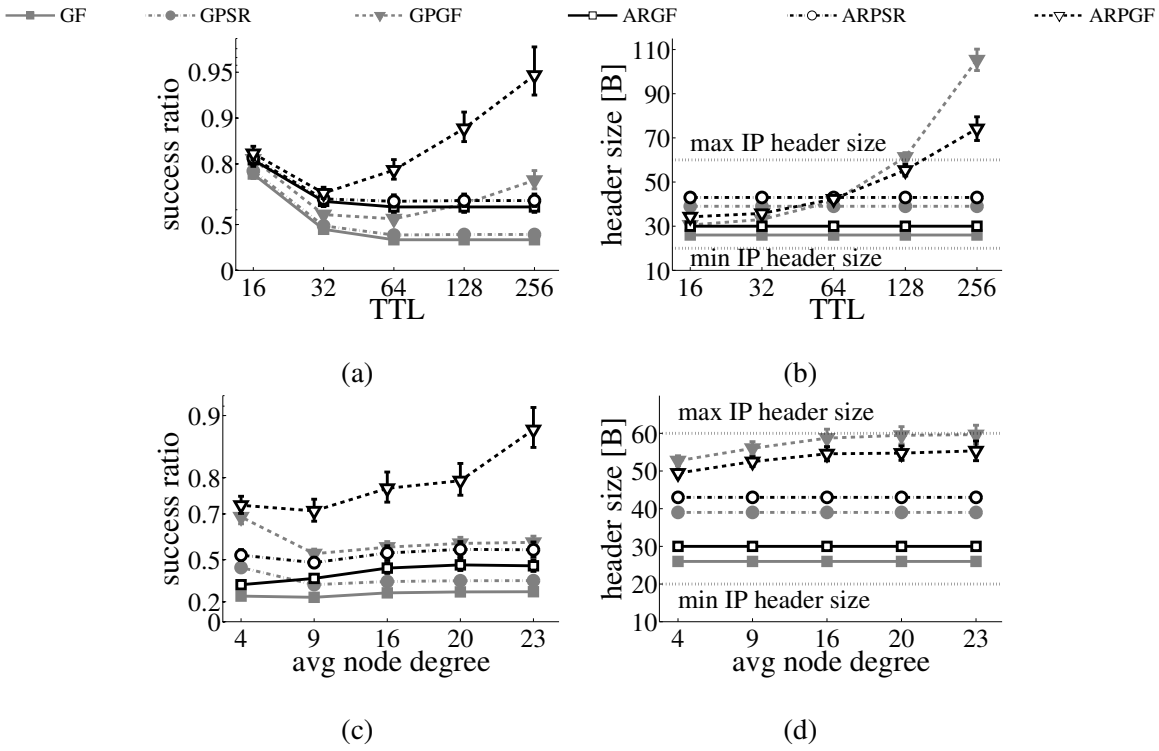


Figure 3.10: Success ratio and header size. (a,c) In presence of 100 obstacles ($\approx$ 60% of obstacles occupancy) and under various TTL path length restrictions as well as for various network densities (with TTL$=$ 128) ARPGF outperforms related greedy forwarding algorithms by showing the highest success ratio. (b,d) Under the same conditions and under some legitimate assumptions, all algorithms may utilize the available space in the IP packet header to use greedy forwarding.

- *2D or 3D node coordinates have total size of 3 bytes;* this can be achieved by e.g., converting GPS coordinates into coordinates of a finite grid which spans regions covered by the network as in [40].

- *Both node ID and number of nodes visits have size of 1byte.* As the path length is limited by the TTL, we cannot visit more than TTL-1 nodes during the *Pressure* packet recovery phase. Hence, it is possible to find a hash function of e.g., an IP address to map a node's ID between 0 and 255, with a minimum collision probability. When operating in same subnetwork, we can just use the last byte of an IP address as a node ID.

Let us now analyze the overhead of ARGF and ARPGF; with the above assumption, we need the total of 6 bytes (to store source and destination coordinates) + 4 bytes (to store

last potential in *Repulsion* mode), so only 10 bytes an extra space for ARGF protocol (see Table 3.2). ARPGF header size is then 10 bytes (as for ARGF) + 4 bytes (to store last potential in *Attraction* mode) + $2 \cdot n$ bytes (to node visits during *Pressure* recovery) = 14 + $2 \cdot n$ bytes, where n - number of unique node visits (see Table 3.3). Having 40 bytes of available space in packet header allows ARPGF track up to 13 unique nodes during *Pressure* recovery.

Table 3.2: ARGF Packet Header

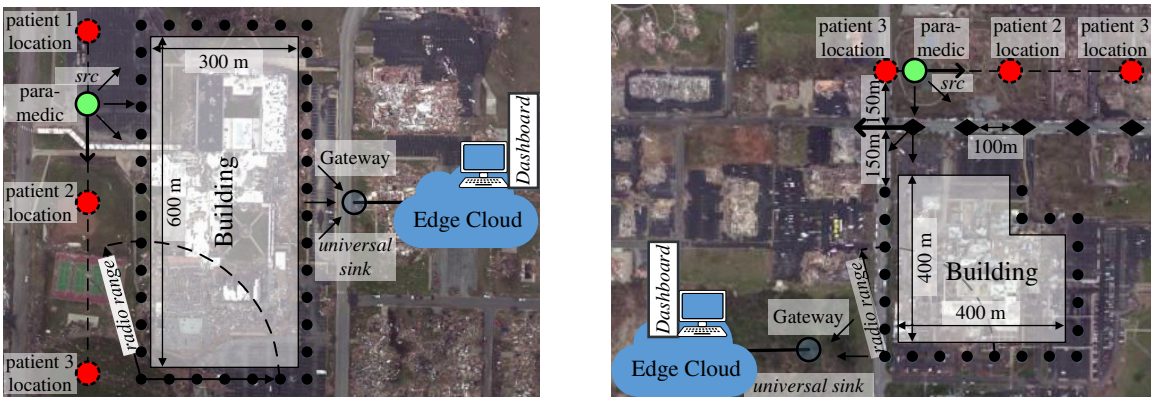
Source coordinates	Destination coordinates	Repulsive potential
3 bytes	3 bytes	4 bytes

Table 3.3: ARPGF Packet Header

Source coordinates	Destination coordinates	Repulsive potential	Attractive potential	$\text{Node}_{1..n}$ ID	$\text{Node}_{1..n}$ visits
3 bytes	3 bytes	4 bytes	4 bytes	n bytes	n bytes

3.4.2 Incident-Supporting Application Case Study Results

Simulation Settings. To evaluate the impact of the path stretch on the performance of higher layer protocols under potentially failing and mobile MANET nodes, we compared two stateless greedy forwarding algorithms — i.e., the proposed ARGF and the known *face routing* GPSR protocol [35] using the NS-3 simulator [76]. We have implemented our ARGF protocol by extending the GPSR protocol in the NS-3 with our *repulsive forwarding* mode (see Sections 3.2 and 3.3). The details of the GPSR protocol implementation can be found in [4]. Note that in this simulation we do not use the *pressure* forwarding mode to evaluate impact on throughput of the proposed *repulsion* field. To compare with stateful ad-hoc routing solutions, we use the known reactive Ad-Hoc On Demand Distance Vector (AODV) protocol [77]. We also use Hybrid Wireless Mesh Network (HWMP) protocol of



(a) (b)

Figure 3.11: AI-augmented geographic routing evaluation using recreated disaster scenes of damages by a tornado at the Joplin High School (a) and at the Joplin Hospital (b) buildings in Joplin, MO (2011): a paramedic acts as a source sending data to the gateway (universal sink) over a resilient ad-hoc network. Video streams gathered on-site are sent over a TCP session to the dashboard located in an edge-cloud for further data processing in conjunction with a core cloud. (a), we evaluate our approach under severe failures, (b) under high mobility. We assume that the information regarding a damaged buildings (e.g., its center coordinates and radius) was provided from the edge-cloud through a Gateway using proposed obstacle detector (see Section 6.1) on pre-uploaded satellite maps.

Table 3.4: Simulation Environment Settings

Topology:		**Physical/Link layers:**	
Number of nodes:	30 - 40	Frequency:	2.4 GHz
Grid placement:	50 - 150 m	Tx power:	20 dBm
1^{st} obstacle size:	600 x 300 m	Tx gain:	6 dB
2^{nd} obstacle size	400 x 400 m	Rx gain:	0 dB
Radio range:	250 m	Detection threshold:	-68.8 dBm
Avg node degree:	$\approx 3 - 10$	Delay prop. model:	CONSTANT SPEED
Overall settings:		Loss prop. model:	TWO-RAY
Node failure period:	≈ 0.033 Hz	Technology:	802.11g/s
Node failure probability:	0.05 - 0.5	Modulation:	OFDM
Mobile nodes speed:	5 - 20 m/s	Data rate:	54 Mbps
Time at each location:	180 sec	**Transport/App layers:**	
Src speed:	2.8 m/s	Transport protocol:	TCP
Simulation time:	720 - 780 s	Payload:	1448 bytes
Beaconing frequency:	1 - 4 Hz	Application bit rate:	5 Mbps

802.11s standard [78] which combines reactive (with AODV) as well as proactive routing (using the spanning tree algorithm).

We use realistic disaster scenes of damaged by tornado Joplin High School and Joplin

Hospital buildings in Joplin, MO (2011) to evaluate the performance of stateless greedy forwarding algorithms under mobility and (severe) node failures. To recreate these disaster scenes, we used the available satellite maps of Joplin, MO, tornado response imagery [109]. In our disaster-incident scenario, we simulate the 5 Mbps high-definition video streaming over a TCP connection from a heads-up display device worn by a paramedic e.g., Google Glass acting as a visual data source.

The paramedic stays for 3 minutes at each patient location and moves at a jogging speed (2.8 m/s) between these locations. The simulations are designed to cause a geographic routing to face a local minimum when the paramedic source is near the second (first scenario) or third (second scenario) patient locations. Aside from the source mobility, in the node failure simulation scenario (see Figure 3.11a), nodes around an obstacle can fail for the next 30 seconds[4] with a probability sampled from the interval $[5\%, 50\%]$ (from low to severe node failures). Under these failure conditions, the goodput degrades due to losses (e.g., caused by packet collisions) that increase with the path length or path reconstruction of the stateful routing approaches. Note that when nodes fail for continues periods, any "store and forward" solution is inadequate [112, 113]. We evaluate the impact of nodes mobility in a second simulation scenario (see Figure 3.11b), where paramedics can communicate with the gateway only through moving vehicles on the road which speed is sampled uniformly from the interval $[5, 20]$ m/s (from low ≈ 10 mph to high ≈ 40 mph mobility cases).

Finally, nodes are placed on a grid ranging from 50 - 150 m step, each node has a radio range of 250 m, and an obstacle (a building) is located approximately in the center of this grid. Each node has roughly $3-10$ neighbors for resilience purposes. Table 3.4 summarizes our simulation details.

(iii) The repulsive field improves a network's goodput. For low node failures (5%), ARGF delivers all packets with a TCP throughput of $\approx$ 3 Mbps (Figure 3.12a). GPSR instead has a 20% failure rate in delivering packets. When GPSR enters the recovering

[4]Such behavior is expected due to possibility of an intermittently available power supply, or due to a physical damage caused by rescue workers near the disaster scene.

mode it uses planarization which, in turn, can significantly stretch paths. As a result, GPRS shows lower TCP throughput (< 2 Mbps) than ARGF due to a lower congestion window size (see Figure 3.12e), and only 40% of the time it shows similar performance. Under severe node failure conditions (nodes fail 50% of the time they receive a packet to forward), we observe similar behaviors (Figure 3.12b): GPSR experiences lower TCP throughput 40% of the time compared to ARGF, caused again by the congestion window size (see Figure 3.12f) when GPSR faces a local minimum. Under such severe failures, both GPSR and ARGF fail to deliver packets $\approx 45\%$ and 35% of the time.

For low node mobility (5 m/s), both ARGF and GPSR deliver all packets with a TCP throughput of $\approx 1 - 2$ Mbps (Figure 3.12c). However, when GPSR enters the recovering mode near patient location 3 (after ≈ 500 sec), due to its planarization (which stretches paths), it shows lower TCP throughput (≤ 1 Mbps) than ARGF. That is in line with a lower congestion window size (see Figure 3.12g). At the same time, 60% of the time (first 500 sec) it shows similar performance. Under high node mobility conditions (20 m/s), we again observe similar behaviors (Figure 3.12d): GPSR experiences lower TCP throughput 40% of the time compared to ARGF, caused again by the planarization when GPSR faces a local minimum. That is confirmed by congestion window size (see Figure 3.12f) . Under such high mobility, both GPSR and ARGF are still able to deliver all packets, which makes geographic routing more attractive to disaster-incident response activities which benefit from the real-time situational awareness.

Even though both AODV and HWMP have advantages over pure proactive stateful routing solutions, in a challenged disaster scenario they do not show acceptable throughput level, leading to service outages caused by disconnections (from 20% to 90% percent of the time). Recent solutions in stateful greedy forwarding literature can help cope with some disaster incident challenges [40, 41]. For example, recent stateful greedy forwarding solutions have shown promising results under severe node failures [40]. However, we found no stateful greedy forwarding algorithm which can cope with both severe node failures

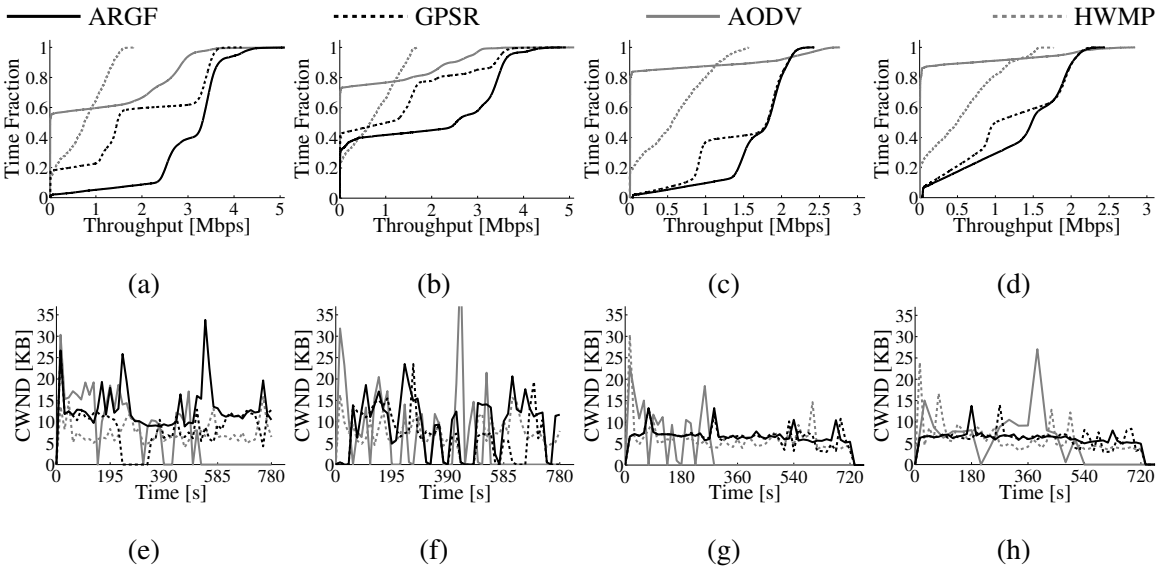


Figure 3.12: Time fraction of the TCP throughput (top row) and congested window (CWND) size (bottom row) averaged over 1 and 15 seconds, respectively, under (a, e) low node failures (5%) and (b, f) severe node failures (50%) and under (c,g) low node mobility (5 m/s) and (d,h) high node mobility (20 m/s): half of the time GPSR faces a local minimum showing two times worth throughput and lower CWND than ARGF due to higher path stretch caused by GPSR planarization. Both AODV and HWMP stateful routing solutions show worse throughput level within a disaster scene, due to its challenging conditions. As expected, performance of all algorithms degrades as we increase node failures or high node mobility.

and high mobility. The superior performance of ARGF is due to its knowledge about a static physical obstacle located within the disaster scene, which in most cases allows local minima avoidance by using our proposed *Repulsion* forwarding.

Chapter 4

On Constrained Shortest Path for Virtual Network Service Management

The constrained shortest path problem is the NP-hard problem of finding the shortest path subject to an arbitrary set of hop-to-hop and end-to-end constraints. As we will see in the next chapter, it plays fundamental role in the (primary) service function chaining problem.

4.1 The Constrained Shortest Path Problem and Virtual Network Service Management

In this section, we first define the constrained shortest path problem using optimization theory. We then motivate the importance of its *flexible* and *scalable* management in diverse virtual network services.

4.1.1 Constrained Shortest Path Problem

Let l be the number of hop-to-hop or *link constraints* for min/max network metrics, e.g., bandwidth, and p be the number of end-to-end *path constraints* for additive/multiplicative network metrics, e.g., delay or loss. Moreover, we denote with $l \oplus 1$ paths with multiple

links and a single path constraints, and with and $l \oplus p$ paths with multiple links and multiple path constraints. Given the above notation, we define the constrained shortest path problem as follows:

Problem 1 (constrained shortest path)**.** *Given a physical network graph $G = (V, E)$, where V is the set of vertices and E the set of edges; let us denote with D the flow demand to be transferred and let u_{ij} denote a capacity of the directed edge e_{ij}; let f_{ij} be a binary variable $f_{ij} \in \{0, 1\}$ denoting a ratio of flow on the edge e_{ij}, and let c_{ij} denote a cost of transferring a unit of flow through such edge; finally, let $\bar{l}$ and $\bar{p}$ denote vectors of link (hop-to-hop) and path (end-to-end) constraints, excluding capacity constraints, where $\bar{l}$ corresponds to min/max edge e_{ij} weights $w_{ij}^{\bar{l}}$ (i.e., $\geq$ or $\leq$, respectively), and $\bar{p}$ corresponds to additive edge e_{ij} weights*[1] *$w_{ij}^{\bar{p}}$; the problem of finding constrained shortest path between source v_s and destination v_t vertices can be formulated as follows:*

$$\textit{minimize} \sum_{e_{ij} \in E} c_{ij} D f_{ij} \tag{4.1}$$

subject to

Flow Conservation Constraints:

$$\sum_{v_j \in V} f_{ij} - \sum_{v_k \in V} f_{ki} = \begin{cases} 1, & i = s \\ 0, & i \neq s \textit{ or } t \\ -1, & i = t \end{cases}, \forall v_i \in V \tag{4.2}$$

Capacity Constraints:

$$D f_{ij} \leq u_{ij}, \forall e_{ij} \in E \tag{4.3}$$

Other Link Constraints:

$$w_{ij}^{l} f_{ij} \leq l, \forall e_{ij} \in E, l \in \bar{l} \tag{4.4}$$

[1]Note that multiplicative constraints (e.g., packet loss) can be converted to additive by composing them with a logarithmic function to avoid nonlinearity.

Path Constraints:

$$\sum_{e_{ij} \in E} w_{ij}^{p} f_{ij} \leq p, \forall p \in \bar{p} \tag{4.5}$$

Existential Constraints:

$$f_{ij} \in \{0, 1\}, \forall e_{ij} \in E \tag{4.6}$$

Finding a *shortest path* (without constraints) has a polynomial time complexity: consider Equations 4.3, 4.4 and 4.5: in absence of any link or path constraints, the constraint matrix of the above optimization problem is unimodular [114]. This condition allows us to solve the optimization problem using (polynomial) linear programming. Such time complexity bound does not necessarily hold in presence of at least a single link or path constraint ($\bar{l} \neq 0$ or $\bar{p} \neq 0$). In that case, we have to solve the above optimization problem using (NP-hard) integer programming [114]. Note that we can always avoid specifying capacity constraints in Equation 4.3 by simply setting all $f_{ij} = 0$, which corresponding physical edge e_{ij} capacity u_{ij} is not sufficient to allocate flow demand D. The same applies for other link constraints (see Equation 4.3).

4.1.2 Constrained Shortest Path for Virtual Network Service Management

Now let us show how finding a flexible and scalable solution to Problem 1 benefits a wide range of path finder subproblems to manage virtual network services such as Traffic Engineering (TE), Virtual Network Embedding (VNE) and NFV Service Chaining (NFV-SC).

Finding Virtual Paths in Resource Constrained Scenarios. We begin by considering a variant of the constrained shortest path that operates on a resource constrained scenario, e.g., a natural or man made disaster scenario where connectivity is scarce. In those cases, one aim is to minimize the overall physical resource consumption of virtual paths. To this end, we define the *resource optimal constrained path* by modifying the objective of Problem 1 as follows:

Problem 2 (resource optimal constrained path)**.** *The resource optimal constrained path is a path that satisfies an arbitrary set of link/path constraints using minimal amount of physical bandwidth:*

$$\textit{minimize} \sum_{e_{ij} \in E} D f_{ij} \tag{4.7}$$

where f_{ij}, e_{ij}, D and E are as defined in Problem 1.

Note that by defining an equal weight c to all edges we seek the minimum hop path that satisfies an arbitrary set of link/path constraints (e.g., imposed by SLO).

4.1.3 Traffic Engineering

TE techniques today can be roughly divided into two groups: oblivious i.e., no a-priori knowledge of the SLO demands [115], and demands-aware, when such knowledge is available [6, 7]. Moreover, the later has a superior performance (e.g., can better utilize substrate network resources) than the former [115] at the expense of having a centralized forwarding (or routing) control [6, 7, 116].

We broadly classify demands-aware traffic engineering solutions (see e.g. [6, 7]) with the following network utility maximization problem:

$$\textit{maximize}[\min_{f_i \in F} fairness_i(f_i)] \tag{4.8}$$

where F denotes a set of all demands (or commodities); in [7], for example, such commodities are $\{src, dst, SLO\}$ tuples; $f_i \in [0, 1]$ is continuous variable that denotes the ratio of flow for commodity i with bandwidth demand D_i; and $fairness_i(f)$ is a linear piecewise-defined function whose definition is based on path service's demands SLO constraints. For a complete problem formulation we refer to [117, 118].

Constrained shortest path relevance. There are two standard ways of formulating the TE optimization problem shown in Equation 4.8 — the arc-based [118] and path-based [117]

formulations. In practice, due to hardware granularity limitations, the arc-based linear programming solution can be infeasible to implement, or require use of NP-hard integer programming which can be intractable even for moderate size networks. The shortest path algorithms such as Dijkstra (e.g., within k-shortest path algorithms [119]) are currently used to find a set of paths as an input for the path-based linear programming formulations or for their simplified greedy solutions [6, 7]. The hope is also to map the highest priority flows to the minimum latency paths first. However, if we are aware of the services' SLO demands (e.g., bandwidth, latency, loss rate, etc.), we can use them as constraints to optimize physical network utilization (and hence the flow fairness) as described in Problem 2.

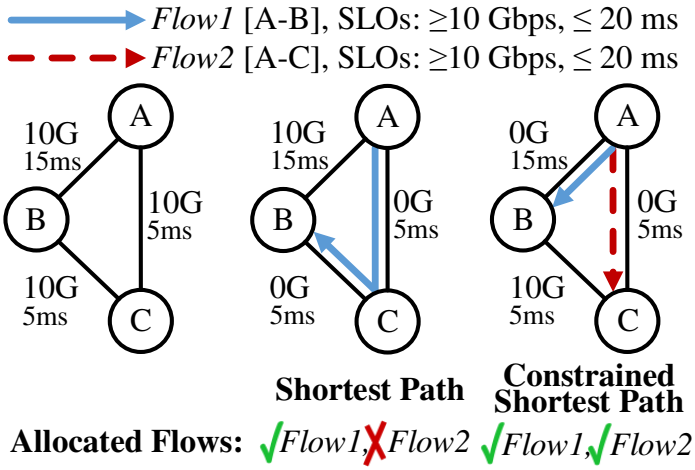


Figure 4.1: Applying constrained shortest paths to allocate traffic flows improves network utilization: the widely applied shortest paths, e.g., to allocate traffic flows, in this case $Flow1$, can hinder allocation of other flows; allocating $Flow1$ on a constrained shortest path as in Problem 2 instead permits the allocation of $Flow2$ as well.

Motivating example. Consider Figure 4.1: two flows $Flow1$ and $Flow2$ with bandwidth and latency constraints are to be allocated on a physical network of Figure 4.1-left. Attempting an allocation of $Flow1$ by merely considering the shortest path algorithm to minimize the latency (Figure 4.1-center) — as current TE solutions do, — the hosting physical path would have to use up to two physical links ($A \rightarrow C \rightarrow B$), thus preventing the allocation of the subsequent requested flow $Flow2$. If instead a constrained shortest path is used to allocate $Flow1$ and $Flow2$ as shown in Figure 4.1-right, then path $A \rightarrow B$ could be found, leaving capacity for the allocation of $Flow2$ resulting in its higher fairness.

On the contrary, when shortest path algorithms are used to find min-hop paths, such algorithms can lead to an infeasible solution even when a feasible one exists, leading to TE performance degradation. For example, consider the mapping of $Flow1$ with a latency constraint of 10 ms (tighter than 20 in Figure 4.1). The min hop (shortest) path $A \rightarrow B$ violates the 10 ms latency constraint, preventing the allocation of $Flow1$, whereas the constrained shortest path algorithms would find the feasible and optimal $A \rightarrow C \rightarrow B$ path even though more hops are needed to satisfy such tighter SLO demand. Note, however, that the above problem solution of allocating sequential flow requests can be suboptimal w.r.t. the problem solution of allocating such flows when they are known in advance. We dissect such network utilization gains when constrained shortest paths are applied for TE in Section 4.4.

Embedding Virtual Networks and Service Chains. Embedding a virtual network (service chain) requires a constrained virtual service request to be mapped on top of a physical network hosted by a single infrastructure provider, or by a federation of providers. To solve this (NP-hard [120]) graph matching problem, earlier work proposed centralized (see e.g. [60, 61, 62, 121, 122, 123, 124]) and distributed [74, 125, 126] algorithms. Such solutions either separate the node embedding from the link embedding phase [74, 123, 125, 126], or simultaneously apply the two phases [60, 61, 62, 121]. When link embedding is considered separately, paths can be found dynamically or precomputed for each virtual link using Dijkstra or k-shortest path algorithms [74, 123, 125]. When instead node and link embedding are considered jointly, recent work has shown how embedding problems can be formulated as known multi-commodity flow problems [60, 61, 62]. On the one hand, all these problems show how there is a need to minimize providers' management costs when allocating the virtual network or the service chain [62] as shown in the following objective:

$$\textit{minimize} \sum_{e_{st} \in E_V} \sum_{e_{ij} \in E_S} c_{ij} D_{st} f_{ij}^{st} + \sum_{v_s \in V_V} \sum_{v_i \in V_S} c_i D_s x_i^s \tag{4.9}$$

where V_S and E_S (V_V and E_V) denote sets of physical (virtual) vertices and edges, respectively; c_{ij} and c_i denotes unitary bandwidth and CPU cost on physical edge e_{ij}, respectively; $f_{ij}^{st} \in [0, 1]$ is continuous variable that models the ratio of flow from v_s to v_t virtual vertices transferred through the physical edge e_{ij} when a flow with demand D_{st} is splittable, or, in its binary form, $f_{ij}^{st} \in \{0, 1\}$ for unsplittable flows. Finally, $x_i^s \in \{0, 1\}$ is the binary variable that denotes whether the virtual vertex v_s with computation demand D_s is assigned to the physical vertex v_i or not. For a complete problem formulation we refer readers to [60, 61, 62]. On the other hand, when virtual network (or service chain) requests are unknown in advance, the cost minimization strategy presented in Equation 4.9 can cause physical network partitioning, that in turn can lead to lower physical network utilization [60, 61]. To maximize the long term provider's revenue i.e. to allocate more virtual network requests, often a load balancing objective is sought [60, 61]:

$$\textit{minimize} \sum_{e_{st} \in E_V} \sum_{e_{ij} \in E_S} \frac{c_{ij} D_{st}}{u_{ij}} f_{ij}^{st} + \sum_{v_s \in V_V} \sum_{v_i \in V_S} \frac{c_i D_s}{u_i} x_i^s \tag{4.10}$$

where u_{ij} and u_i denote available capacities of physical edge e_{ij} and vertex v_i, respectively.

Constrained shortest path relevance. To cope with integer programming intractabilities for solving path-based multi-commodity flow problems, the well-known *column generation approach* can be applied as in [61, 62]. The column generation approach iteratively adds only those paths (i.e., flow variables or columns) to the problem formulation that improve objective. In [61, 62] the Dijkstra shortest path algorithm is used to find the best shadow price paths (columns) for each virtual link to include them in the formulation. Although such approach best improve the objective value of e.g., Equation 4.10, it can be suboptimal when virtual links have additional constraints such as latency or loss rate. In this scenario, a path between any two physical nodes (found by solving Problem 1) can be a part of the optimal solution, even if it has a worse objective value than the shortest path, but satisfies all virtual link constraints. Generally, for any two stage or one-shot VNE

algorithm, constrained shortest paths can be used to best improve the objective value while satisfying an arbitrary number of virtual link constraints. In Section 4.4, we confirm such intuition empirically, by studying the allocation ratio improvements (a proxy for provider's revenue) when our method is used to find paths for embedding services.

4.2 Neighborhoods Method

To solve the NP-hard constrained shortest path problem with an arbitrary set of (e.g., SLO) constraints in practical time, we propose the novel Neighborhoods Method (NM).

Why the name "Neighborhoods Method"? Most path seeking algorithms require at least two inputs for each node: (i) knowledge of neighbors, and (ii) awareness of all adjacent link costs, often dictated by policies, or SLO constraints. Such constraints are then used by the path seeking algorithm to compute the lowest-cost paths. The Dijkstra algorithm e.g., recursively finds the shortest path traversing the source neighbors and the neighbors of their neighbors. This recursive notion leads to our definition of "neighborhoods" in NM, $i.e$, a set of nodes that can be reached from the source node with the same number of hops, where each "neighborhood" (being a set) contains unique elements. Based on Bellman's "Principle of Optimality" [127], such node repetitive structures can be used for label-correcting dynamic programming that we apply to reduce a number of exhaustive search path candidates.

4.2.1 The NM General Case ($l \oplus p$ case)

As the exact constrained shortest path algorithm, the worst case time complexity of NM when accepting an arbitrary set of hop-to-hop and end-to-end constraints ($l \oplus p$) is exponential. Our NM complexity analysis shows that the time complexity exponent is, however, halved with respect to common branch-and-bound path finders methods based on exhaus-

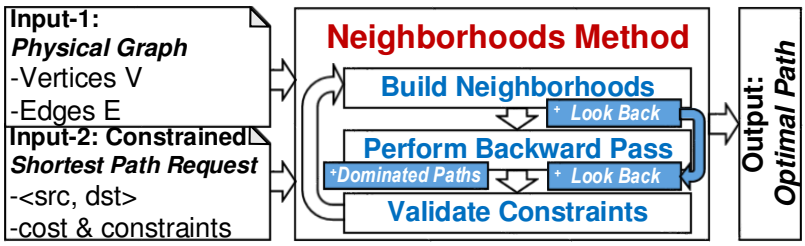


Figure 4.2: Neighborhood Method Workflow in $l \oplus p$ case: The k^{th}-hop Neighborhood Build (forward pass) first (using label-correcting), and the Backward Pass later find all simple paths of k length (with any instance of branch-and-bound exhaustive search); in the third step, we check candidates feasibility; we then repeat recursively the backward pass with the $k+1$ neighborhoods, until the *optimal* (constrained shortest) path is found or all candidates have been eliminated. To improve the time to solution, we couple NM with a dominated path and a look-back search space reduction technique.

tive search [53, 54, 55] (see Section 4.3). Note also that in addition to the constrained shortest path, NM can simplify the process of finding all simple, k-constrained shortest or Pareto-optimal paths [54] from the source to the destination.

The general workflow of our NM algorithm is shown in Figure 4.2, and Algorithm 3 outlines its work: NM is executed in three phases: (i) a *forward pass* or neighborhoods building (by using label-correcting), (ii) a *backward pass* (with any instance of branch-and-bound exhaustive search), and a final (iii) *constraints validation* phase. During the forward pass, NM builds the neighborhoods to estimate the path length. The backward pass is used to find end-to-end paths with a given length (hop count). The final constraints validation phase is used to keep the best path candidate and decide whether or not the path search should be extended to longer path candidates involving more neighbors.

Algorithm 3: General NM algorithm ($l \oplus p$ case)

Input: X:= src Y:= dest, $l \oplus p$:= constraints list
Output: The *optimal path* between X and Y (which satisfies $l \oplus p$).

```
1 begin
    /* Build neighborhoods < NH > from X to Y                                  */
2   < NH >⟵ Build Neighborhoods /* Backward Pass to find whole set of simple paths
        <path> of <NH>.size length                                              */
3   < path >⟵ Perform Backward Pass /* Validate SLO constraints; if not satisfied,
        add one more neighborhood and repeat Backward Pass                      */
4   Validate Constraints
5 end
```

Example 1. *Consider a network consisting of 4 nodes X, Y, A and B, as shown in Figure 4.3a. On the link (X, A), we denote with the first value 5 the link constraint (in*

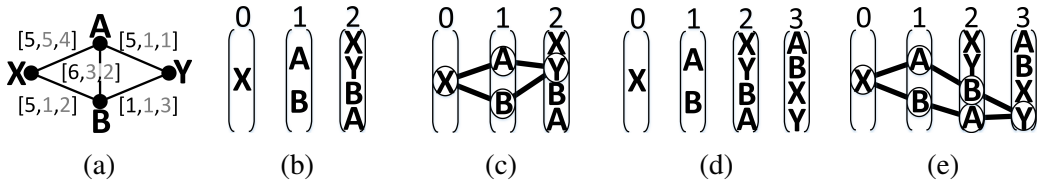
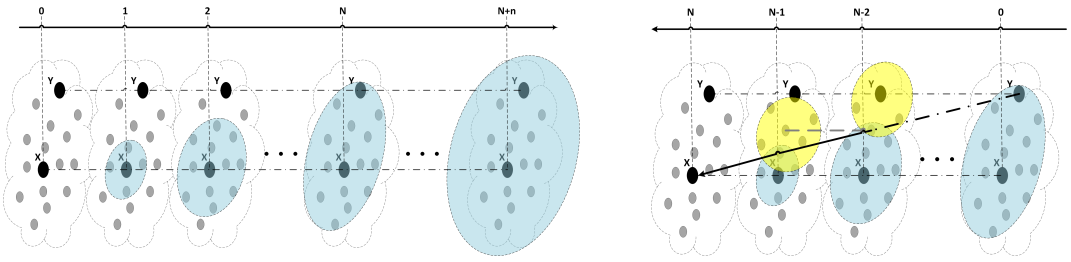


Figure 4.3: Running example of NM in $l \oplus p$ case: (a) An example network configuration with [bandwidth, delay, cost] link metrics; (b) NM forward pass phase estimates the min hop count distance to Y during the first iteration, and (c) backward pass identifies all the shortest path candidates, which in this case do not contain the *optimal* path; (d) the forward pass adds more neighborhoods to find a path with longer distance to Y. (e) The Backward pass identifies all paths of a given new length; in this case the path contains the *optimal* $X \rightarrow B \rightarrow A \rightarrow Y$ solution.

this case bandwidth), and with the second 5 *we refer to the first path constraint (in this case end-to-end delay); with the third value* 4 *we refer to the second path constraint (in this case an arbitrary cost). In its general case, NM finds a path from* X *to* Y *satisfying the three constraints* $bw \geq 5$*,* $delay \leq 5$ *and* $cost \leq 5$ *as follows: in the first phase, NM builds all neighborhoods starting from the source node* X *until the destination node* Y *is reached (Figure 4.3b). Upon reaching* Y*, NM begins the backward pass to find the full set of min hop paths (Figure 4.3c). If the set contains the optimal path (in this case for Problem 2), NM terminates with a solution. If no such path is found, NM builds an additional neighborhood as shown in Figure 4.3d and performs another backward pass to check for optimality among paths that are one hop longer than at the previous iteration (Figure 4.3e). NM iterates until either the solution is found, or the maximum path length is violated. In this example, NM returns the constrained shortest path* $X \rightarrow B \rightarrow A \rightarrow Y$*.*

Forward pass for $l \oplus p$**.** Given an arbitrary graph $G(V, E)$, where $|V|$ and $|E|$ represent the number of vertices and edges, respectively, in this phase NM successively builds neighborhoods $< NH >$ from the source X to the destination Y. Algorithm 4 describes the forward pass of NM, and Figure 4.4a illustrates its process. To build a new neighborhood NH, we add therein neighbors (adjacent vertices) of each vertex u in the current neighborhood cNH (line 6). For example, the first NH includes nearest neighbors of the source X (which is in the zero NH), and the second NH contains nearest neighbors of vertices in the first NH, and so on. The first phase ends as soon as the destination Y appears in cNH



(a) (b)

Figure 4.4: (a) Illustration of the neighborhoods building (i.e., the forward pass): each neighborhood involves more vertices than previous one. (b) Illustration of the backward pass coupled with Exhaustive Breadth-First Search (EBFS): we significantly reduce a number of path candidates by considering only intersections of previously built neighborhoods with processed vertex neighbors.

(line 4), or $< NH >$ size is more or equal to $|V|$ (line 13).

Algorithm 4: Build Neighborhoods ($l \oplus p$ case)

```
   Input: X:= src, Y:= dest
   Output: The list of neighborhoods < NH > from X to Y
1  begin
2      cNH ⟵ X
3      < NH > ⟵ < NH > ∪ cNH
4      while Y ∉ cNH do
5          NH ⟵ ∅
6          foreach Vertex u ∈ cNH do
7              NH ⟵ NH ∪ adjacent(n)
8          end
9          if < NH > .size < |V| then
10             < NH > ⟵ < NH > ∪ NH
11             cNH ← NH
12         else
13             Return Y is unreachable.
14         end
15     end
16 end
```

Backward pass for $l \oplus p$. The best exhaustive search strategy may depend on the network topology, constraint and cost functions, and we leave it as a policy for NM. In this thesis, we use EBFS for the backward pass of NM detailed in Algorithm 5 and illustrated in Figure 4.4b. Note however, that this phase can use any exhaustive search (e.g., EBFS [54], EDFS [53] or exhaustive k-shortest path [55]) with the only difference being that *we do not process all neighbors (adjacent vertices) of each vertex u but only those which are within previous NH* (line 10). The first step is to find the intersection between neighbors of the destination Y with its previous NH (line 5). This intersection is not an empty set, it contains at least one vertex v. For all obtained vertices we again build the intersection of

their neighbors with the penultimate NH (line 16). The second phase ends as soon as we hit the zero NH (line 6), and as a result we obtain the collection of all paths with a length of $< NH >$ size between the source X and the destination Y.

Algorithm 5: Perform Backward Pass ($l \oplus p$ case)

```
   Input: The list of neighborhoods < NH > from X to Y
   Output: All paths < path > from X to Y of < NH > .size length
1  begin
2      path ⟵ Y
3      < path > ⟵ < path > ∪path
4      k ⟵ 1
5      NH ⟵ < NH > [size − k]
       /* EBFS:                                                          */
6      while NH ≠ < NH > [0] do
7          < tempPath > ⟵ ∅
8          foreach path ∈ < path > do
9              Vertex u ⟵ path[1]
10             foreach Neighbor v ∈ adjacent(u) ∩ NH do
11                 < tempPath > ⟵ v ∪ path
12             end
13         end
14         < path > ⟵ < tempPath >
15         k ⟵ k + 1
16         NH ⟵ NH[size − k]
17     end
18 end
```

Constraints validation for $l \oplus p$. In this last phase, we check for candidates optimality, i.e., we check whether or not a candidate path satisfies all $l \oplus p$ constraints, and keep the best candidate. At each consequent iteration, we first build an additional $(N+1)$ neighborhood, repeat the backward pass and subsequently obtain all paths of length $N+1$, and then we check their feasibility and update the best known path candidate, if needed. Similarly to IBF [58], we keep iterating while the candidate path length is less than the number of vertices $|V|$ and then return the optimal solution.

We close this subsection with three important remarks: **(1) NM can be used to find k-constrained shortest paths**: the backward path at each iteration returns all possible path candidates of the same hop count. To find k-constrained shortest paths we need to keep not a single best (shortest) path candidate, but a set of k best path candidates at each iteration and update this set if needed. Clearly, as in the worst case NM traverses all possible path candidates, its upper bound complexity does not change (see Section 4.3). **(2) NM can be applied to both directed and undirected graphs making it an interesting solution even**

for NFV chain instantiations. When traversing directed graphs, NM simply uses vertices' outgoing neighbors during the forward pass and incoming neighbors during the backward pass. **(3) A distinctive feature of our NM is the construction of the intersection of two neighborhoods**. This leads to a quadratic reduction of path candidates for exhaustive search algorithms (see Section 4.3).

4.2.2 NM Search Space Optimizations

In this subsection we show how NM can be coupled with existing search space reduction techniques, i.e., dominant paths or look ahead [53, 54, 55], to speed up its time to solution. By leveraging the NM's double pass, we also propose a variant of the look ahead technique, $viz.$, "Look Back" without complexity overhead. We first describe the *dominated paths* method. Observe that the dominant paths technique is not applicable in case we wish to use NM as k-constrained shortest paths, as it removes suboptimal candidates which can be among k paths.

Dominated paths (pruning by dominance or bound). The basic idea behind dominated paths states that an algorithm can avoid evaluating candidate paths when their (multidimensional) distance is longer than other candidates distance, since they cannot be a part of the shortest solution [54, 55]. Consider for example Figure $4.3a$: note how path $X \rightarrow A \rightarrow Y$ with distance vector $\bar{d} = [6, 5]^T$ is dominated by $X \rightarrow B \rightarrow Y$ path with distance vector $\bar{d} = [2, 5]^T$ and hence it can be excluded to avoid unnecessary time-consuming processing. Path dominance is formally defined as follows:

Definition 1 (dominant path)**.** *We say that path P_1 dominates path P_2 if and only if:*

$$\exists\, i \in 1, \ldots, p, c : d_i(P_1) < d_i(P_2) \wedge d_j(P_1) \leq d_j(P_2),$$
$$\forall j \neq i \in 1, \ldots, p, c$$

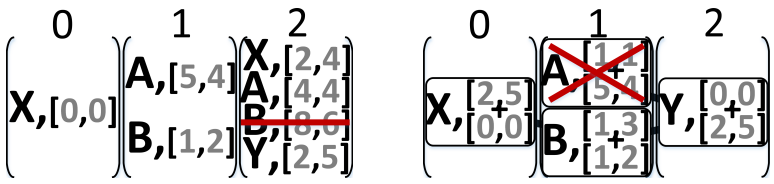


Figure 4.5: NM running in its in $l \oplus p$ general case reduce its search space with a *Look Back*. (a) Forward pass: excludes vertices violating at least a path constraint; (b) Backward pass: NM looks back and removes a path candidate which sum of its current $delay$ and the best (estimated during the forward pass) residual $delay$ violates the $delay$ constraint (i.e., $1 + 5 > 5$).

where d_i is a distance corresponding to p_i path constraint or to c path cost.[2]

In its general form, we can reduce NM's search space by removing the dominated paths. This is accomplished by keeping track of only non-dominated paths during its backward pass (Algorithm 5, line 11). In particular, a path is added to a vertex v if it is not dominated by any other path to v, or if it is not dominated by other paths from the source to the destination.

Look ahead (pruning by infeasibility). We can further reduce NM's search space by omitting path candidates with endpoint v, if the sum of their current path distances and the residual best distances from v to the destination violates any of the path constraints. This technique is known as "look ahead" [55] and requires a run of Dijkstra or Bellman-Ford algorithm p times to pre-compute best distances (corresponding to path constraints p) from all vertices to the destination.

Looking backwards (pruning by infeasibility). To avoid running e.g., Dijkstra algorithm $p + 1$ times, in this thesis we propose a more efficient look back technique that reduces the search space. The best distance could be estimated now from the intermediate vertex v to the source (instead of the destination) for each neighborhood containing v during the NM forward pass. We then use such information during the backward pass to exclude paths from v at neighborhood N that violate corresponding path constraints.

Note that when our looking backward technique is used, we can learn the best distance

[2]Note how, if only the cost distance $d_c(P)$ is used in Equation 1, pruning by path dominance is the well-known pruning by bound technique [53].

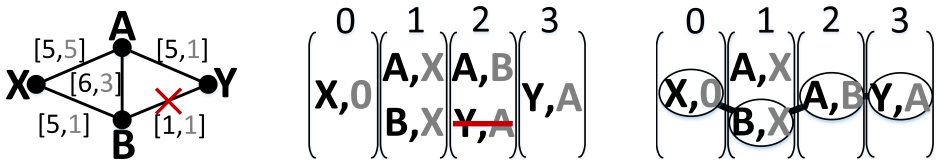
for each node in each neighborhood. Moreover, as physical nodes may appear in more than one neighborhood, we have more chances to prune path candidates than with the look ahead search space reduction technique. In addition, as we coupled the look back space reduction with the NM forward pass, we do not introduce additional overhead with respect to the look ahead reduction (i.e., running Dijkstra $p + 1$ times).

Example 2. *In this example we revisit Example 1, to describe how NM in its the general case coupled with a Look Back search may reduce the search space when finding a path from X to Y. We assume that two path constraints $delay \leq 5$ and $cost \leq 5$ need to be satisfied. During the forward pass (Figure $4.5a$) NM estimates the path length to Y, keeping track of all constraint-satisfying distances for each vertex and at each neighborhood. Vertices whose estimated distance violate at least one path constraint are instead removed. During the backward pass, NM removes $A \rightarrow Y$ path candidate whose sum of the current $delay$ (which equals to 1) and the best estimated $delay$ from the source to A (which equals to 5) violates $delay$ constraint $1 + 5 > 5$ (Figure $4.5b$).*

4.2.3 NM for l links and a single path constraint ($l/l \oplus 1$ cases)

The worst case running time of NM becomes polynomial (by avoiding an exhaustive search) if either a general constrained shortest path (see Problem 1) with only l link constraints or the resource optimal constrained path (see Problem 2) with l links and a single path constraint ($l \oplus 1$) are sought (see Section 4.3). This is because unnecessary iterations and phases (including the exhaustive search) are avoided. Note however that the constrained shortest path in $l \oplus 1$ case is still NP-hard [128], and hence the general NM version should be used.

Example 3. *Revisiting our example 1, NM finds a path from X to Y satisfying the two constraints $bw \geq 5$ and $delay \leq 5$ as follows: In the pre-routing phase (see Figure $4.6a$) NM prunes $B \rightarrow Y$ due a bw violation. In the forward phase (see Figure $4.6b$) NM finds the*



(a) (b) (c)

Figure 4.6: Running example of NM in $l \oplus 1$ case: (a) In the *pre-routing phase* NM prunes $B \rightarrow Y$ due a bw violation; (b) In the *forward phase* NM finds the best length to Y saving vertices' predecessors from previous neighborhoods; (c) *back track phase* identifies the *resource optimal constrained path* $X \rightarrow B \rightarrow A \rightarrow Y$ by recursive predecessor visits.

best length to Y saving information about vertices' predecessors from previous neighborhoods, discarding candidates that violate the path constraint. During the back track phase NM recursively constructs the resource optimal constrained path by recursive predecessor visits (see Figure $4.6c$*).*

NM pseudocode for $l \oplus 1$. Algorithm 6 describes the NM procedure with only $l \oplus 1$ constraints. The neighborhoods data structure $< NH >$ identifies each vertex with its predecessors in previous neighborhoods; each vertex u also contains a path distance $D(u)$. During the pre-routing phase, $D(u)$ (line 3), the weights of each edge not satisfying all l constraints (line 7) are set to ∞.

During the forward phase of NM, we successively build the neighborhoods $< NH >$; for each *current* neighborhood cNH of u we exclude all neighbors that do not satisfy the path constraint or that have previous distance lower than $dist$ (line 18). Note that to compute $dist$ (line 17) we assume $D(u)$ to be a distance of u after its inclusion in current neighborhood (cNH). The forward phase ends when the destination Y appears in cNH (line 13). If the number of neighborhoods $< NH >$ is equal to the number of vertices $|V|$, the algorithm terminates concluding that a negative weight cycle is detected (line 28), or that a path does not exist (line 30). During the back-track phase, NM then recursively finds the *resource optimal constrained path* between X and Y.

Algorithm 6: NM in $l \oplus 1$ case

Input: X:= src, Y:= dest, l link constraints, p single path constraint.
Output: The *optimal path* between X and Y (which satisfies $l \oplus 1$).

```
1  begin
       /* pre-routing phase:                                                    */
2      foreach Vertex u ∈ |V| do
3          D(u) ← ∞
4      end
5      foreach Edge u → v ∈ |E| do
6          if u → v does not satisfy l then
7              w(u → v) ← ∞
8          end
9      end
       /* forwarding phase:                                                     */
10     D(X) ← 0
11     cNH ⟵ (X, NIL)
12     < NH > ⟵ < NH > ∪cNH
13     while Y ∉ cNH do
14         NH ⟵ ∅
15         foreach Vertex u ∈ cNH do
16             foreach Neighbor v ∈ adjacent(u) do
17                 dist ← D(u) + w(u → v)
18                 if dist < D(v) and dist ≤ p then
19                     D(v) ← dist
20                     NH ⟵ NH ∪ (v, u)
21                 end
22             end
23         end
24         if NH ∉ ∅ and | < NH > | < |V| then
25             < NH > ← < NH > ∪NH
26             cNH ← NH
27         else if | < NH > | == |V| then
28             Return Negative Weight Cycle is detected.
29         else
30             Return Y is unreachable.
31         end
32     end
       /* back track phase:                                                     */
33     path ← Y
34     k ← | < NH > |
35     while k > 0 do
36         predecessor ← < NH > [k, path(1)]
37         path ← predecessor ∪ path
38         k ← k − 1
39     end
40 end
```

4.2.4 NM with only link constraints (l case)

In presence of only l link constraints and when the resource optimal constrained path is sought, NM can be further simplified to run in linear time by omitting the path constraint checking and by using hop count as a vertex distance metric (Algorithm 6, line 18). As a result, NM traverses each vertex exactly once, and hence all neighborhoods contain unique and distinct vertices, i.e., a common vertex for two or more neighborhoods cannot exist.

4.2.5 NM Optimal Solution

We conclude this section stating an important result, whose proof (by contradiction) is included in the Appendix A.1.

Theorem 1. (Theorem of the Optimal Solution)

The Neighborhood Method always finds the optimal path if it exists.

Proof. See Appendix A.1. ■

4.3 Asymptotic Complexity Analysis

In this section, we provide a complexity analysis comparison among our NM and related algorithms such as: IBF [58], EDijkstra [105], EDFS [53] and EBFS [54, 106] — common branch-and-bound exhaustive search approaches. Appendix A.2 summarizes the main results of this comparison, where $|V|$ is the total number of vertices, and $|E|$ is the total number of edges.

Theoretical results summary. The major benefits of NM arise when we are seeking the NP-hard [53, 54] constrained shortest paths in $l \oplus 1$ and $l \oplus p$ cases. We show how, under l constraints, EDijkstra finds constrained (shortest) paths faster than IBF and NM. We also show how NM is an alternative for IBF to find the resource optimal constrained path when accepting $l \oplus 1$ constraints (and thus *flexible*), and how EDijkstra loses any path optimality guarantees in that case. Finally, when accepting $l \oplus 1$ and $l \oplus p$ constraints, we show how time and space EBFS and EDFS complexities are quadratically higher with respect to our NM.

4.3.1 Complexity Analysis for the l and $l \oplus 1$ Cases

EDijkstra complexity (l and $l \oplus 1$ cases). Given the presence of merely link constraints in the l case, EDijkstra can find the constrained shortest path by minimizing an arbitrary

cost function and its variant the resource optimal constrained path by minimizing path hop count. The original Dijkstra algorithm can run in $O(|V|log|V| + |E|)$ time utilizing Fibonacci Heap [129]. Its min hop count variant can be reduced to $O(|V| + |E|)$ with Thorup's algorithm [130]. The corresponding space complexity for both Dijkstra variants is $O(|V|+|E|)$. Before applying Dijkstra, we also need to verify the l constraints satisfaction by pruning all edges which violate l in O($|E| \cdot l$). Thus, the total time complexity of finding the constrained shortest path by EDijkstra is $O(|V|log|V|+|E|\cdot l)$ with a space complexity of $O(|V| + |E|)$. For the resource optimal constrained path, the total time complexity of EDijkstra is reduced to $O(|V| + |E| \cdot l)$ with the same space complexity.

Given the presence of link and a singe path constraints in the $l \oplus 1$ case, EDijkstra has to minimize distance which is a metric for the required path constraint to guarantee feasibility of the solution, i.e., to find a feasible solution if it exists. Thus, EDijkstra loses its ability to any path length optimization. The total time complexity of finding a constrained path by EDijkstra in $l \oplus 1$ case is again $O(|V|log|V| + |E| \cdot l)$ with a space complexity of $O(|V| + |E|)$. Note that the constrained shortest path in $l \oplus 1$ case is NP-hard [128], and thus requires an exhaustive search for the solution. As a result, EDijkstra cannot find any variant of the constrained shortest path with $l \oplus 1$ constraints.

IBF complexity (l and $l\oplus1$ cases). The original Bellman-Ford shortest path algorithm [127] runs in $O(|V||E|)$ time and similarly to EDijkstra can be extended to meet l constraints in $O(|E| \cdot l)$ additional time. Thus, in l case we can find the constrained shortest path by minimizing an arbitrary cost function with Bellman-Ford algorithm with $O(|V||E| + |E| \cdot l)$ time and $O(|V||E|)$ space complexities. In contrast to EDijkstra, Bellman-Ford can also iteratively find shortest paths in ascending hop count order. Hence, the iterative version of Bellman-Ford (IBF) algorithm can minimize path hop count by visiting each vertex and "relax" its adjacent edges only once, starting from the source while advancing towards the destination. This steps takes $O(|V| + |E|)$ allowing IBF to find the resource optimal constrained path in l case with $O(|V| + |E| \cdot l)$ time and $O(|V| + |E|)$ space complexities.

The same iterative property of IBF can be used to find the resource optimal constrained path in $l \oplus 1$ case. To this aim, at each iteration when the next best hop count candidate is found, we check its path constraint feasibility. Once, the feasible path is found, IBF returns the resource optimal constrained path, i.e., the min hop count path which satisfy a single path constraint. Thus, IBF finds the resource optimal constrained path in $l \oplus 1$ case with the same $O(|V||E| + |E| \cdot l)$ time and $O(|V||E|)$ space complexities as for the constrained shortest path in l case. Due to NP-hardness of the constrained shortest path in $l \oplus 1$ case, IBF is also not applicable to find it.

EDFS and EBFS complexities (l and $l \oplus 1$ cases). In case of only l link constraints both EDFS and EBFS can find the resource optimal constrained path (by minimizing hop count) without an exhaustive search. To this end, we can run the original DFS and BFS algorithms which have linear time complexity $O(|V|+|E|)$. The only difference lays again in the adjacent link checking that ensures the l constraints satisfaction resulting in a time complexity of $O(|V| + |E| \cdot l)$, and a space complexity of $O(|V| + |E|)$.

However, to find the constrained shortest path with l constraints, or both the constrained shortest and the resource optimal constrained paths with $l \oplus 1$ constraints, EDFS and EBFS algorithms would have to build and check all possible paths from the source node (exhaustive search). For each potential path, algorithms can compute cost and path metrics and check for a path constraint satisfaction in $O(2)$. All constraint violating edges could be pruned prior to running EDFS or EBFS. Note how, as EDFS and EBFS algorithms use a branch-and-bound approach, they may visit each vertex more than once. The total number of path candidates can be bound as $O(b^d)$ [131], where b is the number of neighbors, and d is the maximum loop-free path hop count. The average number of neighbors per vertex b can be obtained from the hand-shaking lemma[3]:

$$b = \frac{\sum_{i=v}^{V} (neighbors\ of\ vertex\ v)}{|V|} = \frac{2|E|}{|V|}. \tag{4.11}$$

[3]Without loss of generality, we can assume undirected network graphs as in a directed graph, b equals to the average outdegree: $b = \frac{|E|}{|V|}$.

Using Equation 4.11 and based on the fact that the maximum loop-free path hop count equals to $|V-1|$, the EDFS and EBFS time complexities $O^{l\oplus 1}$ are:

$$O^{l\oplus 1} = O(2b^d + |E| \cdot l) = O\Big(\Big(\frac{|E|}{|V|}\Big)^{|V|} + |E| \cdot l\Big). \tag{4.12}$$

Based on Equation 4.12, the EBFS space complexity is $O\Big(\Big(\frac{|E|}{|V|}\Big)^{|V|}\Big)$.

NM complexity (l and $l \oplus 1$ cases). To estimate the time complexity of NM of finding the constrained shortest path with l constraints or the resource optimal constrained path with $l \oplus 1$ constraints, we first obtain the average number of neighbors per vertex b from the hand-shaking lemma shown in Equation 3.1. During the *pre-routing phase*, NM visits each vertex and edge to either set initial values or prune an edge due to a constraint violation in $O(|V| + |E| \cdot l)$. During the *back track phase*, which runs in linear time, NM recursively visits each predecessor starting from the destination in $O(|V|)$. During the *forward phase* we have quadratic complexity as to construct a neighborhood with best path distances, we loop over all b neighbors of each node of the previous neighborhood in $O(|V|b)$. The total number of neighborhoods is at most the maximum loop-free path hop count which equals to $|V-1|$. Here we assume that a vertex look up and placement takes $O(1)$ time. The overall NM complexity is hence:

$$O^{l\oplus 1} = O(|V|^2 b + 2|V| + |E|l) = O(|V||E| + |E| \cdot l). \tag{4.13}$$

Based on Equation 4.13, the NM space complexity is $O(|V||E|)$.

As for related algorithms, the process of finding the resource optimal constrained path with l constraints can be further simplified to linear complexity. To minimize path hop count, all neighborhoods need to contain only unique and distinct vertices, i.e., a common vertex for two or more neighborhoods cannot exist; this means that the maximum size of all neighborhoods is $|V|$. For each vertex in the neighborhoods set, the complexity of retrieving its neighbors is $O(b)$, assuming that a vertex look up and placement takes $O(1)$

time. Therefore, the time complexity of NM in this case is $O(|V|b + 2|V| + |E| \cdot l) = O(|V| + |E| \cdot l)$ and its space complexity is $O(|V| + |E|)$.

4.3.2 Complexity Analysis for the $l \oplus p$ Case

EDijkstra and IBF complexities ($l \oplus p$ case). For completeness, we mention that neither EDijkstra nor IBF are applicable to any variant of the constrained shortest path in the $l \oplus p$ case.

EDFS and EBFS complexities ($l \oplus p$ case). We remark that both EDFS and EBFS can find any variant of the constrained shortest path or k such paths in exponential time by exhaustive search. For each potential path, these algorithms check the satisfaction of all p constraints by calculating $p + 1$ new path and new cost metrics in $O(2p)$. All l constraints violating edges could be pruned prior to running EDFS or EBFS. Using Equation 3.1 and based on the fact that the maximum loop-free path hop count equals to $|V - 1|$, EDFS and EBFS time complexities $O^{l \oplus p}$ are:

$$O^{l \oplus p} = O(2p \cdot b^d + |E|l) = O\Big(\Big(\frac{|E|}{|V|}\Big)^{|V|} p + |E| \cdot l\Big) \tag{4.14}$$

The space complexity equals to $O\Big(\Big(\frac{|E|}{|V|}\Big)^{|V|} p\Big)$ due to the fact that we have to store p metrics for each path.

NM complexity ($l \oplus p$ case). Similar to EDFS and EBFS, NM can find any variant of the constrained shortest path or k such paths with $l \oplus p$ constraints in exponential time utilizing any instance of an exhaustive search. The neighborhoods contains non-unique nodes, and the total number of their nodes can be up to $|V|^2$. For each neighbor, NM checks if it already appears in the neighborhood $O(b)$. Taking into account the edge pruning phase, to ensure the constraints satisfaction and to reduce a search space, the time complexity of the

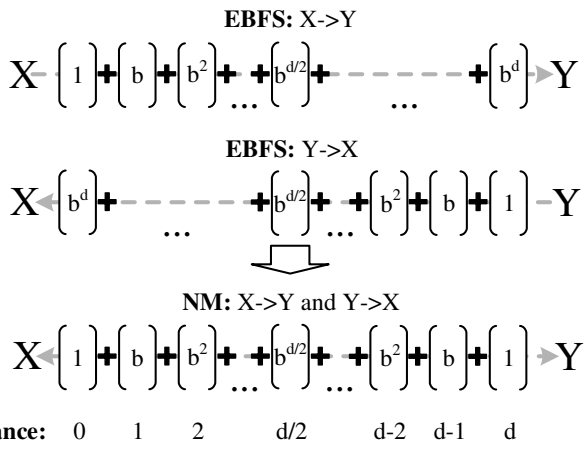


Figure 4.7: Differently than the branch-and-bound exhaustive search of EBFS, in which traversed paths grow between X and Y, NM's, reduces the search space significantly reducing the path candidates (from $O(b^d)$ to $O(b^{\frac{d}{2}})$ paths) due to its forward and backward passes.

neighborhoods building step $O_1^{l \oplus p}$ is:

$$O_1^{l \oplus p} = O(|V|^2 b + |E|l) = O(|V||E| + |E| \cdot l) \tag{4.15}$$

During the backward pass, NM builds all possible paths from the destination node using any exhaustive search methods such as EDFS or EBFS. *However, there is a difference in that, we process only those vertex neighbors which appear in its previous neighborhood*. This allows us to significantly reduce the total number of the processing paths: instead of processing $1+b+b^2+...+b^d$ or $O(b^d)$ paths, due to double pass (i.e., forward and backward passes) we process only: $1 \cap b^d + b \cap b^{d-1} + b^2 \cap b^{d-2} + ... + b^{\frac{d}{2}} \cap b^{\frac{d}{2}} + ... + b^{d-2} \cap b^2 + b^{d-1} \cap b + b^d \cap 1 \leq 1 + b + b^2 + ... + b^{\frac{d}{2}} + ... + b^2 + b + 1$ or $O(b^{\frac{d}{2}})$ paths. Hence, the time complexity of the backward pass step $O_2^{l \oplus p}$ is quadratically lower than for EDFS or EBFS (see Equation 4.14):

$$O_2^{l \oplus p} = O(2p \cdot b^{\frac{d}{2}}) = O\Big(\Big(\frac{|E|}{|V|}\Big)^{\frac{|V|}{2}} p\Big) \tag{4.16}$$

The total time complexity of the NM is $O\Big(\Big(\frac{|E|}{|V|}\Big)^{\frac{|V|}{2}} p + |V||E| + |E| \cdot l\Big)$ or just $O\Big(\Big(\frac{|E|}{|V|}\Big)^{\frac{|V|}{2}} p + |E| \cdot l\Big)$. Similarly as for EDFS and EBFS, the total space complexity is $O\Big(\Big(\frac{|E|}{|V|}\Big)^{\frac{|V|}{2}} p\Big)$.

4.4 Performance Evaluation

In this section, we evaluate NM's scalability and flexibility performance through simulations and our prototype implementation in the context of its applicability to several complementary virtual network services. To assess the flexibility of NM, we compare, with and without it: (i) the embedding performance of several online VNE and real-time NFV-SC mechanisms within the management plane; (ii) several traffic engineering solutions within the data plane. Our goal is to show how our NM can be used to improve the overall network utilization (allocation ratio or total flow throughput), optimality (load balancing or fairness of flows), as well as energy consumption within both planes; (iii) To assess NM scalability, we then compare it with related solutions under different network scales and service requests/topology scenarios when accepting multiple link and multiple path constraints.

Evaluation settings. In our simulations, we used a machine with an *Intel Core i5* processor with dual core CPU of 2.7 GHz and 8GB RAM. We use the BRITE [132] topology generator to create our physical and virtual networks. Our results are consistent across physical networks that follow Waxman and Barabasi-Albert models [133], that are known to approximate well subsets of Internet topologies [8]. For lack of space we only show results relative to Waxman connectivities. In our NM prototype evaluation instead, we use a physical network obtained with the GENI testbed [85]. All our results show 95% confidence interval, and our randomness lays in both the service request (i.e., in its type and constraints to accept) and in the physical network topology. In most of our physical topologies, the average node degree equals to 4, a known common value within Internet topologies [8].

Results summary. *Efficiency and Provider's revenue:* During the virtual network formation (management plane), we found that using NM within recent VNE or NFV-SC algorithms increases their allocation ratio while improving the network utilization by better load balancing (close to the optimal), which in turn decreases energy consumption. *Network Utilization:* Our results evaluating NM within the data plane instead show how utilizing a set of paths found with NM is beneficial for TE in terms of minimum path hop count,

network utilization and in some cases even energy consumption. *Time to Solution (Convergence Time):* When we attempted to allocate flows (virtual links) with multiple link and multiple path constraints over large scale physical networks using NM with the proposed search space reduction techniques, we found a path computation speed-up of almost an order of magnitude w.r.t. common exhaustive search algorithms. Moreover, we also found almost 3 orders of magnitude running time improvement w.r.t. the same integer programming problem solved with CPLEX [134]. This confirm how at large scale the time needed for a path computation with NM will have the same order of magnitude as the time needed for the path allocation introducing no significant bottleneck for the end-to-end virtual link embedding.

4.4.1 Management Plane Evaluation

Simulation settings. To assess the impact of NM on the virtual network embedding, we include results obtained with simulated physical networks of 20 nodes (as in similar earlier studies [61]), following Waxman connectivity model, where each physical node and each physical link have uniformly distributed CPU and bandwidth from 0 to 100 units, respectively. Note that we use fairly small scale physical networks due to complexity of the integer programming formulation. We attempt to embed a pool of 40 VN (service chain) requests with 6 virtual nodes and random (linear) virtual topologies. Each virtual node and each virtual edge have uniformly distributed CPU and bandwidth demand from 1 to 10 units, respectively. When we tested NM within virtual network embedding algorithms, we vary the virtual node degree from 1 (the VN has linear topology) to 5 (VN is a fully connected topology). Moreover, we also assume that each virtual edge has a propagation delay stretch latency constraint uniformly distributed between 1 to 4. We defined propagation delay stretch the ratio between the propagation delay and the propagation delay encountered traversing the diameter of the physical network. When we tested NM within the real-time service chaining use case, we vary the latency constraint of virtual links (service-to-service

communication) from ∞ (i.e., SC is not real-time sensitive) to 1/4 (i.e., SC is highly real-time sensitive) of the propagation delay stretch.

Evaluation metrics. To demonstrate the advantages of using NM within the virtual network embedding (VNE) and real-time service chaining (SC) mechanisms, we compared four representative VNE algorithms. To compare them, we replace their (original) Dijkstra-based shortest path with our NM. We have chosen to compare against [61, 60] as the optimal VNE scheme formulated as integer programming multi-commodity flow is the best to our knowledge solution for online VNE (yet intractable for large scale networks). We refer to this solution as Opt, as in Optimal. Opt in fact attempt to minimize the ratio between the provider's costs of embedding a VN request and the available substrate resources provided for this request, with the aim of balancing the network load. We also compare Opt against its version where a path (or column) generation approach is used to make Opt more scalable [61]. We refer to this scheme as PathGen and, even in this case, substitute its original shortest path algorithm (used to find new paths within the multi-commodity flow) with our path management solution (see details in Section 4.1.2). Note that we used the one-shot VNE approximation algorithm proposed in [121] as an initial solution for the column generation approach to avoid two stage VNE limitations when physical network is initially unbalanced [61]. Finally, we compare against a Consensus-based Auction mechanism (CAD) [74], the first policy-based distributed VNE approximation algorithm with convergence and optimality guarantees. The link embedding of CAD is a policy that runs a shortest path algorithm to either precompute the k-shortest paths or find these paths dynamically. For fairness of comparison, we assume the later holds, and as in PathGen case, we substitute currently used Dijkstra with our NM constrained shortest path finder.

In this simulation scenario we have tested the potential revenue loss by specifying the fraction of VN request accepted over the VN requested (allocation ratio), to what extent physical links were utilized (link utilization) and how many paths of the particular length in total were used per VN pool. Finally, we used the idle energy model proposed previ-

ously [5] to access the energy consumption of the network:

$$\text{Energy Consumption} = \sum_{e \in E} (M - E_0)U_e + E_0 \tag{4.17}$$

where E is a set of physical edges, U_e is an edge e utilization, and M and E_0 are numerical values taken from [5] that correspond to the maximum and idle energy consumption of the switch interface, respectively. We use $M = 2$ and $E_0 = 1.7$ maximum and idle energy consumptions (measured in Watts) assuming gigabit channel communications. In our results, we show an energy consumption increase relative to the idle network state (in %).

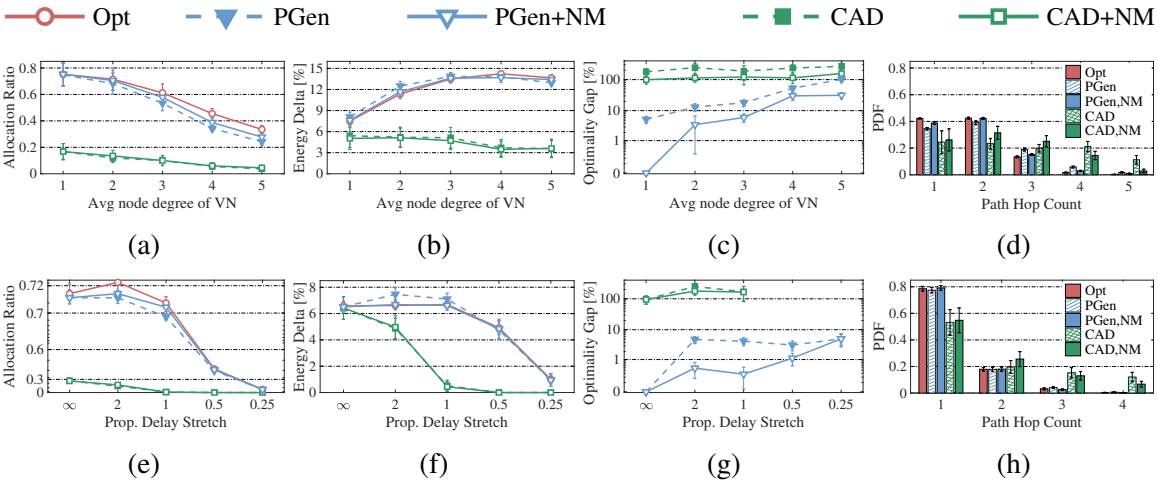


Figure 4.8: Virtual network (VN) embedding and real-time NFV service chaining (SC) results obtained with physical networks of 20 nodes following Waxman connectivity model: by addressing the constrained shortest path problem with NM versus using commonly adopted shortest path algorithms (e.g., Disjktra), the allocation ratio of VNE (a) and real-time NFV-SC (e) can be improved when the virtual node degree increase or when requests are not highly sensitive to delays. Optimality gap of VNE (c) and NFV-SC (g) can be also improved (resulting in our case into a better load balancing) which leads to a lower energy consumption (b) and (f) relative to the network idle state [5], respectively. NM's benefits are due to its ability of finding more path with a lower hop count that can satisfy latency demands and simultaneously improve objective value for full-mesh VNs (d) and moderate real-time sensitive SC (h) pools.

NM improves VNE/NFV-SC allocation ratio and energy efficiency. Figures $4.8a$ and $4.8e$ show how including constrained shortest paths (e.g., found with NM) during column generation of the PathGen approach can improve overall VNE and NFV-SC acceptance ratio. Particularly, the highest acceptance (i.e., allocation) ratio gains arise in the case of dense (e.g., full-mesh) VNs or under moderate real-time sensitivity of NFV-SCs. Moreover, we

can see how in some cases, utilizing NM within PathGen leads to a lower energy consumption (see Figures $4.8b$ and $4.8f$). This is due to an improved network utilization because of a better (closer to the optimal) load balancing (see Figures $4.8c$ and $4.8g$).

These optimality gains in turn arise due to NM's ability to find more paths that can satisfy all virtual link constraints (e.g., bandwidth and latency) and simultaneously improve the objective value (see Figures $4.8d$ and $4.8h$). These results confirm our expectations in Section 4.1.2. At the same time, optimality improvements with NM demonstrate no significant benefits (in terms of allocation ratio or energy efficiency) for the CAD over standard shortest path management. This is due to the fact that in our settings, separate node and link embedding approach demonstrates the worst performance caused by significantly limited feasible space for the virtual link mapping. Such limitations are due to randomized capacities of physical nodes and edges (i.e., due to initially unbalanced physical network) further exacerbated by randomized virtual link capacity and latency constraints.

4.4.2 Data Plane Evaluation

In the next set of results we analyze the benefits of using our solution within the data plane by evaluating NM performance within standard Traffic Engineering schemes [6, 7], under different physical network topologies and under different severity of service level objectives.

Simulation settings. To evaluate the impact of NM (used to compute constrained shortest paths) within Traffic Engineering [6, 7], we use a physical network topology of $10,000$ nodes, where each physical link has bandwidth uniformly distributed between 1 and 10 Gbps. We attempt to allocate flows for 1000 random source destination pairs by solving max-min fairness problem shown in Equation 4.8 with the fixed latency SLO demands. To evaluate the maximum possible gains, we assume infinite bandwidth demands of the flows and we omit any constraints imposed by hardware granularity due to rule count limits or flow quantization limitations [6, 7]. For clarity, we also assume that all flows have the same

priority. Thus, the fairness of the flow is its total allocated throughput. We denote with low, medium and high delay SLO constraints, 4, 1.5, and 1 times of a propagation delay stretch defined in Section 4.4.1, respectively.

In the first simulation scenario, we use LP-based solution (that is costly to address in practice [6, 7]) with fixed average physical node degree equal to 4 (common for the Internet [8]), where we vary the maximum number of paths available for each flow allocation. In the second scenario, we use instead scalable greedy solution proposed in [7] with unrestricted number of paths per flow. To this end, once the best currently available path (or tunnel) gets fully saturated, we find the next best path dynamically. In this scenario, we vary average physical node degree from 8 to 1.

Evaluation metrics. To evaluate the performance of NM for data plane TE solutions, we compare NM with (extended) Dijkstra algorithm within the greedy-based TE (i.e., in the second scenario), and their corresponding k-shortest path [119] and k-constrained shortest path (that uses general version of NM coupled with *Look Back* technique) algorithms within LP-based TE (i.e., in the first scenario). We remark that NM's superior performance w.r.t. Dijkstra-based TE is expected due to its ability of minimizing provider's associated cost for flow allocation e.g., minimizing provisioned physical bandwidth (see Problem 2) under an arbitrary set of (e.g., SLO) constraints. This difference is expected to degrade in the first scenario (where LP-based formulation is used) with either number of maximum paths per flow or SLO severity increase, as in this case the k-shortest path set converge to the k-constrained shortest path set resulting in the equal LP formulation. However, in the second scenario, as we allocate bandwidth for flows on their most preferable tunnels (best paths) first, that superior performance is expected to be preserved and can vary with different node degree or SLO severity. For comparison we use the following four metrics: total gained throughput of all flows, cumulative distribution of flows' throughput which corresponds to their fairness, energy consumption relative to the network idle state and path hop count.

Path hop savings lead to network utilization and flow fairness gains. In Figure $4.9a$, we

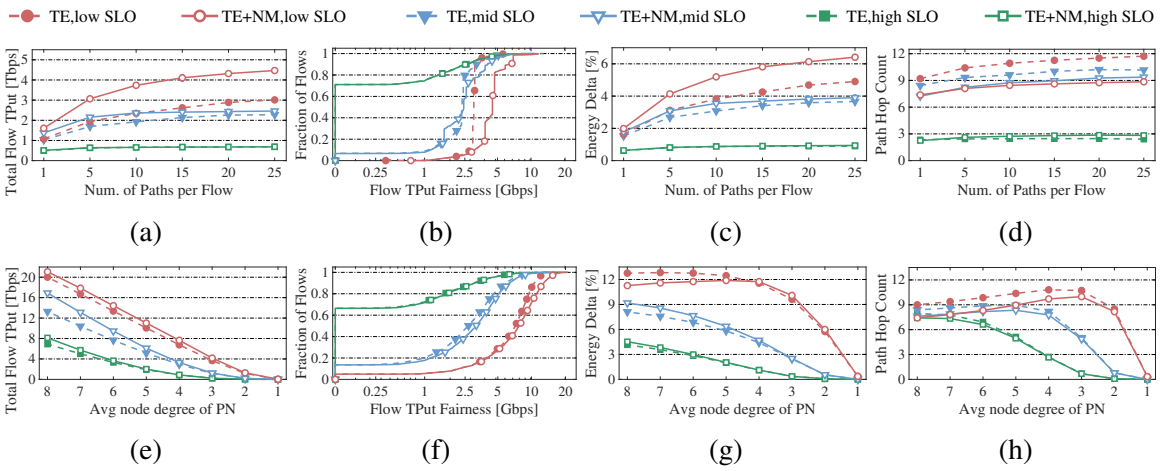


Figure 4.9: Performance analyses of LP-based (*Top*) and greedy (*Bottom*) max-min fairness Traffic Engineering (TE) algorithms [6, 7] utilizing the constrained shortest path management with NM versus their original shortest path management with Dijkstra on Waxman topologies in terms of: (a) and (e) total gained flow throughput; (b) and (f) cumulative distribution of flow throughput for 25 paths per flow and for average node degree 4 (common for Internet [8]), respectively; (c) and (g) energy consumption increase relative to the network idle state [5]; and (d) and (h) number of average path hops per flow.

show how the correlation between the total gained throughput across all flows and the maximum number of paths available per each flow is a logarithmic function — increasing number of paths linearly brings a logarithmic growth to the total gained throughput. Figure $4.9e$ shows how the total gained throughput of all allocated flows changes when multiple physical links become available (the average physical node degree increases). This dependence is an affine function: the maximum possible total gained throughput increases linearly with the available physical links.

In both scenarios, we can see how the total flow throughput and the resulting flow fairness (e.g., the particular flow throughput) are higher for NM than for Dijkstra-based TE (see Figures $4.9a$, $4.9b$, $4.9e$ and $4.9f$). These results demonstrate how minimizing the total physical bandwidth provisioned for a single flow with NM can significantly benefit traffic engineering solutions. In particular, due to the path hop count optimization under SLO constraints within the data plane, NM gains up to 50% of total flow throughput under low SLO in the first (LP-based) scenario, and up to 20% of total flow throughput under mid SLO in the second (greedy-based) scenario w.r.t. Dijkstra-based TE (see Figures $4.9a$ and $4.9e$).

Such gains allow, in turn also to improve flow fairness (see Figures $4.9b$ and $4.9f$). Note that such large throughput gains in the first scenario (i.e., up to 50%) are partly due to the k-shortest path [119] and the general version of NM (that finds k-constrained shortest paths) algorithms difference, i.e., the former has a higher probability of finding paths with more shared edges then the later. As expected, NM gains decrease with the node connectivity, as less physical path choices are available to map virtual links. Also, for LP-based scenario these gains decrease with increase of the maximum number of paths or SLO severity, as both shortest and constrained shortest path sets converge to each other resulting in equal LP formulations.

Average path length and energy consumption tradeoff. We further investigate the reasons why we observed such gains in total throughput of all flows w.r.t. Dijksra-based TE. In particular, observing Figures $4.9d$ and $4.9h$ we note that there are $\approx 2-3$ hops difference in the average path length between NM and Dijkstra-based TE when allocating low SLO flows. At the same time, for the medium SLO constraints this difference is reduced to circa one hop. Finally, when the SLO constraints are high, there is no significant physical path length difference. To understand why the average path length changes with the constraint severity, note how the longer is an end-to-end physical path, the lower is the probability that the entire path satisfies the SLO constraints. On the other hand, the higher the number of hops, the higher is the number of candidates paths, and so the higher is the probability of finding one which satisfies these constraints. This explains the trade-offs in average path length behavior observed in Figure $4.9h$.

The hop count savings minimize the physical bandwidth provisioned for a single path, allowing the provider to accept more flows or allocate more bandwidth for a single flow. As a result, the overall link utilization increases leading to a higher energy consumption (see Figures $4.9c$ and $4.9g$). We observe one exception when throughput gains are low and path hop count savings are high (as observed in the management plane scenario in Section 4.4.1). An example of such situation can be also observed in the second scenario

for dense physical networks (with average node degree ≥ 5) when allocating flows with low SLO demands. In that case, we can see a small reduction (of $\approx 2\%$) in the energy consumption simultaneously with low throughput gains (of $\approx 5\%$).

4.4.3 Scalability Results

In the next set of results we test the scalability performance of NM when accepting multiple link and multiple path constraints ($l \oplus p$ case). We remark that in this case only exponential exact solutions exist for the constrained shortest path problem due to its NP-hardness [53].

Scalability simulation settings. To assess NM scalability, we simulate on-line requests for allocating constrained virtual links (or traffic flows). In particular, we generate physical network topologies of 10, 100, $1K$ and $10K$ nodes, where each physical link has bandwidth uniformly distributed between 1 and 10 Gbps. In addition, we set each physical link with a cost uniformly distributed between 1 and 10. We attempt to find the constrained shortest path variant — the resource optimal constrained path for 10% of physical network nodes random source-destination pairs, where for each pair we allocate as many virtual links as possible with the fixed demands. We denote with low and medium bandwidth constraints, 1 and 4 Gbps, respectively; these values represent approximately 10% and 45% of the maximum physical link capacity. Similarly, we denote with low and medium (propagation) delay SLO constraints, 4 and 2.5 times of a propagation delay stretch defined in Section 4.4.1, respectively. In addition, we denote with low and medium cost constraints, 100 and 50 that represent 10 and 5 times of the maximum physical link cost.

Scalability evaluation metrics. To evaluate the NM scalability, we compare the general version of NM with the EBFS (common branch-and-bound exhaustive search) algorithm and with the CPLEX [134] performance (that uses 4 parallel threads) of solving the common arc-based constrained shortest path formulation. Both NM and EBFS are coupled with dominant paths search space reduction techniques. Moreover, we couple EBFS with a Look Ahead (EBFS+LA) search space reduction technique [55] and NM with a Look Back

(NM+LB) search space reduction technique - a variant of Look Ahead without complexity overhead (see Section 4.2.2).

We compare NM with EBFS and CPLEX across two metrics: the number of traversed paths required to find the constrained shortest path and the average path computation time. Note that in case of CPLEX, the number of traversed paths corresponds to the total number of iterations.

Dominant paths prevent intractabilities. Figures $4.10a$ and $4.10c$ show that dominant paths technique reduces the number of traversed paths per virtual link to a linear function of a physical network size for both EBFS and NM. Moreover, due to its "double pass" technique, NM traverses up to two orders of magnitude paths less then EBFS. However, we can see how EBFS works slightly faster than NM for large scale physical networks with medium link and path constraints (see Figure $4.10d$). That can be explained with the more expensive forward pass of NM for larger physical networks ($\geq$ 10K nodes). The NM's unnecessary iterations can be however reduced by the proposed Look Back technique.

NM scalability with backward pass and look back. Our experiments show that when NM backward phase is coupled with a Look Back technique, the number of traversed paths by NM further reduces. This reduction is almost independent from the size of the physical network (see Figures $4.10a$ and $4.10c$). Using the Look Back search space optimization does not introduce any significant path computation overhead, while the same cannot be said for the Look Ahead search space reduction technique (see Figures $4.10b$ and $4.10d$).

Even though CPLEX uses 4 parallel threads (instead of a single thread for NM and EBFS) and traverses moderate number of path (similar to NM without Look Back technique), it shows the worst performance in all cases. That is due to the fact, that finding constrained shortest paths with the commonly utilized arc-based integer programming formulation is NP-hard and no existing techniques can reduce that complexity to pseudo-polynomial. On the contrary, NM and EBFS complexities can be reduced to pseudo-polynomial by applying the dominant paths search space reduction technique [55]. Pro-

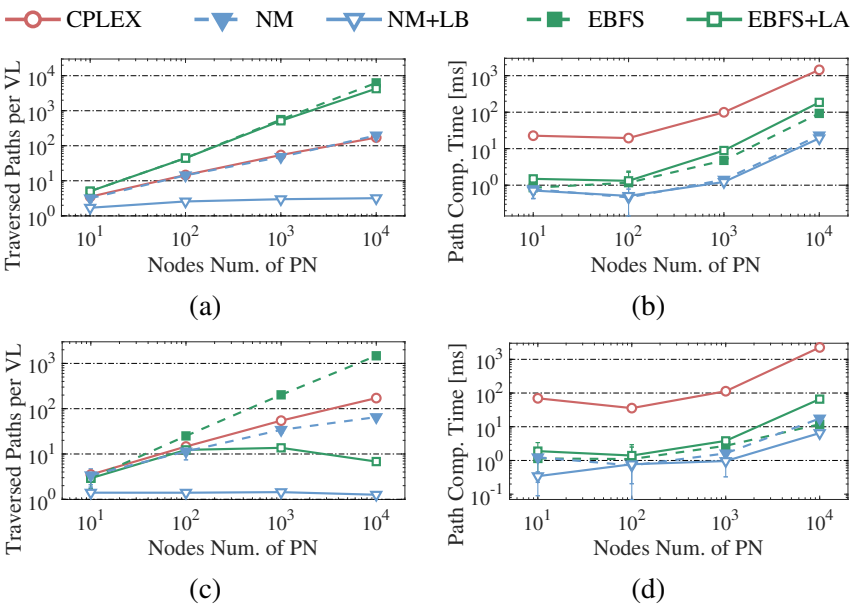


Figure 4.10: Performance analyses of general NM versus EBFS with dominant paths, look-back for NM (NM+LB) and look-ahead for EBFS (EBFS+LA) search space reduction techniques, and versus IBM CPLEX solver for the constrained shortest path formulation in Problem 2 for low (top row) and medium (bottom row) SLO constraints in terms of: (a,c) number of traversed paths to find the *optimal* virtual path; (b,d) the virtual path computation time.

posed novel *double pass* and *Look Back* search space reduction techniques further reduce the practical complexity of finding constrained shortest paths. Thus, NM is almost an order of magnitude faster in comparison with EBFS and is almost 3 orders of magnitude faster than CPLEX for large-scale physical networks, and hence *scales* better. We remark that such *scalability* improvements over existing constrained shortest path algorithms are essential for the virtual network service management at large scale. For example, the column generation approach can generate tens of thousands paths per a single VN request at large scale. Thus, it will take 100 ms x $10K \approx 17$ minutes when EBFS is used. On the contrary, we need only 10 ms x $10K \approx 100$ seconds with NM which significantly reduces VN request blocking probability.

Chapter 5

Data Computation/Consumption and Reliable Service Chain Orchestration

"Dīvide et imperā"

(divide and rule)

— Philip II of Macedon

In the previous chapter, we have shown the constrained shortest path importance for the virtual network service management. What follows next is the first-of-its-kind metapath composite variable approach that significantly reduces combinatorial complexity of the optimal service function chaining. We remark that we use service chaining of computer vision functions as part of the function-centric computing paradigm for data computation/consumption over geo-distributed edge/core cloud infrastructures. The idea is to divide the (larger) service chaining problem into a set of (smaller) constrained-shortest (meta)path problems (that can be solved efficiently with our state-of-the-art path finder) to aggregate multiple optimization decisions within a single binary variable. However, before digging into details of our metapath composite variable approach we first discuss potential challenges of decoupling existing computer vision applications into a set of functions. We then describe in more details the problem of reliable service chain orchestration, i.e., its initial composition and the consequent maintenance.

5.1 Challenges of Computer Vision Function Decoupling

In this section, we first outline the function decoupling challenges of computer vision applications that can be used for geospatial video analytics. We then show the need for a reliable function chain orchestration mechanism due to computer vision application application and infrastructure challenges that need to be addressed in order to apply the function-centric computing paradigm.

5.1.1 Application Challenges: Situational Awareness in a Disaster Scenario

As we have seen in Section 1.2, computer vision and other computationally intensive applications are typically designed without portability in mind: their code is tightly coupled to the rest of the system to optimize performance. This, in turn, limits the applications' ability to scale seamlessly on a distributed set of physical (or logical) resources.

We can decouple imagery/video-processing into functions, leveraging their natural decomposability into sub-processes. For instance, our tracking use case application can be decomposed to the following sub-processes: (a) acquisition, (b) pre-processing, (c) analysis, and (d) post-processing. The feasible decoupling of imagery/video-processing subprocesses into functions for our theater-scale and regional-scale application case studies is shown in Figures 1.3, 1.5 and 1.7. We can divide computer vision functions into two main classes, depending on their functionality: *small* and *large* function processing as discussed in Section 1.2. Small functions can be placed at the edge cloud (i.e., near the infrastructure edge) or as network functions at the core infrastructure network; large functions instead typically need to be processed in the core cloud (see Figure 1.8).

Challenge 1: Computation and storage demand uncertainty. A first challenge of application decoupling into functions is demand uncertainty. Disaster scenes have different scales and the amount of collected data at a disaster scene is typically unknown in advance.

Table 5.1: An example set of computer vision application demands for different geo-scales of data collection

Data Scale	Expected Computation Demand per Function	Expected Data Collection Size	Expected Computation Time	Expected Data Rate
Theater-scale	0.5-5 TFlops	1-10 GB	10-100 ms per frame	10-100 Mbps
Regional-scale	1-10 TFlops	10-100 GB	10-100 ms per frame	0.1 - 1 Gbps

For example, at the theater i.e., medium-scale, scenes are characterized by very large collection of distributed light-weight to medium resolution ground-based cameras (including fixed-mount surveillance cameras, mobile smartphones, body-worn cameras, and robotic tele-operated cameras). At the larger regional-scale, in our exemplar case study application, a small number of aerial imaging platforms are collected with high-resolution (multi-spectral) day-night imagery to rapidly assess damage, and monitor movement of vehicles and people to route assistance to the most needed sites. Table 5.1 lists a few exemplar task characteristics of computer vision applications and their anticipated resource usage requirements across the two geographical scales. Response times vary based on the allocation of hardware resources at the infrastructure edges, and also if caching techniques are used at the infrastructure edges near visual data consumption sites for improving the frame fetch latency of visual data when requested in an on-demand manner by the first responders. Data sizes may vary depending upon the resolutions of collected images/video content, and computation times may vary as well, dependent on the hardware used for the visual data processing.

5.1.2 Infrastructure Challenges: Edge Computing and Reliable Service Chain Orchestration

Small instance functions such as encryption and features extraction are common for many computer vision applications; such applications may benefit from function-centric computing if the computations occur within the network itself. Certain functions can be placed as 'network functions' to software switches [135, 136]. At the same time, other functions still

need to be processed on cloud servers either in the core/public or edge clouds. To avoid manual decisions of *how* application's sub-processes need to be offloaded and *where* to offload them, we aim to build upon NFV service chain orchestration concepts tailored to our function-centric computing paradigm.

Challenge 2: Service Chain Representation and NP-hard Composition. The first step is to abstract the (computer vision) application services via a chain representation to form service chain requests. Service chain requests then need to be composed from either network functions, network-edges or core/public cloud resources. To overcome this challenge, we propose a reliable service chain orchestration approach that composes and maintains service chains under varying application demands and physical infrastructure uncertainties caused e.g., by disaster incidents. The service chain composition is the NP-hard graph matching problem of composing a constrained service chain on top of a shared constrained physical network [73]: each application's service needs to be mapped onto a physical node (i.e., service placement). In order to enable communications between a pair of services, we need to find a loop-free path (i.e., service chaining).

Challenge 3: Infrastructure Outages. Due to the fact that the response time is a vital factor during disaster relief activities, providing real-time visual awareness with latency-aware mechanisms is imperative. Satisfying latency requirements even at the application level may require computation of some functions at the edge. This, in turn not only increases the input size of an already NP-hard constrained graph matching problem, but also brings new resource provisioning and management challenges. For instance, large-scale cloud/edge infrastructures can be subject to outages which leads to the available physical resource uncertainties. In addition, infrastructure outages can be caused by an ongoing disaster incident [3]. Hence, a reliable service chain composition approach should also account for available physical resource uncertainties caused by infrastructure outages.

5.2 Modeling Reliable Service Chain Composition

In this section, we define the problem of joint (and reliable) *service chain composition* that can be formulated as the integer multi-commodity-chain flow problem for an augmented cloud infrastructure graph [69] which is a generalization of a well-known multi-commodity flow problem [60]. To proactively ensure reliability of a service chain composition, we use backup policies as well as probabilistic 'chance' capacity constraints instead of deterministic ones. Thus, we use a chance-constrained programming [137]. We also extend this problem with geo-location and latency constraints to satisfy all QoS demands of geo-distributed latency-sensitive service function chains.

Objective and example of the online chain composition. Based on providers' policies, the service chain composition problem can be used to minimize (expected) values of different fitness functions $F_{\mathbb{E}}$. One example of common fitness functions is an additive function of service chain demands and corresponding physical resource capacity ratios. Such function is known to best balance the physical network load [60]. In most cases service chain requests can be unknown in advance, and using the load balancing fitness function allows one to increase the acceptance ratio of these requests. Such optimization is also known as the 'online optimization' [60, 61].

Figure 5.1 shows an example of the online service chain composition that minimizes the network load balancing function: by minimizing a sum of service chain demands and corresponding physical resource capacity ratios, for $a - b - c$ service chain we achieve its minimum value $F_{\mathbb{E}} = \frac{8}{10} + \frac{2 \cdot 1}{5} + \frac{8}{10} + \frac{1}{5} + \frac{4}{5} = 3$. As a result, we compose this service chain request with X, Y and A physical nodes (e.g., servers) to place a, b and c services, respectively. To enable service communications $a-b$ and $b-c$, we chain them with $X-B-Y$ and $Y - B$ physical paths, respectively. This also allows us to compose the subsequent $d - e$ request. In the rest of this chapter, our objective is to minimize the expected value of the load balancing fitness function $F_{\mathbb{E}}$ formally defined below (see Equation 5.1) for the case of service chain demand and available physical resource uncertainties.

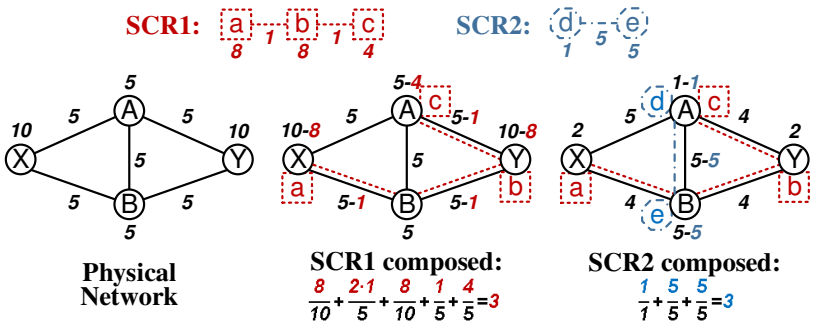


Figure 5.1: Illustrative example of the *online composition* of service chain requests (SCR) on top of the capacitated physical network (numbers indicate service demands and corresponding resource capacities).

Service chain composition sets and variables. We model each service chain $a \in A$ as a chain graph $G^a = (N_V^a, E_V^a)$. A service G^a is composed by a set N_V^a of services and a set E_V^a of corresponding service communications (or service links) representing logical network connectivities among elements in N_V^a. Moreover, each service chain a has a set of backup resources B^a. We then model the physical infrastructure on which the service functions run as a physical network graph $G = (N_S, E_S)$, composed by a set N_S of substrate nodes and a set E_S of substrate edges.

We define two types of binary variables: one for the service chain link mapping, and another for (node) service mapping. Particularly, let binary variable $f_{ij}^{st}(b, a) = 1$ if a flow for $st \in E_V^a$ service link of a backup $b \in B^a$ of a service chain $a \in A$ is assigned to the physical edge $ij \in E_S$, i.e., $f_{ij}^{st}(b, a) = 1$, or 0 otherwise. Furthermore, let binary variable $x_i^s(b, a) = 1$ if a service $s \in N_V^a$ of a backup $b \in B^a$ of a service chain $a \in A$ is assigned to the physical node $i \in N_S$, i.e., $x_i^s(b, a) = 1$, and 0 otherwise. Having sets and variables defined, we now can formulate the online service chain composition problem under uncertainty using integer multi-commodity-chain flow problem.

Problem 3 (online service chain composition under uncertainty)**.** *Given a set of service chains represented as graphs $G^a = (N_V^a, E_V^a)$ and a physical network graph $G = (N_S, E_S)$, the online service chain composition problem under service chain demands and available physical resources uncertainties can be formulated as follows:*

$$\min F_{\mathbb{E}} = \sum_{a\in A}\sum_{b\in B^a}\left(\sum_{s\in N_V^a}\sum_{\tau\in T}\sum_{i\in N_S}\mathbb{E}\left[\frac{\mathbf{D}_s^{a\tau}}{\mathbf{C}_i^{\tau}}\right]x_i^s(b,a) + \sum_{st\in E_V^a}\sum_{ij\in E_S}\mathbb{E}\left[\frac{\mathbf{D}_{st}^{a}}{\mathbf{C}_{ij}}\right]f_{ji}^{st}(b,a)\right) \quad (5.1)$$

subject to

Service Placement Constraints:

$$\sum_{i\in N_S} x_i^s(b,a) = 1, \forall s \in N_V^a, b \in B^a, a \in A \quad (5.2)$$

$$\mathbb{P}\left[\sum_{a\in A}\sum_{b\in B^a}\sum_{s\in N_V^a}\mathbf{D}_s^{a\tau}x_i^s(b,a) \leq \mathbf{C}_i^{\tau}\right] \geq R, \forall i \in N_S, \tau \in T \quad (5.3)$$

Service Chaining Constraints:

$$\sum_{j\in N_S} f_{ij}^{st}(b,a) - \sum_{j\in N_S} f_{ji}^{st}(b,a) = x_i^s(b,a) - x_i^t(b,a), \forall i \in N_S, st \in E_V^a, b \in B^a, a \in A \quad (5.4)$$

$$\mathbb{P}\left[\sum_{a\in A}\sum_{b\in B^a}\sum_{st\in E_V^a}\mathbf{D}_{st}^{a}f_{ij}^{st}(b,a) \leq \mathbf{C}_{ij}\right] \geq R, \forall ij \in E_S \quad (5.5)$$

Specific QoS Constraints (Geo-Location, Latency, etc.):

$$gd_{si}^a x_i^s(b,a) \leq \mathcal{GD}_s^a, \forall i \in N_S, s \in N_V^a, b \in B^a, a \in A, \quad (5.6)$$

$$\sum_{ij\in E_S} w_{ij}^k f_{ij}^{st}(b,a) \leq \mathcal{K}_{st}^{ak}, \forall st \in E_V^a, b \in B^a, a \in A, k \in K \quad (5.7)$$

Additional Policy Constraints (e.g., No-Consolidation):

$$\sum_{b\in B^a}\sum_{s\in N_V^a} x_i^s(b,a) \leq 1, \forall i \in N_S, a \in A \quad (5.8)$$

where all symbols and notationsare summarized in Appendix B.1.1.

Service chain composition constraints discussion. Minimization of $F_{\mathbb{E}}$ in Equation 5.1 is subject to a set of constraints which contains both — basic composition constraints and constraints specific to the geo-distributed latency-sensitive service chains. The basic constraints include service *placement* for a specified number of duplicates (Equation 5.2), service *chaining* or well-known multi-commodity flow constraints (Equation 5.4). Additional policy constraints for service chain composition problem are also acceptable. One such example is a common 'no consolidated service placement' constraint that prohibits placement of two or more different services (or their backups) belonging to the same service chain onto one physical node (Equation 5.8). Note that this policy further complicates a combinatorial complexity of the (NP-hard) integer multi-commodity-chain flow problem. In contrast to prior service chain composition problems [69, 82], we now use probabilistic physical node and link capacity constraints to ensure that physical resources satisfy service chain QoS demands given some acceptable risk (Equations 5.3 and 5.5).

The specific geo-distributed latency-sensitive service chain constraints include physical node geo-location and service communication end-to-end network QoS constraints such as latency, packet loss, etc. (Equation 5.7).

Objective and chance-constraint deterministic equivalents. Due to physical node and link outage risks, their capacities are random discrete variables. Taking this into account and with the legitimate assumption of the worst-case scenario that all service chain demands follow a normal distribution and are strongly correlated, we can substitute our objective and chance-constraints in Equations 5.3 and 5.5 with the following linear deterministic equivalents[1]:

$$F{=}\sum_{a\in A}\sum_{b\in B^a}\left(\sum_{s\in N_V^a}\sum_{\tau\in T}\sum_{i\in N_S}\frac{\mu_s^{a\tau}+K_{\frac{R}{P_i}}\sigma_s^{a\tau}}{C_i^{\tau}}x_i^s(b,a){+}\sum_{st\in E_V^a}\sum_{ij\in E_S}\frac{\mu_{st}^{a}+K_{\frac{R}{P_{ij}}}\sigma_{st}^{a}}{C_{ij}}f_{ji}^{st}(b,a)\right) \quad (5.9)$$

[1]Note that such assumption is also valid when service chain demands are not correlated or their correlation coefficients are unknown. In these cases, linear deterministic equivalents of chance-constraints always satisfy availability requirements R and can result in even higher (actual) availability at expense of worse resource utilization.

$$\sum_{a\in A}\sum_{b\in B^a}\sum_{s\in N_V^a}\left(\mu_s^{a\tau}+K_{\frac{R}{P_i}}\sigma_s^{a\tau}\right)x_i^s(a,b)\leq\begin{cases}C_i^\tau,\ R<P_i\\0,\text{otherwise}\end{cases},\quad\forall i\in N_S,\tau\in T \tag{5.10}$$

$$\sum_{a\in A}\sum_{b\in B^a}\sum_{st\in E_V^a}\left(\mu_{st}^{a}+K_{\frac{R}{P_{ij}}}\sigma_{st}^{a}\right)f_{ij}^{st}(a,b)\leq\begin{cases}C_{ij},R<P_{ij}\\0,\text{otherwise}\end{cases},\quad\forall ij\in E_S \tag{5.11}$$

where $K_{\frac{R}{P_{ij}}}$ is a constant in a standard Normal distribution table, $R\in(0,1)$ is the service chain reliability that reflects an acceptable risk, and P_i (or P_{ij}) is a probability that physical node i (or link ij) is available, i.e., $\mathbf{C_i^\tau}=C_i^\tau$ (or $\mathbf{C_{ij}}=C_{ij}^\tau$).

The detailed derivation of the above constraints can be found in the Appendix B.1.2, and it is valid for application demands that follow normal distribution. Note that if service chain demands do not follow normal distribution, deterministic equivalents of Problem 3 objective and chance-constraints can be easily generated, provided that the probability distributions are known.

Service chain reliability and chance-constraints discussion. It is known that chance-constrained programming is less effective than multi-stage recourse programming to model uncertainties [137]. This is because to provide the same reliability level chance-constrained service chain composition under-provisions more physical resources than its recourse programming alternative. On the other hand, solving a recourse program for the service chain composition is intractable even with moderately small network sizes. This is due to the fact that solving it requires computations over an exponential number of scenarios, i.e., the problem is equivalent to an integer program of exponential size [82]. To avoid considering an exponential number of scenarios, we use a policy-based reliability for the service chain composition instead. Specifically, we allow for policy specifications of chance-constraints acceptable risks and service backups.

For instance, by decreasing an acceptable risk and/or increasing number of backups, we can leverage the overall probability of a service chain disruption that requires its re-composition (e.g., migration of virtual resources) during its maintenance. For example, given a risk of 5%, i.e., $R = 0.95$, and 5 services for a single service chain, the lower bound probability that its demands will be satisfied is $P_{lb} = R^5 = 0.95^5 \approx 0.77$ not considering inter-service communication demands and not allowing backup resources. Thus, approximately in 1 out of 5 cases the service chain needs to be re-composed. Alternatively, if we at least duplicate the service chain physical resources (i.e., compose 2 service chain backups), the lower bound probability that service chain demands will be satisfied by at least one of the duplicates becomes $P_{lb} = 1 - (1 - R^5)^2 \approx 0.95$. As a result, the service chain needs to be re-composed only in 1 out of 20 cases. However, with tighter reliability policies (i.e., the lower acceptable risk or the higher number of backups), fewer feasible solutions are available, and the optimal solution achieves a poorer objective value of the optimal solution, and thus, worse performance of the online service chain composition is realized. We show such reliability/performance trade-offs of our approach using trace-driven simulations in Section 5.5.

Integer multi-commodity-chain flow problem intractabilities. When deterministic equivalents of the objective in Equation 5.1 as well as of capacity chance-constraints are known, we can use any integer programming solver (e.g., CPLEX [134]) for Problem 3 to reliably compose all (known at a time) service chain requests. However, due to NP-hardness of this composition, the solution can be intractable for large-scale geo-distributed (edge/core) cloud infrastructures. To improve its scalability limitations, existing column generation [61], heuristic and metaheuristic [82] approaches can be used (often at expense of the master problem optimality). In the next section, we propose a near optimal *metapath composite variable* approach that simplifies a combinatorial complexity of the service chain composition outlined in Problem 3.

5.3 Service Chain Composition via Metapaths

In this section, we outline our previously proposed *metapath composite variable* approach that aims to simplify the combinatorial complexity of the integer multi-commodity-chain flow-based service chain composition problem [17]. Thus, similarly to existing composite variable schemes [79], we create a binary variable that composes multiple (preferably close to optimal) decisions. To this end, we build upon a known result in optimization theory: all network flow problems can be decomposed into paths and cycles [138]. We first introduce our notion of *metapath* and its relevance to the constrained shortest path problem [52, 53, 54, 55]. We then use the constrained shortest metapaths to create variables with composite decisions for the service chain composition problem and discuss scalability improvements of this approach.

Metalinks and metapaths. Before defining the metapath, it is useful to introduce the idea of 'metalinks'. Metalinks have been widely adopted in prior NFV/VNE literature to solve optimally graph matching problems [69, 61]. A metalink is an augmentation link in a network graph. In our case, it represents the (potential) *feasible* placement of some service a on some physical node A, as shown in Figure 5.2. Formally, we have:

Definition 2 (metalink)**.** *A link si for service chain $a \in A$ belongs to the set of metalinks E_M^a if and only if the service $s \in N_V^a$ of service chain a can be placed onto the node $i \in N_S$.*

Building on the definition of a metalink, we can define a metapath as the path that connects any two services through the physical network augmented with metalinks. For example, consider following metapaths $a - A - Y - b$ and $a - A - B - b$ shown in Figure 5.2. Formally, we have:

Definition 3 (metapath)**.** *The path P_{ij}^{st} is a metapath between services s and t for service chain $a \in A$ if and only if $\forall kl \in P_{ij}^{st} : kl \in E_S \vee kl \in \{si, tj\}$.*

Intuitively, metapath P_{ij}^{st} is formed by exactly two metalinks that connect s and t to the physical network and an arbitrary number of physical links $kl \in E_S$.

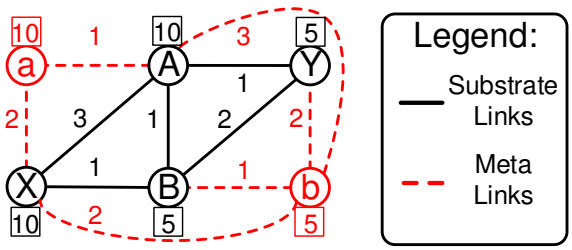


Figure 5.2: Illustrative example of the augmented with metalinks physical network which represent feasible service a, b and c placements; numbers indicate fitness function values - red and black values annotate service placement and service chaining via some physical link, respectively. Assuming 'no consolidation' policy, constrained shortest metapaths $a-A-Y-b$ and $b-B-X-c$ represent optimal single-link chain $a-b$ and $b-c$ compositions, respectively.

Constrained shortest metapaths. Having defined metapaths, let us consider a simple case of the service chain composition problem - composition of a single-link chain (i.e., two services connected via a single virtual link): the optimal composition of a single-link chain can be seen as the *constrained shortest (meta)path* problem that connects two services via the augmented physical network, where all physical links have arbitrary fitness values of a service chaining (virtual link mapping) and all metalinks have arbitrary fitness values of a service placement divided by the number of neighboring services (i.e., by 1 for a single-link chain). In our example, shown in Figure 3, the optimal single-link service chain $a-b$ composition can be represented by the constrained shortest metapath $a-A-Y-b$ that satisfies all service chain composition constraints with the overall fitness function of 3. Further, we prove our intuition formally:

Theorem 5.3.1. (The optimal single-link service chain composition) *The optimal single-link chain composition is the constrained shortest metapath.*

Proof. In Appendix B.2.1 we show how the optimal single-link chain composition corresponds to the problem of finding the constrained shortest path between two services in the augmented with metalinks substrate network graph, where all metalinks have cost of service placement and all substrate links have cost of a virtual link mapping. ■

Corollary 5.3.1. (The optimal single-link service chain composition complexity) *The optimal single-link service chain composition has a pseudo-polynomial complexity.*

Proof. Based on Theorem 5.3.1, the optimal single-link service chain composition is the constrained shortest metapath which is by Definition 3 the constrained shortest path in the augmented network graph. However, it is known that the constrained-shortest path can be found in pseudo-polynomial time [55]. ■

We conclude that constrained shortest metapaths are good candidates to perform composite decisions, i.e., to optimally decide on a single-link service chain composition in terms of its services placement and chaining with a single binary variable.

Multiple-link chain composition via metapath. While observing Figure 5.2, we can notice how using only a single constrained shortest metapath per a single-link segment of a multiple-link service chain $a-b-c$ can lead to an unfeasible composition: as the optimal $a-b$ composition is $a-A-Y-b$ metapath, and the optimal $b-c$ composition is $b-B-X-c$ metapath - service b has to be simultaneously placed on Y and B physical nodes. Thus, we cannot *stitch* these metapath, and we need to find more than one constrained shortest metapath per a single-link chain. *In our composite variable approach, we find k-constrained shortest metapaths (to create k binary variables) per each single-link segment of a multi-link service chain.* To find metapaths any constrained shortest path algorithm can be used [52, 53, 54, 55]. In this paper, we build upon the path finder proposed in our previous chapter that is an order of magnitude faster than recent solutions [52].

To further benefit from constrained shortest metapaths and simplify the chain composition problem, we *offload* its constraints (either fully or partially) to either metalinks or the path finder. Specifically, *geo-location and an arbitrary number of end-to-end network (e.g., latency) QoS constraints can be fully offloaded to metalinks and to the path finder, respectively*. At the same time, capacity constraints of the service chain composition problem are global and can be only partially offloaded. Once k-constrained shortest paths have been found for each single-link service chain segment, we can solve GAP problem [80] to assign each single-link chain segment to exactly one constrained shortest metapath and *stitch* these metapaths as described below.

Allowable fitness functions for metapath-based variables. In general, fitness functions qualify for our *metapath composite variable* approach if they are comprised from either additive or multiplicative terms. The above requirement fits for most service chain objectives [65], and other objectives can also qualify if well-behaved (e.g., if their single-link chain fitness values can be minimized by a path finder). As the load balancing fitness function $F_{\mathbb{E}}$ in Equation 5.1 qualifies, we compute its single-link chain value $\mathbb{E}\left[F_{ijk}^{sta}\right]$ for k metapath as following:

$$\mathbb{E}\left[F_{ijk}^{sta}\right] = \sum_{\tau \in T} \mathbb{E}\left[\frac{\mathbf{D}_s^{a\tau}}{\mathbf{C}_i^{\tau}}\right] / deg(s) + \sum_{st \in E_V^a} \sum_{\substack{\{uv \in E_S: \\ uv \in P_{ijk}^{sta}\}}} \mathbb{E}\left[\frac{\mathbf{D}_{st}^{a}}{\mathbf{C}_{uv}}\right] + \sum_{\tau \in T} \mathbb{E}\left[\frac{\mathbf{D}_t^{a\tau}}{\mathbf{C}_j^{\tau}}\right] / deg(t), \tag{5.12}$$

where $deg(s)$ corresponds to the service s degree, i.e., $deg(s) = 1$ for $s \in \{in, out\}$ services that handle input and processed output data of service chains, respectively; and $deg(s) = 2$ otherwise. Note that in NFV in and out are dummy services that corresponds to the flow source and sink physical nodes and have no computation demands. The first and the last terms represent the fitness values of metalinks, and the middle term corresponds to the sum of physical links' fitness values. Note also that other examples of additive and multiplicative fitness functions can be found in our Appendix B.2.2.

Problem 4 (Service chain composition via metapaths)**.** *Given a set of service chains $a \in A$ represented as graphs $G^a = (N_V^a, E_V^a)$, a physical network graph $G = (N_S, E_S)$, and having set of k-constrained shortest metapaths $P_{ijk}^{sta} \in \mathcal{P}_a^{st}$ and their corresponding fitness function values F_{ijk}^{sta} found for each virtual link $st \in E_V^a$ in the service chain a, let a binary variable $f_{ijk}^{st}(b, a) = 1$ if the single-link chain segment st is assigned to the metapath P_{ijk}^{sta} of the backup $b \in B^a$ of service chain $a \in A$, or 0 otherwise. The service chain composition problem via metapaths can be formulated as follows:*

$$\min \sum_{a \in A} \sum_{b \in B^a} \sum_{st \in E_V^a} \sum_{P_{ijk}^{sta} \in \mathcal{P}_a^{st}} \mathbb{E}\left[F_{ijk}^{sta}\right] f_{ijk}^{st}(b, a) \tag{5.13}$$

subject to

Metapath Stitching (Assignment) Constraints:

$$\sum_{P_{ijk}^{sta} \in \mathcal{P}_a^{st}} f_{ijk}^{st}(b,a) - \sum_{P_{jik}^{tsa} \in \mathcal{P}_a^{ts}} f_{jik}^{ts}(b,a) = \begin{cases} -1, & t = in \\ 1, & t = out \\ 0, & otherwise \end{cases} \tag{5.14}$$

$$\forall t \in \{in, out\} \vee tj \in E_M^a, b \in B^a, a \in A$$

Node Capacity Chance-Constraints:

$$\mathbb{P}\left[\sum_{a \in A} \sum_{b \in B^a} \sum_{t \in N_V^a} \mathbf{D}_t^{a\tau}/deg(t) \left(\sum_{P_{ijk}^{sta} \in \mathcal{P}_a^{st}} f_{ijk}^{st}(b,a) + \sum_{P_{jik}^{tsa} \in \mathcal{P}_a^{ts}} f_{jik}^{ts}(b,a) \right) \leq \mathbf{C}_j^{\tau} \right] \geq R, \tag{5.15}$$

$$\forall j \in N_S, \tau \in T$$

Link Capacity Chance-Constraints:

$$\mathbb{P}\left[\sum_{a \in A} \sum_{b \in B^a} \sum_{st \in E_V^a} \sum_{\substack{\{P_{ijk}^{sta} \in \mathcal{P}_a^{st}: \\ uv \in P_{ijk}^{sta}\}}} \mathbf{D}_{st}^a f_{ijk}^{st}(b,a) \leq \mathbf{C}_{uv} \right] \geq R, \tag{5.16}$$

$$\forall uv \in E_S$$

where symbols and notations of sets, parameters, variables and functions are summarized in Appendix B.1.1.

We remark that in and out are services that handles input and processed output data of service chains, respectively. Note also that deterministic equivalents for the objective coefficients and capacity constraints in Equations 5.13, 5.15 and 5.16 are similar to deterministic equivalents of Problem 3.

5.3.1 Service Chain Composition via Lagrangian Relaxation

We found that our metapath-based service chain composition (see Problem 4) scales well in practice - for both US Tier1 and US Regional providers' network scales (300 and 600 nodes) our metapath-based service chain composition reaches 99% optimality in average and takes ≤ 1 minute in average to compose a large chain of 20 services which is approximately 3 orders of magnitude faster than the service composition in Problem 3. However, in addition to solving the NP-hard GAP Problem 4, in this thesis we also propose its better scalable alternative. Particularly, we can solve GAP problem using its Lagrangian relaxation by compromising both its optimality and feasibility guarantees [80, 81].

Our approach. Problem 4 has two types of constraints - stitching (assignment) and capacity constraints. The assignment constraints (Equation 5.14) represent flow conservation constraints for metalinks $tj \in E_M^a$. Hence, these constraints form a totally unimodular constraint matrix. When having the linear objective function (Equation 5.13), this property allows us to relax integrality constraints on $f_{ijk}^{st}(b,a)$ variable in the uncapacitated service chain composition case (when capacity constraints are omitted). As a result, we can solve the above problem using polynomial Linear Programming (LP).

Lower Bound Algorithm. Similarly to [81], we use the unimodularity property benefits and push capacity constraints (see Equations 5.10 and 5.11) to the objective. To this end, let us denote $g_1^{\tau j} = R - \mathbb{P}_j^\tau$ and $g_2^{uv} = R - \mathbb{P}_{uv}$ functions for each constraint in Equations 5.15 and 5.16, respectively. Let us define $u_1^{\tau j}$ and u_2^{uv} as the Lagrangian multipliers specified for each iteration of the subgradient method [81]; we now can define (deterministic) Lagrangian weights as following:

$$w_{ijk}^{sta} = F_{ijk}^{sta} + u_1^{\tau j}\left(\frac{\mu_s^{a\tau} + K_{\frac{R}{P_i}}\sigma_s^{a\tau}}{deg(s)} + \frac{\mu_t^{a\tau} + K_{\frac{R}{P_j}}\sigma_t^{a\tau}}{deg(t)}\right) + \sum_{uv \in P_{ijk}^{sta}} u_2^{uv}\left(\mu_{st}^a + K_{\frac{R}{P_{uv}}}\sigma_{st}^a\right) \quad (5.17)$$

We then can solve the following linear program $\mathcal{L}$ with any LP solver:

$$\mathcal{L} = \min \Bigg(\sum_{a \in A} \sum_{b \in B^a} \sum_{st \in E_V^a} \sum_{P_{ijk}^{sta} \in \mathcal{P}_a^{st}} w_{ijk}^{sta} f_{ijk}^{st}(b,a) - \\ - \sum_{j \in N_S} \sum_{\tau \in T} u_1^{\tau j} \cdot \begin{cases} C_j^{\tau}, R \leq P_j \\ 0, \quad R > P_j \end{cases} - \sum_{uv \in E_s} u_2^{uv} \cdot \begin{cases} C_{uv}, R \leq P_{uv} \\ 0, \quad R > P_{uv} \end{cases} \Bigg) \tag{5.18}$$

subject to constraints in Equation 5.14. Note that to improve LB while solving $\mathcal{L}$, we can also fix all variables $f_{ijk}^{sta} = 0$ whose node (or link) mappings do not satisfy reliability, i.e., if $R > P_i$ or $R > P_j$ (or if $\exists uv \in P_{ijk}^{sta} : R > P_{uv}$).

If solution of $\mathcal{L}$ satisfies GAP capacity constraints, we can stop and report optimal (or suboptimal) solution to GAP. However, if $\mathcal{L}$ solution is unfeasible to the primal GAP problem, we can project it back to the feasible space using some polynomial heuristic algorithm to get an upper bound (UB) of the primal GAP problem.

Upper Bound Algorithm. In this thesis, we propose a new (polynomial) greedy regret lower bound replication (GRLBR) algorithm that we found fast enough for our large scale GAP problem with flow assignment constraints. We build our GRLBR algorithm upon both lower bound replication and greedy regret algorithms proposed earlier in [81], and its pseudo code is outlined in Algorithm 7.

GRLBR starts by detecting the largest regret service chain segment st of service chain a' (lines 5-12), i.e., the segment with the largest difference between the first best and the second best corresponding lagrangian weights w_{ijk}^{sta} for its potential feasible assignments. If there are no feasible metapaths assignments for st of a that satisfy both assignment and capacity constraints (see Equations 5.14, 5.15 and 5.16), we stop and report no feasible solution (lines 6-8). Once, st of a' is found, we add it to the priority queue $Q_{a'}$ based on its langrangian weight $w_{ijk'}^{sta'}$ (line 13). We then retrieve and remove the head of this queue and try to map it to the LB metapath solution first (lines 19-20), or to the lowest lagrangian

Algorithm 7: GRLBR

Input: $\hat{f}^{st}_{ijk}(a,b)$:= solution of $\mathcal{L}_2$; w^{sta}_{ijk}:= lagrangian weights; $P^{sta}_{ijk} \in \mathcal{P}^{st}_a$:= set of k-constrained shortest metapaths and their corresponding fitness values F^{sta}_{ijk} found for each virtual link $st \in E^a_V$

Output: UB := upper bound to GAP problem; $f^{st}_{ijk}(a,b)$:= feasible solution to GAP problem

1 **begin**

/* **Step 0: initialize** */

2 $A' \leftarrow A$

3 $B'^a \leftarrow B^a, \forall a \in A$

4 **while** $A' \notin \emptyset$ **do**

/* **Step 1: find highest regret virtual link** sta' */

5 **forall** $st \in E^a$ and $a \in A'$ **do**

6 **if** $\nexists P^{sta}_{ijk} : P^{sta}_{ijk}$ is feasible **then**

7 terminate and report no feasible solution

8 **end**

9 $ijk'_{sta} \leftarrow arg\min\{w^{sta}_{ijk} : P^{sta}_{ijk}$ is feasible$\}$

10 $\rho_{sta} \leftarrow \min\{w^{sta}_{ijk} - w^{sta}_{ijk'(sta)} : P^{sta}_{ijk}$ is feasible, $ijk \neq ijk'_{sta}\}$

11 **end**

12 $sta' \leftarrow \underset{st \in E^a, a \in A'}{arg\max} \{\rho_{sta}\}$

/* **Step 2: allocate all service chain segments that contains** sta' */

13 Put sta' to the priority queue $Q_{a'} \leftarrow \{sta', w^{sta'}_{ijk'}\}$

14 $b' \leftarrow \min\{B'^{a'}\}$

15 **while** $Q_{a'} \notin \emptyset$ **do**

16 $\hat{st} \leftarrow$ retrieve and remove $Q_{a'}$'s head

17 **if** $\nexists P^{\hat{sta}'}_{ijk} : P^{\hat{sta}'}_{ijk}$ is feasible **then**

18 terminate and report no feasible solution

19 **else if** $P^{\hat{sta}'}_{ijk} : \hat{f}^{\hat{st}}_{ijk}(a',b') == 1$ is feasible **then**

20 $UB \leftarrow UB + F^{\hat{sta}'}_{ijk}$

21 **else**

22 $P\hat{sta}'_{ijk'} \leftarrow arg\min\{w^{\hat{sta}'}_{ijk} : P^{\hat{sta}'}_{ijk}$ is feasible$\}$

23 $UB \leftarrow UB + F^{\hat{sta}'}_{ijk'}$

24 **end**

25 allocate corresponding physical resources for $\hat{st}$

26 add adjacent virtual links of $\hat{st}$ and their best lagrangian weights to $Q_{a'}$

27 **end**

/* **Step 3: mark** b' **backup of** a' **service chain as allocated and go to Step 1** */

28 $B'^{a'} \leftarrow B^{a'} - b'$

29 **if** $B'^{a'} \in \emptyset$ **then**

30 $A' \leftarrow A' - a'$

31 **end**

32 **end**

33 **end**

weight metapath $P\hat{sta}'_{ijk'}$ (lines 22-23), or report no feasible solution and terminate, otherwise (lines 17-18). Finally, we allocate corresponding metapath solution resources for the service chain st segment of a' and add all its adjacent segments (lines 25-26). Once $Q_{a'}$ is empty, all service chain segments of service chain a' for its backup b' have been placed. We then mark b' backup of service chain a' as mapped and remove it from further consideration by GRLBR (lines 28-31). Note that at any time $Q_{a'}$ contains only two elements due to a linear service chain topology.

Subgradient method. Having LB and UB algorithms outlined, we use them within the general subgradient method to iteratively improve LB and UB as in [81]. To this end we start with zero u_1 and u_2 lagrangian multiplier vectors. At each iteration we track if LB solution is feasible, and if so we terminate our subgradient algorithm. Moreover, if LB has been improved, i.e., if $LB_{new} > LB$, and LB is not feasible, we project LB solution back to the feasible space with our GRLBR algorithm to obtain new UB_{new} solution and update existing UB solution if $UB_{new} < UB$. If $\frac{||UB-LB||}{||LB||} < \epsilon$ or number of iterations is exceeded, we terminate the subgradient algorithm. At the end of each iteration u_1 and u_2 are calculated w.r.t. to their objective gradient. More implementation details as well as best practices on the subgradient method can be found in [81].

5.4 Service Chain Maintenance via Metapaths

Any network operating in a challenged scenario is subject to instabilities. Consider e.g., a physical network after a natural disaster. In this section, we present a novel *metapath-based service chain maintenance* algorithm that utilizes a distributed control plane to cope with such network instabilities by allowing migration i.e., reallocation of (part of) the service chain to maintain its services. We first discuss challenges of guaranteeing distributed control plane consistency. We then outline our service chain maintenance algorithm, and then prove its eventual correctness property to qualify for the Simple Coordination Layer use that avoids expensive consensus protocols [71].

5.4.1 On Distributed Control Plane

To avoid both congestion in the centralized control plane and its single point of failure, we aim to design and develop a distributed service chain maintenance approach which is based on distributed control plane system. Today distributed control plane systems are usually uti-

lized for SDN frameworks like ONIX [139], ONOS [70], etc. It's known that such systems require consistency guarantees to avoid various violations [71]. Examples of such violations tailored to the service chain composition and maintenance can be double assignment of services, looping paths, QoS constraints violations, etc. A common approach to guarantee consistency of a distributed control plane is by establishing a consensus. The latter can be done by running known consensus algorithms (e.g., Paxos [140], Raft [72], etc.). To the best of our knowledge, the only distributed service chain composition and maintenance algorithm that builds upon consensus literature is 'Catena' [73, 74]. This algorithm ensures a consistency by running consensus algorithms based on the policy specified and can be used safely within distributed control plane. Catena runs expensive consensus algorithms for every service in a service chain request.

The recent work in [71] proves however that complex consensus algorithms are unnecessary to guarantee consistency if the distributed control plane qualifies for the "eventual correctness" property. In order to qualify, control mechanisms needs to be *deterministic*, have *idempotent behavior*, *triggered recomputation* and be *proactive* w.r.t., to data plane. To this end, the authors in [71] introduce their Simple Coordination Layer that can be used when control mechanisms qualify for the *eventual correctness property*.

5.4.2 Metapath-based Service Chain Maintenance

Firstly, we assume that all demand increase requests of service chains which QoS demands are not sufficient are handled proactively by the *root controller* – a controller associated with a physical node of the chain root service. By convention the root service is a service that outputs service chain processed data. We now present our metapath-based service chain maintenance algorithm for a distributed control plane which logic is outlined in Algorithm 8, and Figure 5.3 illustrates its work. Upon a non-functional service chain segment detection (Figure 5.3a), the algorithm starts from the *root* controller and then, recursively, checks all service chain segments between the corresponding controllers to find all non-

functional assignments (Step 1). In Step 2, the algorithm generates k-constrained shortest metapaths or finds them among the list of pre-computed, e.g., during the service chain composition. When all service chain segments belonging to some controller are checked or temporally restored (Figure 5.3b), this controller requests the next chain segment (Steps 3 and 4; Figure 5.3c). Once the termination criteria is met (Figure 5.3d), the best found mappings or a failed service chain error message are returned (Step 5).

Algorithm 8: Metapath-based Service Chain Maintenance

Step 1: Upon receiving a message from controller l, start from the service s at the controller j and verify the next service chain link segment $st \notin$ checked segments

- *If* st is non-functional, stop and go to Step 2
- *Else if* $t \in j$, make $s = t$ and check next segment
- *Else*, send checked service chain segments to the controller $i : t \in i$

Step 2: If failed st found, generate k-constrained shortest metapaths set K for st (sorted in ascending order by their fitness function values) *dynamically* (to be discussed later) or among pre-computed metapaths during the composition step

Step 3: Iterate while $K \notin \emptyset$

- Retrieve and remove k metapath from K, then temporally allocate st on k and add it to checked segments
- Check current fitness function value:
  - *If* current fitness function values is worse then the best known objective value, skip this step
  - *Else if* $t \in j$, make $s = t$ and resume from the step 1
  - *Else*, send checked service chain segments to the controller $i : t \in i$
- If all segments have been checked and current fitness function value is the best known, track best physical resource mappings for non-functional segments
- Release st resources and remove it from checked segments

Step 4: Reply back to the controller l

Step 5: Once a termination criteria is met, permanently provision best known physical resources for non-functional service chain segments.

The termination criteria is met when all combinations of possible failed segment allocations are checked or some heuristic number is reached, e.g., the number of maximum recursive calls, used metapaths, etc.

Note that as we use k constrained-shortest metapaths at each recursive call, one potential problem that arises is an exponential complexity of our algorithm which can be bounded as $O(k^{E_V})$ for the worst case scenario when all service chain segments have failed. To avoid such complexity intractabilities, we suggest to use a simple decay function for k

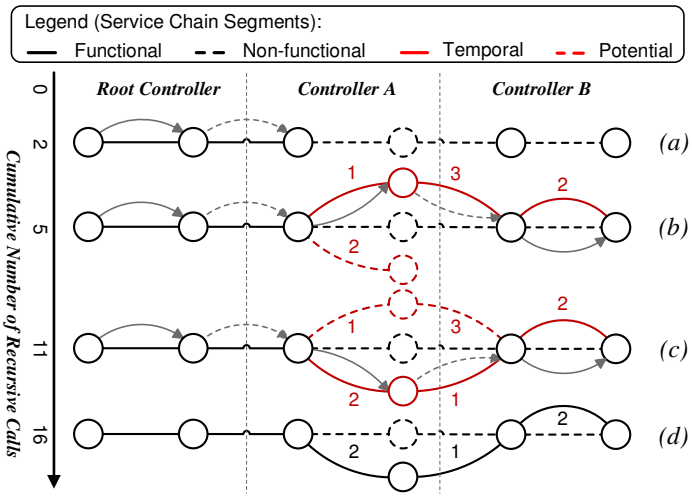


Figure 5.3: Illustrative example of our distributed metapath-based service chain maintenance algorithm: (a) after simple coordination layer detects changes in physical resources, e.g., their failures or congestion, (b) $root$ controller starts its recursive analysis to verify that its related service chain segments are functional before requesting this check from controller A; upon receiving a maintenance requets from the $root$ controller, controller A finds 2 metapaths for its part of failed service chain segments, temporally reserves corresponding physical resources for the best metapath (of cost 1) and requests controller B to continue; (c) controller B is then finds a metapath for the last failed service chain segment and replies back to controller A with the best known mapping so far; finally, once controllers A and B check all potential mappings, the best mapping of the failed service chain segments (with the total fiitness function of 5) is provisioned permanently.

policy, i.e., $k = round(k^{\frac{1}{d}})$, where d is a depth of a service chain segment from the root. Thus, if $k = 10$, we use up to 10 metapaths for the first segment, up to 3 for the second one, etc.

To generate metapaths, we have two options - use metapaths which have been *pre-computed* during the service chain composition step, or find such metapaths *dynamically*. We evaluate different policy trade-offs in Section 5.5. To find such metapaths *dynamically*, we propose our distributed constrained shortest metapath algorithm based on our proposed Neighborhood Method constrained shortest path finder in Chapter 4.2.

Distributed constrained shortest metapath algorithm.

Our distributed constrained shortest metapath algorithm is built upon our previous work on the constrained shortest path finder [53] and has the following two phases:

- **Phase 1**: we start by estimating the best multi-dimensional distances (i.e., the cost c and distances related to $\vec{t}$ constraints) of metapath candidates from the source service. The goal of this phase is to track levels of discovered physical nodes and their corresponding best possible constraint metrics. This phase ends as soon as all possible candidate distances are estimated.

- **Phase 2**: if the destination service virtual node has been discovered during Phase 1, we start a branch-and-bound exhaustive search from the destination to the source service. However, we use only those path candidates whose distances have been discovered during Phase 1 at the corresponding level. The phase ends, when all candidates are estimated, or desired optimality is achieved.

In addition to pruning path candidates (of an exhaustive search) during Phase 2 by the discovered during Phase 1 physical node levels and their path metrics, we use other two powerful candidate pruning techniques, i.e., pruning by *infeasibility* and *path dominance* techniques as described in Section 4.2.2. We remark that pruning by *path dominance* technique cannot be applied in the case of finding k-constrained shortest paths, and only a pruning by k^{th}-bound reduction technique can be used.

Algorithm 9 outlines logic of our constrained shortest metapath algorithm and Figure **??** illustrates its work. Phase 1 is initiated by the source service src controller C_{src} (lines 6-9), and src is located at level 1 with $\vec{0}$ multidimensional distance (line 2). At each successive level, physical nodes' controllers reject Phase 1 message received from node neighbors located at the previous level if their best discovered distances at current level are *infeasible* or message Time To Leave (TTL) is violated (line 22).

Once a physical node that can host the destination service dst is discovered at some level with feasible $distances$, it initiates Phase 2 by sending corresponding messages to its neighbors (line 30). Further, at each preceding level nodes reject Phase 2 messages if these messages have the different discovery level (line 35). For example, consider destination node Y that has been discovered at level 4 through node A discovered at levels 2 and 3.

Algorithm 9: k-constrained shortest metapaths algorithm

Input: id:= of the service chain link; src:= source service; dst := destination service;
C_{src} := src controller; $\vec{t}$:=vector of end-to-end constraints; c:=cost to minimize; k policy
Output: *k-constrained shortest metapaths* between src and dst that satisfy all constraints $\vec{t}$

```
/* each physical node i runs an agent that searches for constrained shortest paths
   when requested                                                                  */
```

initialization:
  if $i == src$ **then** $distances_i^{id} \leftarrow < 1, (0, \vec{0}) >$
  else $distances_i^{id} \leftarrow \emptyset$
  $candidates_i^{id} \leftarrow \emptyset$
end
send (*called by* C_{src} only):
  $m_{id} \leftarrow id$; $m_{type} \leftarrow$ Phase 1; $m_{src} \leftarrow src$; $m_{dst} \leftarrow dst$; $m_{cost} \leftarrow c$; $m_{constr} \leftarrow \vec{t}$; $m_{distances} \leftarrow distances_i^{id}(1)$; $m_{level} \leftarrow 1$;
  send_to_neighbors(src, C_{src})
end
send_to_neighbors (*called by node* i *controller* C_i):

```
/* C_i sends m to all physical neighbors of i as following:                        */
```

  if neighbor $\notin C_i$ **then**
    send m through data plane
  else
    send m through C_i local loop (in memory)
  end
end
receive (*Phase 1 or 2 message* m *from node* j):
node i delivers m to its C_i controller;
$id \leftarrow m_{id}$; $c \leftarrow m_{cost}$; $\vec{t} \leftarrow m_{constr}$;
  if m_{type} is Phase 1 message **then**
    $\vec{dist} \leftarrow m_{distances} + (c_{ji}, \vec{t_{ji}})$
    if $\vec{dist}$ is *feasible* for $\vec{t}$ and $m_{level} + 1 \leq$ Time To Live and $\exists x \in \vec{dist} : dist_x < distances_i^{id}(m_{level}+1, x)$ **then**
      track best distance in $distances_i^{id}(m_{level}+1)$ using $\vec{dist}$
        if i can host m_{dst} service (i.e., has a metalink to m_{dst}) **then**
          $m_{type} \leftarrow$ Phase 2; $m_{distances} \leftarrow (0, \vec{0})$; $m_{path} \leftarrow i$;
        else
          $m_{distances} \leftarrow distances_i^{id}(m_{level}+1)$
          $m_{level} \leftarrow m_{level} + 1$
        end
        send_to_neighbors(i, C_i);
      end
    end
  elseif m_{type} is Phase 2 message and $distances_i^{id}(m_{level}) \notin \emptyset$ **then**
    $src \leftarrow m_{src}$; $dst \leftarrow m_{dst}$; $\vec{dist} \leftarrow m_{distances} + (c_{ji}, \vec{t_{ji}})$;
    if $(\vec{dist} + distances_i^{id}(m_{level}))$ *feasible* for $\vec{t}$ and *dominant* **then**
      add *dominant* path $(i \cup m_{path})$ to $candidates_i^{id}(dst)$
      if i can host src (i.e., has a metalink to src)) then
        C_i adds constrained shortest metapaths from $candidates_i^{id}(dst)$
      else
        $m_{path} \leftarrow i \cup m_{path}$; $m_{level} \leftarrow m_{level} - 1$;
        send_to_neighbors(i, C_i);
      end
    end
  end
end

Thus, if node B receives the Phase 2 message from node Y with level $4 - 1 = 3$ it has to reject this message as it was discovered only at level 2 (see Figure 5.4-right), whereas node A can process this message. The same holds for Phase 2 messages with level 1 and

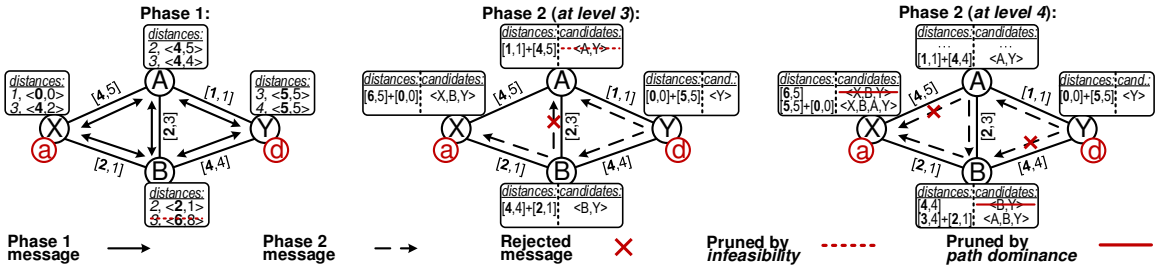


Figure 5.4: Illustrative example of a and d services chaining (hosted at X and Y, respectively) using our constrained shortest metapath algorithm with ≤ 5 end-to-end latency constraint and [**cost**, latency] physical link metrics: we start by sending Phase 1 message from the source node X located at level 1 (left), and at each successive level nodes send Phase 1 message to their neighbors at the next level if and only if nodes' best discovered distances at current level are *feasible*; assuming that destination node Y is first discovered at level 3 with feasible latency, Y sends Phase 2 message to its neighbors (center), and at each preceding level nodes further send Phase 2 message to their neighbors at the previous level if and only if they can be part of the *optimal* path, i.e., path candidates that span these nodes are *feasible* and *dominant*; Phase 2 ends when *optimal* or constrained shortest path, e.g., X, B, A, Y with cost and end-to-end latency equal to 5, is found (in our case path is found from the destination node Y discovered at level 4). Note how all nodes which level do not match with the message level reject it during Phase 2 (e.g., X and B reject messages from A and Y during Phase 2 started at level 4, respectively). Also note how a path candidate of node A during Phase 2 (started at level 3) is pruned due to infeasibility, i.e., its sum of the current distance [**1**,1] and its best discovered distance at level 2 [**4**,5] violates latency constraint ≤ 5, whereas path candidates of nodes X and B (that were added during Phase 2 started at level 3) are pruned by *path dominance* of candidates discovered during Phase 2 started at level 4 (right).

2 that have be rejected by nodes A (see Figure 5.4-center) and X (see Figure 5.4-right), respectively. All these are examples of pruning by *double pass* technique that quadratically reduces time complexity upper bound of an exhaustive search [53].

Moreover, nodes can add themselves to path candidates (stored in Phase 2 messages) and send these messages to their neighbors (lines 40-41), if and only if they can be part of the metapath, i.e., candidates are not pruned neither by *infeasibility* nor by *path dominance* if we want to find the constrained shortest metapath (line 35). An example of pruning by *infeasibility* technique can be observed in Figure 5.4-center. Node A prunes path candidate A, Y as its sum of the current distance [**1**,1] and its best discovered distance at level 2 [**4**,5] violates latency constraint $1 + 5 > 5$. At the same time, an example of pruning by *path dominance* can be seen in Figure 5.4-right. Note how nodes X and B prunes path candidates X, B, Y and B, Y that can be part of the optimal solution as better alternatives (i.e., X, B, A, Y and B, A, Y, respectively) have been found. Phase 2 ends when C_{src} finds k-constrained shortest metapaths (lines 37-38).

5.4.3 On Eventual Correctness of Metapath-based Service Chain Maintenance

Based on the recent results in [71] it is possible to guarantee a distributed control plane consistency without expensive consensus protocols by introducing a simple layer for its coordination. However, in order to qualify for this layer use, control mechanisms have to have *eventual correctness* guarantees. We formally prove such guarantees of our Algorithm 8 below.

Lemma 5.4.1. (The metapath-based service chain maintenance eventual correctness) *Our metapath-based service chain maintenance outlined in Algorithm 8 has eventual correctness guarantees.*

Proof. To have eventual correctness guarantees, by definition control mechanisms needs to be *deterministic*, have *idempotent behavior*, *triggered recomputation* and be *proactive* w.r.t., to data plane [71]. We remark that once service chains are functional, their demands become certain during some time slot τ. Hence, our metapath-based service chain maintenance outlined in Algorithm 8 is *deterministic* and features the *idempotent behavior*, e.g., once a flow rule for some service chain link is setup by one controller this rule remains the same if setup by another one. Moreover, our Algorithm 8 needs to be *triggered* (e.g., by a *root controller*) for a service chain elements migration which is done *proactively* by setting up new flow rules and provisioning new virtual machines to host service functions. ■

We conclude, that Algorithm 8 qualifies for the simple coordination layer use. In the rest of this section, we evaluate performance of our reliable service chain orchestration approach using trace-driven simulations.

5.5 Performance Evaluation

In this section, we evaluate performance of our reliable service chain orchestration approach under challenging disaster incident conditions that can cause severe infrastructure outages [3] in two complementary scenarios: we first evaluate its performance against the state-of-the-art NFV/VNE solutions of the (master) integer multi-commodity-chain flow problem;we then evaluate its maintenance performance w.r.t., the only existing to our knowledge consensus-based service chain orchestration scheme [73, 74].

(a) (b)

Figure 5.5: Simulation data sets: (a) network infrastructure that spans 7 Tier-1 US providers and comprises of 286 Point of Presence (PoP) nodes and 534 links; and (b) network infrastructure that spans 56 regional US providers and comprises of 596 PoP nodes and 1253 links. Both network infrastructures are obtained from the Internet Topology Zoo [1] and Atlas [2] data sets.

General Settings. For our simulations, we use an *HPC Cloud server* with two *Intel Xeon E5-2683 v3* 14-core CPUs at 2.00 GHz (total 56 virtual cores), 256GB RAM, and running the Ubuntu 16.04 allocated in NSF CloudLab platform [141]. We solve math programs with IBM ILOG CPLEX [134]. We use both Internet Topology Zoo [1] and Atlas [2] databases to re-create the US Tier1 and regional providers' networks as shown in Figure 5.5. We assume that each topology has nodes and links with uniformly distributed computation capacity from 5 to 50 TFlops and bandwidth from 1 to 10 Gbps, respectively. Note that the lowerbound 5 TFlops performance can simulate limited network edge servers, whereas 50 TFlops can simulate HPC cloud servers. Moreover, we assume that latency of each physical link is proportional to its propagation delay computed as its geographical length divided by the speed of light in fiber. Finally, we compute physical resources' outage risk w.r.t. to the geographical proximity to the disaster incident epicenter as discussed in [3].

All our results show 95% confidence intervals, and our randomness lays both in service chain requests and in disaster incident events.

Service chain request settings. We generate a pool of 50 service chains composed by 2 to 20 services, unless stated differently. Based on a common object tracking application [16], each service chain has equal chances to express its either High-Performance (HPC) or regular computing demands shown in Table 5.1. As in [3], we assume a strong correlation between service chain demands and the disaster incident intensity. Using natural disaster data sets and their associated infrastructure outage risks specified in [3], we use the following *geo-location policies*: All services handling incoming raw data must be placed within a range of two disaster incident region radiuses from its epicenter. When there are no disaster incidents, these services as well as services that output processing data have to be placed within 200 miles out of the random geographic locations picked within the US.

Results. Our evaluation results can be summarized with the following thrusts: (i) *our metapath approach yields more than 99% optimality on average and is up to 3 orders of magnitude faster than the master problem solution*; (ii) *our metapath approach can secure up to two times more service chains in comparison to the state-of-the-art NFV/VNE approaches under challenging disaster incident conditions*; (iii) *policies allow to trade-off between a service chain reliability and its composition optimality*; and (iv) *our metapath approach enables better service chain maintenance for lesser control messages*.

5.5.1 Service Chain Composition Evaluation

Composition Metrics. We compare performance of our metapath-based service chain composition in Problem 4 (referred as $MpSC$) against its (polynomial) Lagrangian relaxation counterpart (referred as $MpLG$). We also compare $MpSC$ against the VNE/NFV state-of-the-art solutions of the (master) integer multi-commodity-chain flow problem (i.e., Problem 3): IBM CPLEX branch-and-bound version [134] (optimal, but has the highest combinatorial complexity), branch-and-price column generation [61] and recent isomor-

phism detection [82] approaches (suboptimal, but have lower combinatorial complexities). We refer to the branch-and-bound solution of the master problem as Opt, to the column (or path in case of service chains) generation approach as $PgSC$, and to the isomorphism detection as $IsoSC$.

Assumptions. To evaluate the *online* optimization performance of our approach, we assume the most difficult case: all service chain requests arrive sequentially (i.e., unknown in advance) and do not allow a service consolidation, i.e., only one service in a chain can be placed onto the same physical server.

We assess the performance of our service chain composition algorithms by specifying the fraction of successfully composed service chains over the total requested chains (*composition ratio*). Similarly, we assess the reliability of these algorithms by computing a fraction of the number of composed service chains disrupted during disaster incidents over the total number of composed service chains (*disruption ratio*). In addition, we also use an *optimality gap* metric which we define as a gap in % between Opt and all other algorithms. Finally, we use a *composition time* metric to access *scalability* performance of our metapath approach.

(i) MpSC gains more than 99% optimality on average and is up to 3 orders of magnitude faster than Opt. To estimate the baseline performance of our metapath approach, we first assume a scenario without disasters (i.e., no outage risks and $R = 0$) where capacity chance constraints becomes deterministic yielding the largest feasible space (i.e., leading to a higher combinatorial complexity of the master problem). Figure 5.6a shows how its optimality depends on the number of generated metapaths per single service chain (SC) request. We can see how, when this number exceeds $\geq$80 times the number of physical nodes (PNs), the performance of $MpSC$ flattens. On the other hand, when either the service number (SN) in a chain increases or the reliability requirements get tighter in a disaster incident scene, $MpSC$ achieves optimality most of the time and shows 99% optimality on average. Note also how $MpLG$ shows significantly worse performance with respect to $MpSC$; this

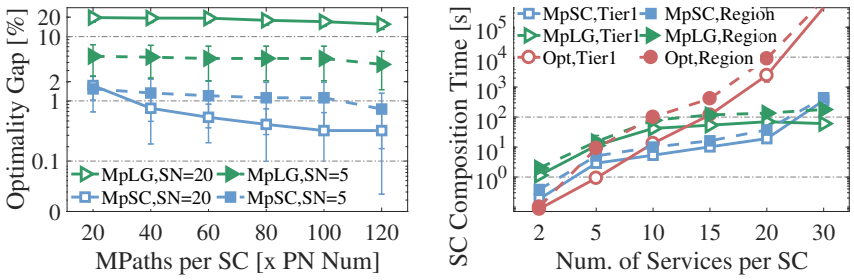


Figure 5.6: Service chain composition optimality gap (a) and time (b) results in presence of no disaster incidents ($R = 0$): our metapath-based service chain composition $MpSC$ reaches ≈99% optimality in average and composes large service chains (of ≥ 20 services) up to 3 orders of magnitude faster than their optimal composition approach Opt. Although Lagrangian relaxation of our metapath-based service chain composition via the subgradient method $MpLG$ shows worse optimization performance, it can be beneficial for large service chains (of ≥ 25 services) as it doesn't use integer programming and is polynomial.

is due to use of greedy heuristics used to recover feasible solutions. However, $MpLG$ can be beneficial for large service chains (of ≥ 25 services) as it is polynomial.

For the rest of our evaluation, we fix the number of metapaths generated per service chain request to 80 and 120 times the number of physical nodes for $MpSC$ and $MpLG$, respectively. These values of metapaths are picked to allow both $MpSC$ and $MpLG$ to compose large service chains. For instance, with these settings $MpSC$ is almost three orders of magnitude faster than the optimal solution (Opt) as shown in Figure 5.6b. For small service chains (i.e., ≤ 5 services) we found, however, no significant scalability improvements of $MpSC$ and $MpLG$ over Opt. This is due to the fact that generating metapaths is time-consuming. Thus, for small service chains, it is recommended to avoid use of metapath-based composite variables and merely consider the Opt policy instead. For the rest of our evaluation, we only use the $MpSC$ service chain composition algorithm.

(ii) MpSC can secure up to 2 times more service chains than PathGen and IsoSC under challenging disaster-incident conditions. Furthermore, we can see how our $MpSC$ outperforms $PgSC$ and $IsoSC$ by securing up to 2 times more service chains under challenging disaster-incident conditions of tornadoes and hurricanes as shown in Figures 5.7a and 5.7b with the service chain reliability $R = 0.8$. This is due to the fact that $MpSC$ reaches the optimality most of the time while being sufficiently scalable. At the same time,

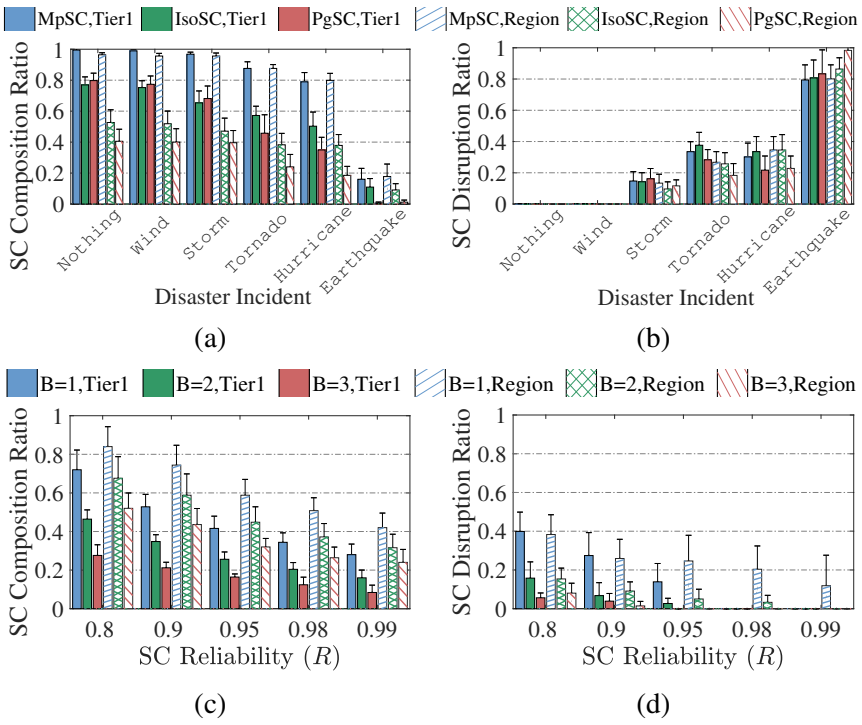


Figure 5.7: Service chain (SC) composition ratio (a,c) and disruption ratio (b,d) results under different natural disaster-incidents with reliability $R = 0.8$ (*first row*), and $MpSC$ results under hurricane disaster-incidents (*second row*).

$PgSC$ is limited by the performance of the service chain composition algorithm (that commonly uses a two-stage composition) to get the initial feasible solution [61]. Moreover, it is also known that column generation approaches such as $PgSC$ converge slowly to the optimal for integer problems [80]. In contrast to $PgSC$, $IsoSC$ doesn't need an initial feasible solution, but can fail to find one or not converge to the optimal solution for the predefined amount of iterations [82].

(iii) Policy-based service chain reliability trade-offs. Further, to achieve a desired level of reliability during service chain composition (i.e., proactively), the capacity chance-constraints acceptable risk (i.e., $1 - R$) and/or the number of backups policies can be adjusted appropriately. As shown for $MpSC$ in Figures 5.7c and 5.7d, increasing either chance-constraints reliability R or the number of backups decreases the number of composed service chains by either prohibiting more physical resources for allocation or utilizing more physical resources for service chain backups. On the other hand, such a strategy can minimize the number of disrupted service chains, therefore minimizing their outages.

5.5.2 Service Chain Maintenance Evaluation

Maintenance Metrics. We compare our metapath-based service chain maintenance algorithm referred as $MpSM$ with the only existing (to the best of our knowledge) consensus-based service chain orchestration approach that can guarantee distributed control plane consistency – $Catena$ [73]. In this simulation scenario we have mainly assessed performance of service chain composition algorithm by specifying the fraction of times service chain events are successfully migrated over their total appearance number (*blocking ratio*). In addition, we also use an *optimality gap* and a *number of control messages* metrics to access the optimality and a complexity performance of our proposed solution.

(iv) Metapaths and simple coordination layer for better service chain migrations with lesser control messages. Figures 5.8a, 5.8b and 5.8c show how $MpSM$ with the pre-computed metapaths P policy can more optimally migrate service chains with a lower blocking probability than $Catena$ and using an order of magnitude less control messages. The reason for these results is two fold. First of all, our algorithm simultaneously considers fitness functions of service placements and their chaining by recursively traversing possible migrations w.r.t. k-constrained shortest metapaths policy, whereas $Catena$ has approximation guarantees only for the service placement and uses k-shortest physical paths (not metapaths) to chain them in a best-effort manner. Secondly, our $MpSM$ uses simple coordination layer to avoid expensive consensus control messages that $Catena$ uses for providing the same distributed control plane consistency guarantees. We can also see how our $MpSM$ reaches optimality of $\geq$90% when number of traversed metapath candidates $k \geq 5$ for both dynamic D and pre-computed P metapath policies. Thus, for the rest of our simulation we fix $k = 5$.

Figures 5.8e and 5.8f illustrate how $MpSM$ can significantly reduce the blocking probability of various service chain migration events when finding metapaths dynamically D. However, the main side effect is that $MpSM$ with D policy demonstrates ≈ 4 orders of magnitude increase in control messages w.r.t. its P policy. Thus, we recommend use D

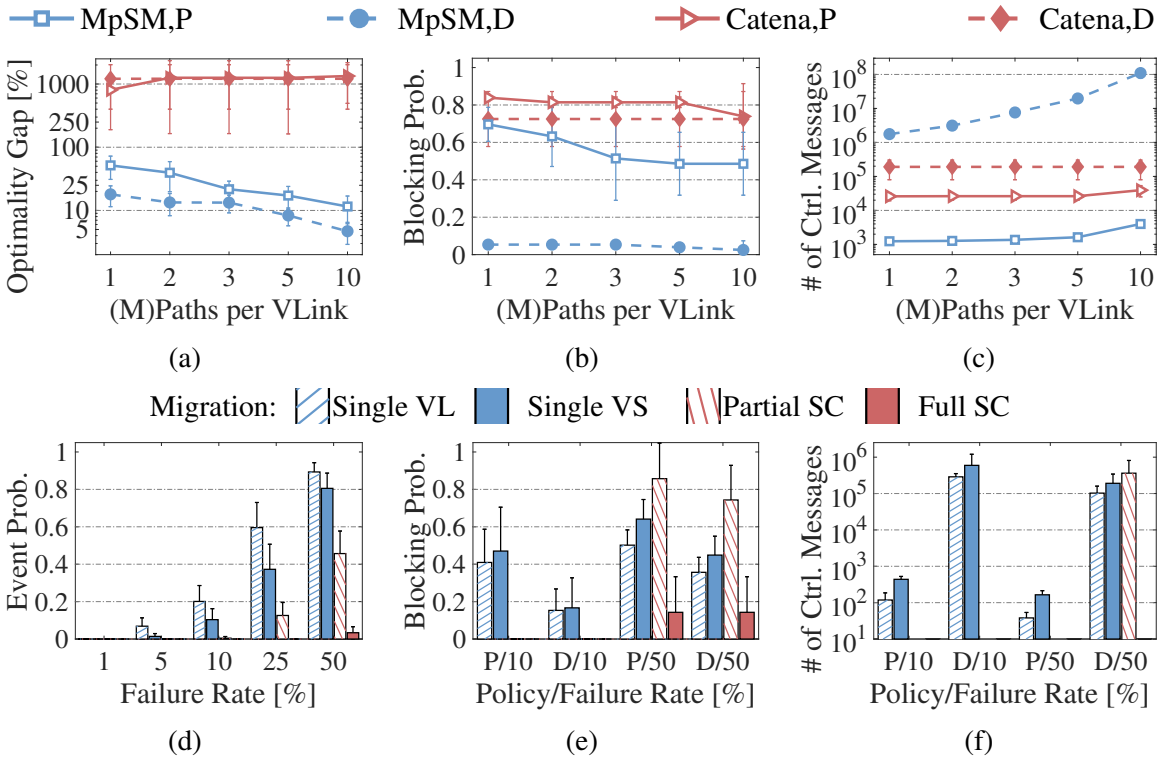


Figure 5.8: Service chain (SC) migration optimality gap (a), blocking probability (b) and number of control messages used for this migration (c) results. Single virtual link (VL), single virtual function service (VS) and partial or full service chain (SC) event migrations probabilities (d), their blocking probabilities (e) and number of control messages used to migrate them (f) results with $k = 5$ metapath policy and different physical network failure rates.

policy if the following criteria are met: (i) single link or service migration events happen; (ii) the physical infrastructure experience severe failures, i.e., $\geq 25\%$; and (iii) number of controllers is at least an order of magnitude less than the number of physical resources. Each of these criteria can decrease the number of control messages approximately by an order of magnitude. Particularly, (i) is due to the fact that single service chain segments are easier to recover, and these events are more common (see Figure 5.8d); (ii) is due to the fact that having more failed physical resources needs less number of control messages and significantly reduces a feasible space for SC migrations; and (iii) is due to the fact that the more physical resources are controlled by a single controller the more recursive calls of $MpSM$ can be done in memory, thus saving on control messages. As a result, number of control messages of $MpSM$ with D policy can be reduced up to 3 orders of magnitude w.r.t. P policy and approximately equal to $Catena$ with P policy.

Chapter 6

Function-Centric Computing Prototype Implementation

In this chapter, we describe several architectural concepts and outline our developed prototypes that can help overcoming data collection, computation and consumption challenges of the function-centric computing paradigm over geo-distributed cloud infrastructures.

6.1 AI-augmented Geographic Routing Architecture

To obtain information about physical obstacles that can be utilized by our edge routing algorithm (see Section 3.3), one can manually label all potential obstacles on the map. However, due to any given disaster-scene scale and due to the fact that time is critical for first responders, labelling physical obstacles on the map all the time (i.e., the manual approach) can be intractable. Thus, we adopt an approach that involves automation of that process.

The artificial intelligence, and more specifically the pattern recognition areas today are ideally suited for such automation processes. The pattern recognition field today includes many approaches related to an object detection (finding the object location and its size)

in the given image (e.g., satellite imagery). These approaches include: Support Vector Machines, Nearest Neighbor, Deep Learning and other techniques. However, the most accurate detectors rely on deep learning approaches [83, 84]. For example, the You Only Look Once (YOLO) [83] deep-learning detector predicts objects in images using only a single neural network composed by 26 layers, an easier task for the edge cloud. At the same time, it can have worse performance than more sophisticated deep learning detectors [84]. Note that alternatives to object detection include also geographical object-based image analysis [142] that relies on the spectral information extracted from image pixels that may require additional and more expensive LiDAR hardware [16] not necessarily available during incident response scenarios.

However, the deep learning object detectors may not find obstacles or misclassify them in some cases. Thus, they still require some human interaction i.e., labeling of some of the detected and correctly classified or misclassified samples. To cope with deep learning complexity, in our "Panacea's Cloud" architecture shown in Figure 6.1, we move deep learning to the core cloud due to limited edge cloud storage and computation resources. Further, we assume that training samples are collected and partly labeled during the previous incident responses at the edge cloud. These samples are then used for supervised or semi-supervised deep learning [143] to enhance performance of the detector in future. The up-to-date detector version

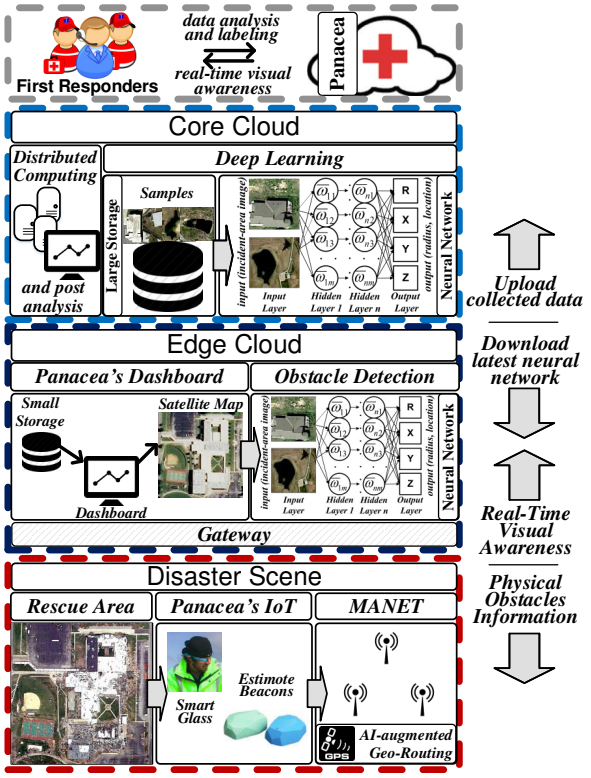


Figure 6.1: To cope the deep learning complexity of the obstacle detector we are moving deep learning to the core cloud. The up-to-date detector version then can be pre-uploaded to the edge cloud and used *off-line* during disaster-incident response activities within a lost infrastructure region to enhance geographic routing.

then can be pre-uploaded to the edge cloud and used *off-line* during disaster-incident response activities within a lost infrastructure region. Once detected, physical obstacles are then propagated to the MANET through a gateway.

6.2 Controller Prototype of Neighborhoods Method

In this section, we establish the practicality of our Neighborhood Method approach for network virtualization described in Section 4.2 with a prototype implementation over a Software Defined Networking infrastructure. Our source code is publicly available at [10]. In particular, our prototype implementation extends the Floodlight OpenFlow controller [9]. Our system architecture is shown in Figure 6.3. Our prototype includes four main logical components: a *physical graph discovery* service, a *path mapping* service, a *path allocation* service and a user *web interface* that interacts with the controller. In the rest of the section we describe with some more details each of the four components of our prototype.

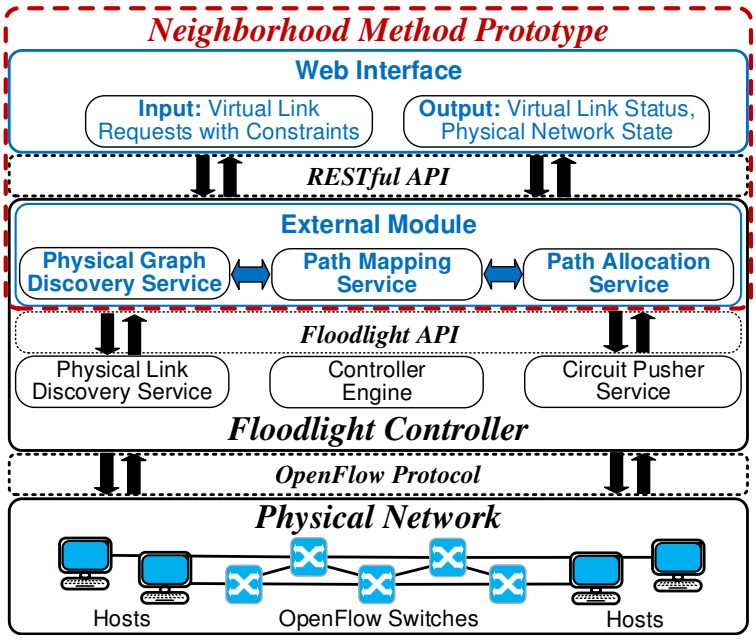


Figure 6.2: System architecture of our NM prototype (which is a module of the Floodlight OpenFlow controller [9]) includes four main logical components: a *physical graph discovery* service, a *path mapping* service, a *path allocation* service and a user *web interface* that interacts with the controller. The prototype source code is publicly available under a GNU license at [10].

Physical graph discovery. Upon bootstrapping a SDN setup, all configured OpenFlow switches connect to the controller allowing a dynamic *switch-to-switch* link discovery. After this phase, the NM module tracks all unknown incoming packets to detect hosts and their corresponding *host-to-switch* links. We specify the main physical link properties, such as capacity and cost (e.g., delay) through an XML configuration file. The XML configuration file indirection allows our NM prototype to easily interact with foreign measurement services, for example for real-time awareness of its link properties. The information collected by the *path discovery* service is then used in the *path mapping* and the path *allocation* steps.

Path mapping. To map constrained virtual link requests, the path mapping module of our prototype uses information from the physical path discovery service and runs one of the following routing algorithm policies: NM for l and $l \oplus 1$ cases (default policy), NM for $l \oplus p$ cases, EDijkstra, IBF or EBFS. In the current prototype version the routing policy is static and is set via our XML configuration file before starting the Floodlight controller.

Path allocation. In this last phase of the path embedding, the NM module sets all appropriate flow rules in all switches via the OpenFlow [144] protocol, along with the computed path obtained during the path mapping phase. Specifically, using OpenFlow protocol messages the module assigns active flows to corresponding queues, i.e., applies an *ENQUEUE* action, for guaranteed bandwidth provisioning. In our XML configuration file, users may specify also the type of timeout for each flow i.e., the flow duration time can start from either the previously matched packet or from the first flow instantiation. To estimate the available bandwidth on each physical link, the NM module uses both the capacity information derived from the XML configuration file, and its allocated bandwidth queried from the flow stored in the OpenFlow switch.

Web interface. To request virtual links and/or to monitor physical network states, we have implemented a web user interface, which uses a RESTful API to communicate with our NM prototype module. The user interface uses technologies such as HTML, PHP, AJAX, JavaScript, and CSS.

6.3 Reliable Service Chain Orchestration Prototype

In this section we describe the architecture of our *reliable service chain orchestration* prototype shown in Figure 6.3. Our prototype architecture includes four main logical components: (i) *control applications*, used to compose and maintain service chains; (ii) the Simple Coordination Layer (SCL) and *root controllers*, used to guarantee consistency of the distributed control plane; (iii) A *SDN*-based system with (iv) a *Hypervisor* to allocate mapped physical resources. Our source code is publicly available at [11]. In the rest of the section, we describe with some more details each of the four components of our prototype.

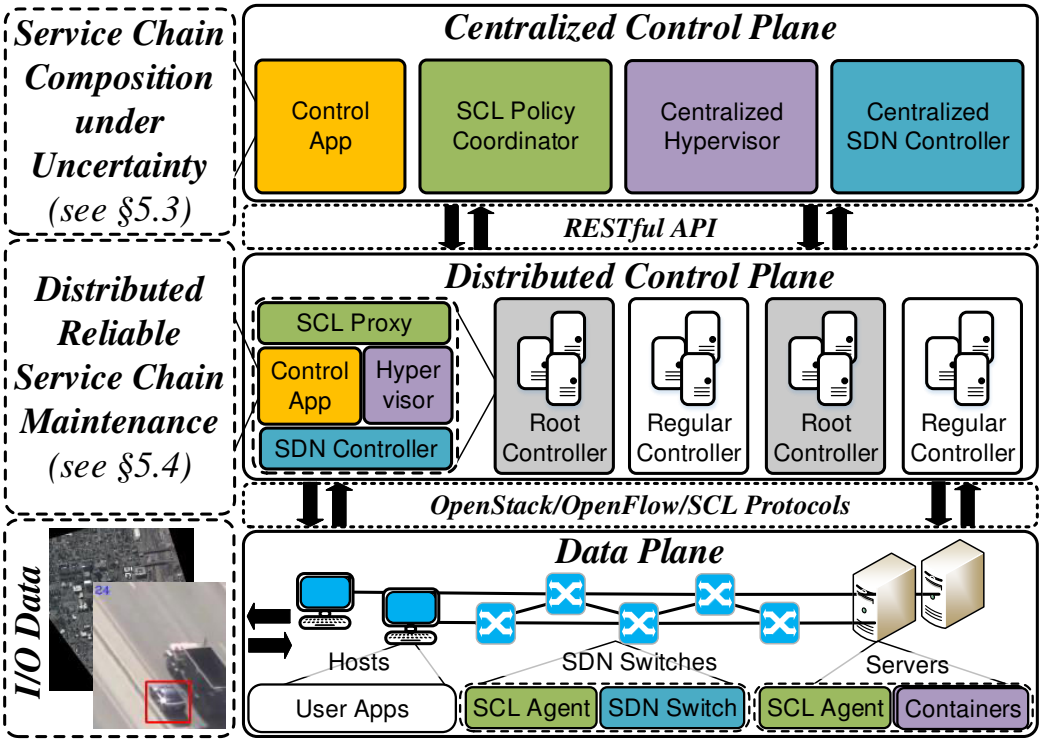


Figure 6.3: System architecture of our reliable service chain orchestration prototype includes four main logical components: (i) *control application* is responsible for a reliable service chain composition in centralized control plane and its maintenance in distributed control plane; (ii) the Simple Coordination Layer (SCL) and *root controllers* are responsible for guaranteeing consistency in the distributed control plane; (iii) *SDN* is responsible for traffic steering in data plane; and (iv) *Hypervisor* is responsible for placing service functions. The prototype source code is publicly available at [11].

Control applications. We have two main types of control applications. The first type is responsible for the reliable service chain composition in the *centralized control plane* as discussed in Section 5.3. The second type is responsible for maintenance of composed service chains as discussed in Section 5.4. We remark that to avoid both a single point of failure as well as congestion in the centralized control plane, we maintain service chains in the *distributed control plane*.

SCL and root controllers. To guarantee consistency in the distributed control plane and avoid various related violations (e.g., looping paths, QoS violations, etc.), our control applications qualify to use the SCL [71]. SCL includes three main components: *SCL Agent* running on physical resources, *SCL Proxy Controller* running on controllers in the distributed control plane, and *SCL Policy Coordinator* running in the centralized control plane. The agent periodically exchanges messages with corresponding proxy controllers and triggers any changes in the physical resources. Proxy controllers send an information about data plane changes to the service chain maintenance control application and periodically talk with other SCL Proxy Controllers. Finally, all policy changes (e.g., in control applications, in SCL, etc.) are committed via 2-phase commit [71] by the policy coordinator.

In order to handle all service chain modification requests from application owners such as demand or latency sensitivity changes, we use *root controllers* - controllers that leverage physical resources associated with root services of service chains. In this thesis, by root services we mean services that provide processed data to end-users.

SDN and Hypervisor. The last two logical components of our prototype are well-known *SDN* and *Hypervisor* systems. Guided by control applications, both SDN and hypervisor are responsible for traffic steering and containers provisioning in the data plane, respectively. We use OpenFlow as our main SDN system [144] and Docker containers [145] as our hypervisor system to place services on the physical server.

6.4 Prototype Evaluation in GENI

6.4.1 Constrained-Shortest Path Management Evaluation

In this final set of results, we use our NM prototype to estimate the impact of the on-demand constrained shortest path computation on the end-to-end virtual link (VL) embedding performance. We also confirm our main simulation results.

Experiment settings. Our setup for the performance experiments includes 15 virtual machines (VMs) from the GENI testbed [85]: Ten of these VMs are OpenFlow Virtual Switches (OVS) [146], and others are hosts. Each *host-to-switch* physical link has 10 Mbps bandwidth and a 0 arbitrary cost, and each *switch-to-switch* physical link has both bandwidth (measured in Mbps) and an arbitrary cost uniformly distributed between 1 and 10. Note, that our arbitrary cost is an additive metric and therefore can represent any path metric, e.g., delay, losses, jitter, etc. We request virtual links with low SLO constraints, i.e., ≥ 1 Mbps bandwidth and ≤ 50 arbitrary cost (5 times greater than the maximum physical link cost), between 5 random $< src, dst >$ pairs of hosts, where for each pair of endpoints we allocate as many virtual links as possible.

Experiment metrics. For each virtual link request we again measure the total gained throughput and the virtual link path hop count. In addition, we measure the time required to compute a path (virtual link mapping), and the time to allocate the computed path (i.e., set appropriate flow rules within OpenFlow switches along the computed path). Note that the overall time for end-to-end virtual link embedding includes both virtual link mapping and its allocation. Our experiment goals are twofold: first, we want confirm our simulation results in real settings; secondly, we want to estimate an overhead of addressing constrained shortest path problem in real settings.

NM gains are confirmed experimentally. Using real-world settings, we were able to confirm constrained shortest path algorithms (i.e., IBF, NM and EBFS) for the online traffic engineering produce superior performance even on a small physical network scale. This is

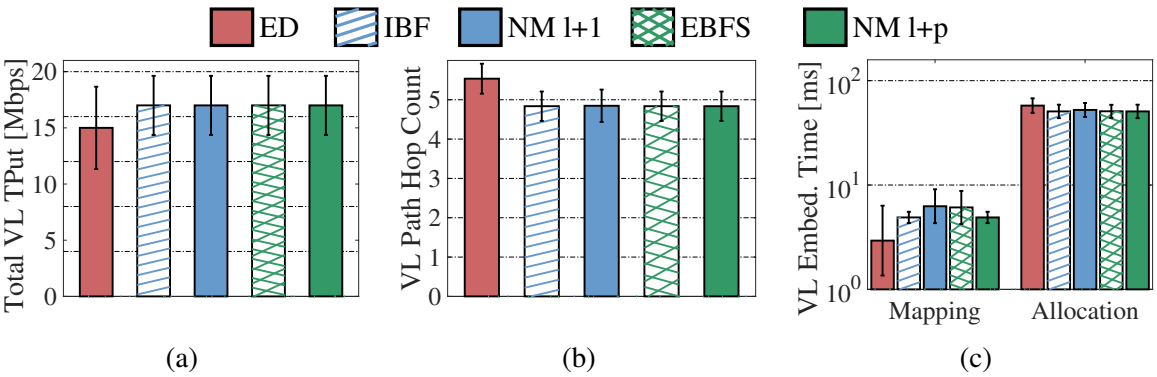


Figure 6.4: Performance analysis of the shortest path algorithm such as the extended version of Dijkstra (ED) versus constrained shortest path algorithms such as NM (in both $l \oplus 1$ and $l \oplus p$ cases), EBFS and IBF on a reserved in GENI small SDN testbed in terms of: (a) total gained throughput; (b) number of path hops per virtual link (VL); and (c) average time per VL embedding, i.e., VL path computation (mapping) and its consequent allocation.

similar to superior results of the offline traffic engineering which utilizes the constrained shortest path algorithms (see Section 4.4.2). Specifically, IBF, NM and EBFS show gains of up to 12% in total VL throughput (network utilization) and find almost 1 hop shorter VL path in average, w.r.t. extended Dijkstra (ED) shortest path scheme as shown in Figures 6.4a and 6.4b. Note that IBF is applicable only l and $l \oplus 1$ cases (see Appendix A.2).

NM running time scales well with physical network size. Figure 6.4c shows how VL mapping is an order of magnitude faster for small scale physical networks (of $\approx 10^1$ nodes) than its allocation for all routing schemes that have been implemented. This is because the path computation is a local (in-memory) operation but the virtual link allocation requires setting up of flow rules within all switches along the loop-free underlying physical path found. Hence, its speed depends on the Round Trip Time between switches and the OpenFlow controller. In reference to Figures 4.10b and 4.10d, we can see how for large scale networks ($\geq 10K$ nodes), the running time of classical constrained shortest path algorithms such as EBFS can become prohibitive (up to two orders of magnitudes larger than in a case of a single path computation for small scale networks). As a result, the VL mapping time becomes a bottleneck. On the contrary, NM is just an order of magnitude slower at large-scale than at small scale. Thus, *NM does not bottleneck the end-to-end virtual link embedding at large scale.*

6.4.2 Case Study Evaluation of Object Tracking Service Chain

In this section, we discuss our edge/core cloud testbed setup in GENI [85] and show improvements in data throughput and tracking time for our case study of object tracking using geo-distributed latency-sensitive service chains w.r.t. the common core cloud computing. To this aim, we use our reliable service chain orchestration prototype implementation outlined in Section 6.3.

(a) (b)

Figure 6.5: Examples of the tracking application results on (a) standard and (b) Full-Motion surveillance video. Each frame in (a) and (b) video datasets is about 2MB compressed.

Setup. Multiple video resolutions in practice need to be processed because the input source imagery in surveillance typically spans a wide variety of sensor technologies found in mobile devices. In our experiments, we consider common resolutions in surveillance video belonging to the broad categories of: (a) Full-resolution WAMI (7800 x 10600) (see Figure 1.6), (b) Large-scale aerial video (2560 x 1900), and (c) Ground surveillance video (640 x 480). To assess *service chaining* benefits for enabling imagery processing at both geo-distributed edge and core cloud sites without large computation demands, we choose the VGA resolution data to track pedestrians [147] in a crowd. The pre-processing step is performed for every image that needs to undergo adaptive contrast enhancement with Imagemagick tool before being used for tracking in the core cloud. The adaptive contrast enhancement requires global image information and thus needs to read in image data into memory, and operates on every pixel. All images are pyramidal tiled TIFF (Tagged Image File Format) and the pre-processing retains the tile geometry.

Our geo-distributed edge/core cloud testbed setup includes 6 virtual machines (VMs) in the GENI platform as shown in Figure 6.6, where three of these VMs emulate OpenFlow switches ($s1$, $s2$ and $s3$) and others are regular hosts ($h1$, $h2$ and $h3$). To consider disaster network scenarios that impact data transfer, we assume a 4G-LTE network configuration at the edge. Hence, each *host-to-switch* link has 100 Mbps bandwidth without delay, each

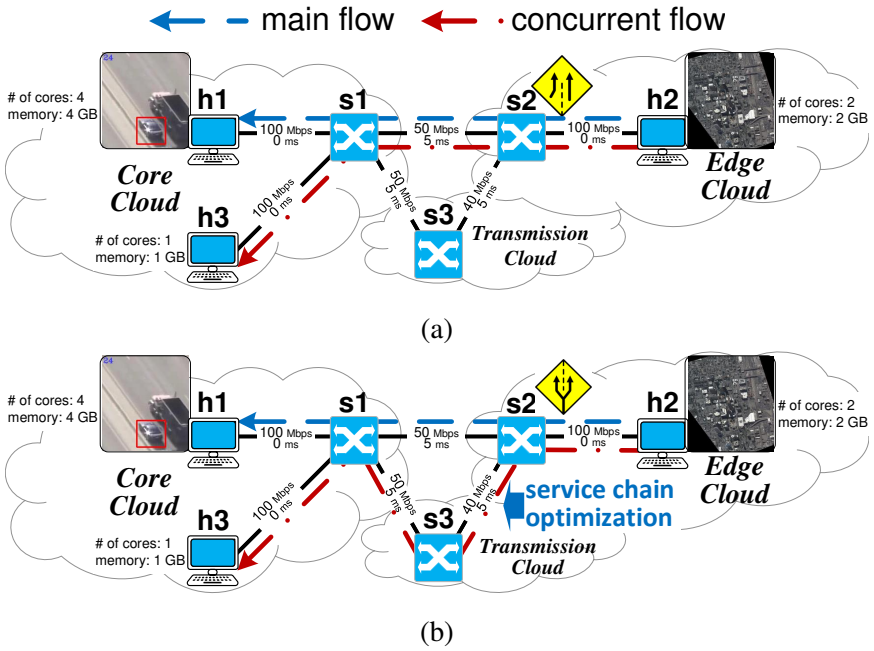


Figure 6.6: Data flows in the allocated in GENI edge/core cloud testbed: (a) object tracking data flow interferes with concurrent flow on the $s2 \rightarrow s1$ link as regular network sends data through the best (the shortest) path; (b) by using our reliable service chain orchestration prototype, we chain tracking services with QoS guarantees which avoids congestion by redirection of concurrent flow through longer path $s2 \rightarrow s3 \rightarrow s1$. Furthermore, by allocating image pre-processing service function at the edge cloud (to $h2$ instead $h1$) it enables near *real-time* tracking.

switch-to-switch link has only 40 or 50 Mbps bandwidth and 5 ms transmission delay to emulate congested and damaged network infrastructure in a disaster scenario. Using our reliable service chain orchestration prototype, the tracking service function is allocated on $h1$ (quad-core CPU and 4GB of RAM) that acts at a *core cloud* site. At the same time, the pre-processing service function is allocated on $h2$ (double-core CPU and 2GB of RAM) that acts at a *edge cloud* site. Finally, $h3$ (single-core CPU and 1GB of RAM) consumes raw data from $h2$ by acting as a remote storage at *core cloud* site. The $h3$ is configured with cross-traffic flows consumption such that it interferes with our object tracking service chain traffic. We call this cross-traffic as the 'concurrent flow', and the object tracking service chain traffic as the 'main flow'.

To differentiate between object tracking via the allocated geo-distributed latency-sensitive service chain and the cloud computing scenario, our experiment workflow is as follows: (i)

Table 6.1: Object tracking case study results.

Performance Metrics	**Object Tracking in the Cloud**	**Object Tracking Service Chain**	**Perceived Benefits**
Preprocess time (s/fr)	0.1955 ± 0.0011	0.202 ± 0.023	**No significant difference**
App thr-put (Mbps)	10.50 ± 0.34	41.85 ± 0.24	Avoiding congestion with our reliable service chain orchestration maximizes object tracking throughput
Tracking time (s/fr)	0.4097 ± 0.0022	0.4229 ± 0.0024	No significant difference
Waiting time (s/fr)	0.902 ± 0.032	$\mathbf{0 \pm 0}$	Achieving maximum speed of tracking service function avoids waiting time and supports real-time data computation
Total time (s/fr)	1.912 ± 0.034	0.4229 ± 0.0024	Real-time data processing via geo/latency-sensitive service chains can produce 4X speedup over common cloud computing scenario

start sending concurrent traffic from $h2$ to $h3$; (ii) start sending main traffic (imagery) from $h2$ to $h1$; (ii.a) while performing data processing via the allocated service chain, start pre-processing concurrently with step (ii) (iii) wait until at least the first frame has been transferred; (iii.b) in case of core cloud computing, start pre-processing before step (iv) (in this case the tracking service function has to wait for each frame when its pre-processing ends); (iv) start tracking; (v) wait until all main traffic has been transferred; and (vi) terminate both the service functions and data transfers.

Results. Table 6.1 shows the final timing results computed with estimate 95% confidence intervals for service chain and cloud computing cases. For each trial, we used a 300 frame video sequence and measured several application performance metrics such as estimated throughout, tracking time, waiting time and total time. In our settings, we are able to pre-process frames faster in the core cloud computing scenario than when a geo-distributed latency-sensitive service chain is used. However, due to congestion in best-effort IP network and the absence of raw data at the *core cloud* site, we cannot track frames in real-time (i.e., with 0 waiting time) in the core cloud computing scenario. Whereas by composing a geo-distributed latency-sensitive service chain of our object tracking pipeline, we can track frames in near real time at $3 - 4$ Hz.[1]

[1]To ease user Quality of Experience assessment, observed computation speedup is also demonstrated on the video - `https://youtu.be/f4-bDvrie_g`.

Chapter 7

Summary and concluding remarks

In this thesis, we have presented a novel function-centric computing approach that addresses data collection (i) and data computation/consumption (ii) problems of geospatial video analytics. Particularly, our function-centric computing approach lays the foundation for interfacing edge computing with core cloud platforms within geo-distributed cloud infrastructures. (i) Specifically, we have addressed the lack of suitable Edge (geographic) routing approaches for IoT-based (e.g., incident-supporting) applications, that can collect their data at constant high-speed data rates to enhance their scalability, reliability and stability. (ii) Furthermore, while solving data computation/consumption problem, we have shown how our reliable orchestration approach can near-optimally compose and maintain service chains in practical time which fosters effective disaster incident response coordination by allowing use of geospatial video analytics to save lives. In the next sections, we first summarize our key findings, we then discuss future work and conclude with a list of function-centric computing open challenges.

7.1 Summary of Contributions

Data Collection and Edge Routing. We have presented a novel AI-augmented geographic routing approach (AGRA), which relies on the physical obstacle information obtained from satellite imagery (available at the edge cloud) by applying deep learning. We then proposed a novel *repulsive* field strategy based on electrostatic potential of Green's function to incorporate physical obstacle knowledge within geographic routing. Our approach theoretically guarantees avoidance of a local minima as well as shortest path approximation. Due to inaccuracies in the obstacles' potential field approximation and the discrete node distribution, in practice the approach cannot guarantee the *local minima avoidance*. To this end, we introduced a novel Attractive Repulsive Greedy Forwarding (ARGF) algorithm which can alternately forward packets in both *repulsive* and *attractive* field modes, to maximize the chances of escaping from, or avoiding local minima. Furthermore, to guarantee packet delivery, we coupled our ARGF algorithm with a known gravity pressure recovery algorithm. As emulating both repulsive and attractive fields allows gradient descent to the destination, the recovery schema can be applied to also minimize the path stretch. Using extensive simulations, we have shown that our proposed algorithms outperform related stateless greedy forwarding solutions in terms of packet delivery success ratio and path stretch. Considering an actual incident-supporting hierarchical cloud deployment scenario, we have also analyzed how ARGF has better goodput performance than other stateless *face routing* solutions (such as, GPSR) as well as stateful reactive mesh routing (i.e., AODV and HWMP).

Data Computation/Consumption and Reliable Service Chain Orchestration. We have shown how our function-centric computing paradigm demands a reliable chain orchestration mechanism, thus extending the concept of Network Function Virtualization. Thus, we presented the *reliable service chain orchestration* approach to ensure reliability of geo/latency-sensitive service chains by handling both their demand fluctuations and possible infrastructure outages during their *composition* (via use of capacity chance-constraints and service backups policies) as well as by *maintaining* these chains throughout their lifes-

pan. Our approach supports specific geospatial video analytics QoS demands such as latency and geolocation and involves addressing the following problems: (a) the NP-hard service chain composition problem, (b) the NP-hard constrained shortest path problem, and (c) the service chain maintenance problem.

(a) We have addressed scalability limitations of the service chain composition problem by proposing a novel *metapath composite variable* approach that uses either (NP-hard) GAP or its (polynomial) Lagrangian relaxation counterpart. Using realistic trace-driven simulations with US Tier-1 and regional infrastructure topologies, we have shown that our metapath composite variable approach reaches 99% optimality on average, is up to 3 orders of magnitude faster than the integer multi-commodity-chain flow problem solution for practically sized problems and can compose twice as many service chain than related NFV/VNE methods.

(b) To enable 'flexible' and 'scalable' metapath composite variable approach (limited by the NP-hardness of the constrained shortest path problem), we have also proposed a novel algorithm viz., 'Neighborhoods Method' (NM). NM utilizes a double pass (a synergy of dynamic programming and a branch-and-bound exhaustive search) and "Look Back" search space reduction techniques. Our computational complexity analysis indicates NM's quadratically lower complexity upper-bound than for recent branch-and-bound exhaustive search methods, and our scalability evaluation results show how NM is faster by an order of magnitude than these methods for networks of $\geq 10,000$ nodes. These results have been also reproduced in a real-world GENI testbed with an NM implementation, whose source code is publicly available under a GNU license at [10].

(c) To avoid a single point of failure, we have proposed the distributed metapath-based service chain maintenance algorithm. To guarantee consistency of the distributed control plane without use of expensive consensus protocols, we have built this distributed plane upon a prior Simple Coordination Layer concept. Both use of metapaths and Simple Coordination Layer allows our solution to reach better optimality for less number of control

messages than a recent consensus-based service chain orchestration scheme - Catena. Finally, we were able to show almost 4x speedup of the data *computation/consumption* in our prototype implementation of an exemplar tracking application case study that uses our reliable service chain orchestration versus common core cloud computing over IP networks. The prototype source code is publicly available under a GNU license at [11].

7.2 Future Work

As part of future work, one can plan to apply our *repulsive*-based Edge (geographic) routing to other known problems in wireless networking. First, the emulated *repulsive* field can be used for traffic engineering by inducing additional charges on heavily-loaded nodes to repulse unbalanced network traffic, and thereby improve the overall network utilization. Second, by inducing additional electrostatic charges in network segments of malicious or selfish behavior, one can improve the overall security. Finally, to improve the overall ad-hoc wireless mesh network vitality, additional electrostatic charges can be induced on nodes with low battery levels. Moreover, our AGRA approach can be also used synergistically with frequency division-based forwarding techniques [92, 93] to further improve routing energy efficiency within IoT devices.

Another interesting avenue to explore is applying our metapath composite variable approach for the service chain composition to other areas of network virtualization area. For instance, one can apply this approach to the popular Virtual Network Embedding problem. To this end, the chaining constraints in the Generalized Assignment Problem should be modified to make sum of all adjacent metapath of the particular metalink si equal to a binary variable $\xi \in \{0, deg(i)\}$. Our Neighborhood method can be also promising for Traffic Engineering problems to satisfy an arbitrary number of constraints while minimizing the cost of a flow routing.

7.3 The road ahead and open problems

We would like to conclude this thesis with a list of open challenges for adopting function-centric computing paradigm in geospatial video analytics. Addressing these challenges is essential for a variety of computer vision applications such as face recognition in crowds, object tracking in aerial wide-area motion imagery, reconnaissance and video surveillance that are relevant for delivering disaster-incident situational awareness.

- *Consideration of Energy-awareness:* To improve overall data collection performance from IoT devices at the wireless edge within lost infrastructure regions, edge routing protocols should not only account for a routing accuracy, but also take into account residual energy levels on routers/IoT devices [21] as well as optimize traffic based on user behavior (e.g., mobility pattern of first responders) to improve the overall edge network utilization as well as satisfy user QoE demands.

- *Consideration of Virtual Topologies of Functions:* In some cases when handling diverse spatiotemporal data sets, computer vision applications may not be easily decomposable to a chain of functions; hence other (more complex) virtual topologies of functions should be addressed in potential cases where benefits of the proposed metapath composite variable approach may not longer hold.

- *Consideration of Processing States:* When computer vision applications for video analytics are not stateless, e.g., tracking objects of interest may require maintenance of an object model, and state information needs to be preserved across functions. This in turn adds an additional layer of complexity that needs to be handled when considering how we can better handle noisy data for processing and compose chains of such functions.

- *Consideration of Hierarchical Control Architectures:* To fully benefit from the function-centric computing paradigm and deliver disaster-incident situational awareness, ex-

isting fog/cloud platforms may need to be extended with a support of hierarchical software-defined control planes to avoid single point of failure/congestion and enable efficient utilization of fog/cloud resources.

- *Consideration of Distributed Intelligence for IoT:* There will undoubtedly be impressive advances in wearables and other smart IoT devices (with higher data resolutions/sizes), 5G wireless networks, and flexible integration as well as management of fog/cloud platforms occur; better modeling, suitable architectures, pertinent optimizations and new machine learning algorithms (particularly deep learning and/or reinforcement learning, real-time recommenders) for geospatial video analytics will need to be considered to adapt these advances for disaster incident-supporting computer vision applications. Solving the above open issues in future explorations is significantly important to advance the knowledge to further the area of geospatial video analytics. Investigations to find solutions to these open issues can lead to a more helpful and accurate disaster-incident situational awareness that can foster effective and efficient disaster relief co-ordination to save lives.

Appendix A

On Constrained Shortest Path for Virtual Network Service Management

A.1 Proof of NM Optimality Theorem

Proof. To prove the NM optimality we need to prove: firstly, that the forward pass of the general NM estimates all possible hop count distances of simple (loop-free) paths between the source and the destination vertices in ascending order; secondly, that the backward pass of NM can return any path of a given length N; and finally that NM returns the *constrained shortest path*.

The first thesis can be proved by contradiction: assume that there is the *optimal path* with (i) the minimum or (ii) the maximum hop count distance to destination, and this distance has not been estimated by NM. Firstly, the forward pass step starts building neighborhoods from the first neighborhood, meaning that NM estimates all possible 1 hop paths first. If there is a path with length lower than 1, we have the special case in which the source node is also the destination assumed to be satisfied, which in turn contradicts with assumption (i). Further, the forward pass step continues to build neighborhoods (estimate possible path hop count distances in ascending order) until their number is equal to the number of

vertices, or the destination vertex appears in the last neighborhood. In the first case, the forward pass ends by estimating the maximum possible distance of a simple (loop-free) solution, and there will never be a case in which a complex path (with loops) can be provided due to a contradiction with the *constrained shortest path* definition (see Definition 1). In the second case, the forward pass ends by finding the hop count distance to the destination. In both cases we contradict assumption (ii).

The second thesis can be also proved by contradiction: assume that NM has not found a path with the N hops length between the source and the destination nodes. This is possible only if at least one of the path's nodes has not appeared in the neighborhoods $< NH >$. In this case, it suggests that this node is not accessible from the source node within N hops, i.e., if it is unreachable, or path does not have the N hops length, which contradicts our assumption.

The third thesis is easy to prove: NM successively checks paths for constraints satisfaction in ascending hop count order, where NM continues to iterate until either one of the path will satisfy all constraint or the solution length will be equal to the number of vertices. In the first case, NM finds the *min hop count* path which satisfies all constraints and by definition is the *resource optimal constrained* path (see Definition 2). In the second case, this length is the maximum possible length of a simple path, and there will never be a case in which a complex path can be provided due to a contradiction with the *constrained shortest path* definition (see Definition 1).

In summary, we proved that the forward pass of NM estimates all possible hop count distances of simple paths between the source and the destination vertices in ascending order. At the same time, the backward pass can return any path of a given hop count length which provided by the forward pass to check for constraints satisfaction. Consequently, we conclude that - if the *constrained shortest path* exists, then NM will find it. ■

A.2 Asymptotic Complexity Analysis

Table A.1 summarizes common constrained shortest path algorithms' complexities.

Table A.1: Constrained Shortest Path Algorithms Asymptotic Complexity Summary

Case:	**EDijkstra [105]**	**IBF [58]**	**EDFS [53], EBFS [54, 106]**	**NM [52]**
l	Time: $O(\|V\| + \|E\| \cdot l)$ Space: $O(\|V\| + \|E\|)$ *resource optimal constrained path*	Time: $O(\|V\| + \|E\| \cdot l)$ Space: $O(\|V\| + \|E\|)$ *resource optimal constrained path*	Time: $O(\|V\| + \|E\| \cdot l)$ Space: $O(\|V\| + \|E\|)$ *resource optimal constrained path*	Time: $O(\|V\| + \|E\| \cdot l)$ Space: $O(\|V\| + \|E\|)$ *resource optimal constrained path*
$l/l \oplus 1$	Time: $O(\|V\|log\|V\| + \|E\| \cdot l)$ Space: $O(\|V\| + \|E\|)$ **constrained shortest path/** constrained path only	Time: $O(\|V\|\|E\| + \|E\| \cdot l)$ Space: $O(\|V\|\|E\|)$ **constrained shortest path/** *resource optimal constrained path*	Time: $O((\frac{\|E\|}{\|V\|})^{\|V\|} + \|E\| \cdot l)$ Space: $O((\frac{\|E\|}{\|V\|})^{\|V\|})$ **constrained shortest path**	Time: $O(\|V\|\|E\| + \|E\| \cdot l)$ Space: $O(\|V\|\|E\|)$ **constrained shortest path/** *resource optimal constrained path*
$l \oplus p$	N/A	N/A	Time: $O((\frac{\|E\|}{\|V\|})^{\|V\|}p + \|E\| \cdot l)$ Space:$O((\frac{\|E\|}{\|V\|})^{\|V\|}p)$ **constrained shortest path**	Time: $O((\frac{\|E\|}{\|V\|})^{\frac{\|V\|}{2}}p + \|E\| \cdot l)$ Space: $O((\frac{\|E\|}{\|V\|})^{\frac{\|V\|}{2}}p)$ **constrained shortest path**

Appendix B

A Near Optimal and Reliable Service Chain Orchestration Approach

B.1 Modeling Reliable Service Chain Composition

B.1.1 Symbols and Notations

All symbols and notations of sets, parameters, variables and functions that are used to model service chain composition under uncertainty are outlined in Table B.1.

B.1.2 Chance-Constraint Deterministic Equivalents

We derive here deterministic equivalents of probabilistic node capacity constraints (see Equation 5.15) given that application demands follow normal distribution. The derivation of deterministic equivalents for the service chain composition problem objective as well as for its link capacity constraints (see Equation 5.5) are similar. Note that if application demands have different than normal distribution, deterministic equivalents of the chance-constraints can be different. The risk of a physical node outage is a random discrete variable, hence the probability of a physical node capacity feasibility is:

Table B.1: Service Chain Composition Symbols and Notations

Service Chain Composition: Sets		
A	$\triangleq$	Set of service chains that needs to be composed
B^a	$\triangleq$	Set of backups for the service chain a
N_V^a	$\triangleq$	Set of service functions composing the service chain a
E_V^a	$\triangleq$	Set of service communications in the service chain a that create a service chain
$\mathcal{P}_a^{st}$	$\triangleq$	Set of metapaths for the st (single-link) service chain segment of service chain a
T	$\triangleq$	Set of QoS demands for service functions such as CPU, memory, storage, etc
K	$\triangleq$	Set of end-to-end network QoS demands for service chain communications such as latency, losses, jitter, etc
N_S	$\triangleq$	Set of physical nodes within the infrastructure
E_S	$\triangleq$	Set of physical links within the infrastructure
Service Chain Composition: Variables		
$x_i^s(b,a)$	$\triangleq$	Binary variable that equals to 1 if the service s backup b for the service chain a is placed on the physical node i
$f_{ij}^{st}(b,a)$	$\triangleq$	Binary variable that equals to 1 if the communication backup b between services s and t for the service chain a is placed on the physical link ij
$f_{ijk}^{st}(b,a)$	$\triangleq$	Binary variable that equals to 1 if the single-link chain segment st is assigned to the metapath P_{ijk}^{st} of the SFC a backup b
$\mathbf{D}_s^{a\tau}$	$\triangleq$	Random variable that corresponds to the service chain a service s SLO demand $\tau \in T$
$\mathbf{D}_{st}^{a}$	$\triangleq$	Random variable that corresponds to the communication bandwidth demand between services s and t for the service chain a
$\mathbf{C}_i^{\tau}$	$\triangleq$	Discrete random variable that corresponds to the possible physical node i capacity of $\tau \in T$ resource, i.e., $\mathbf{C}_i^{\tau} \in \{0, C_i^{\tau}\}$
$\mathbf{C}_{ij}$	$\triangleq$	Discrete random variable that corresponds to the possible physical link ij capacity, i.e., $\mathbf{C}_{ij} \in \{0, C_{ij}\}$
Service Chain Composition: Parameters		
gd_{si}^a	$\triangleq$	Parameter that corresponds to the geographical distance between the desired location of the service s in the service chain a and the physical node i location
$\mathcal{GD}_s^a$	$\triangleq$	Parameter that corresponds to the maximum allowable geographical distance between the desired location of the service s for the service chain a and some physical node location
w_{ij}^k	$\triangleq$	Parameter that corresponds to the additive weight of the physical link ij (or multiplicative when composed with *log* function) of the service communication end-to-end $k \in K$ QoS constraint (e.g., latency)
$\mathcal{K}_{st}^{ak}$	$\triangleq$	Parameter that corresponds to the communication end-to-end $k \in K$ QoS constraint (e.g., latency) between services s and t of the service chain a
R	$\triangleq$	service chain reliability probability that a chance-constraint will be satisfied
K_α	$\triangleq$	Constant in a standard Normal distribution table corresponding to desired α probability
$\mu_s^{a\tau}$	$\triangleq$	Expected (or mean) value of the service chain a service s QoS demand $\tau \in T$
μ_{st}^{a}	$\triangleq$	Expected (or mean) value of the the communication bandwidth demand between services s and t for the service chain a
$\sigma_s^{a\tau}$	$\triangleq$	Variance of the service chain a service s SLO demand $\tau \in T$
σ_{st}^{a}	$\triangleq$	Variance of the the communication bandwidth demand between services s and t for the service chain a
C_i^{τ}	$\triangleq$	Physical node i capacity of $\tau \in T$ resource (e.g., CPU)
C_{ij}	$\triangleq$	Physical link ij capacity

$$\mathbb{P}\left[\sum_{a\in A}\sum_{b\in B^a}\sum_{s\in N_V^a}\mathbf{D}_s^{a\tau}x_i^s(b,a)\leq \mathbf{C}_i^\tau\right] \cong= \mathbb{P}\left[\sum_{a\in A}\sum_{b\in B^a}\sum_{s\in N_V^a}\mathbf{D}_s^{a\tau}x_i^s(b,a)\leq 0\right]\cdot \bar{P}_i + \\ +\mathbb{P}\left[\sum_{a\in A}\sum_{b\in B^a}\sum_{s\in N_V^a}\mathbf{D}_s^{a\tau}x_i^s(b,a)\leq C_i^\tau\right]\cdot P_i,$$

where $\bar{P}_i$ is the outage risk of a physical node i, and $P_i = 1 - \bar{P}_i$. Moreover, without loss of generality, we assume that:

$$\mathbb{P}\left[\sum_{a\in A}\sum_{b\in B^a}\sum_{s\in N_V^a}\mathbf{D}_s^{a\tau}x_i^s(b,a)\leq 0\right]\cong 0.$$

Thus, our physical node capacity chance-constraint is the following:

$$\mathbb{P}\left[\sum_{a\in A}\sum_{b\in B^a}\sum_{s\in N_V^a}\mathbf{D}_s^{a\tau}x_i^s(b,a)\leq C_i^\tau\right]\geq \frac{R}{P_i} \tag{B.1}$$

Given that service chain demands follow a normal distribution $\mathbf{D}_s^{a\tau} \sim \mathcal{N}(\mu_s^{a\tau}, \sigma_s^{a\tau 2})$. Hence, $\sum\limits_{a\in A}\sum\limits_{s\in N_V^a}\mathcal{N}(\mu_s^{a\tau},\sigma_s^{a\tau 2}) = \mathcal{N}(\sum\limits_{a\in A}\sum\limits_{s\in N_V^a}\mu_s^{a\tau}, \sum\limits_{a\in A}\sum\limits_{s\in N_V^a}(\sigma_s^{a\tau 2} + \sum\limits_{a_1\neq a\in A}\sum\limits_{s_1\neq s\in N_V^a}\rho_{ss_1}^{aa_1\tau}\sigma_s^{a\tau}\sigma_{s_1}^{a_1\tau}))$. Moreover, we assume the worst-case scenario that all service chain demands are strongly correlated (i.e., $\rho_{ss_1}^{aa_1\tau} = 1$). Note that such assumption is also valid when service chain demands are not correlated or their correlation coefficients are unknown. In these cases, linear deterministic equivalents of chance-constraints always satisfy availability requirements R and can result in even higher (actual) availability at expense of worse resource utilization. Thus, given any two service chains a and a_1, their total deviation is $\sigma = \sqrt{\sigma_a^2 + \sigma_{a_1}^2 + 2\sigma_a\sigma_{a_1}} = \sqrt{(\sigma_a + \sigma_{a_1})^2} = \sigma_a + \sigma_{a_1}$.

As a results, chance-constraints in Equations 5.3 can be substituted with the following linear deterministic equivalents:

$$\sum_{a\in A}\sum_{b\in B^a}\sum_{s\in N_V^a}\left(\mu_s^{a\tau}+K_{\frac{R}{P_i}}\sigma_s^{a\tau}\right)x_i^s(a,b)\leq\begin{cases}C_i^\tau,\ R<P_i\\ 0,\text{otherwise}\end{cases},\forall i\in N_S,\tau\in T \tag{B.2}$$

where symbols and notations of sets, parameters, variables and functions are summarized in Appendix B.1.1.

B.2 Metapath Decomposition

B.2.1 Proof of The Optimal Single-Link Chain Composition

Proof. Assume the contrary. Let $L_1(s,t)$ be the optimal objective value of the single-link service chain st composition and $L_2(s,t)$ be a length of the constrained shortest metapath P_2. We need to show that $L_1 \neq L_2$:

Case 1 (*$L_1 < L_2$*): In this case, $L_1(s,t)$ solution is mapping of services s and t to physical nodes i and j, respectively, and a service link st to a physical path $P(i,j)$ as defined in the service chain composition problem. Without loss of generality, we can assume that the optimal solution of the service chain composition problem is feasible. Hence, s and t mappings are si and tj metalinks by Definition 2, respectively. Furthermore, let us define the path $P_1 = P(si, P(i,j), jt)$ which by Definition 3 is a metapath. As the optimal solution is feasible, P_1 satisfies all constraints of the single-link chain st composition. Hence, P_1 is a constrained metapath whose length $L_1(s,t)$ is shorter than $L_2(s,t)$ contradicting that P_2 is the constrained shortest metapath.

Case 2 (*$L_1 > L_2$*): In this case, we can present metapath P_2 as $P_2 = P(si, P(i,j), jt)$, where si and tj are metalinks, and $P(i,j)$ is a physical path (see Definition 2). Let us map services s and t on physical nodes i and j, respectively, and service link st on a physical path $P(i,j)$. As P_2 is the constrained metapath, this mapping is feasible with the objective value $L_2(s,t)$ less than $L_1(s,t)$ contradicting that $L_1(s,t)$ is the optimal objective value of the single-link service chain st composition. ■

B.2.2 Allowable Fitness Function Examples for Metapath Composite Variable Approach

Examples of service chain composition fitness functions include the following (additive and multiplicative) functions. In addition to the load balancing fitness function, a common example is the one that minimizes the service chain composition cost while increasing infrastructure providers' revenue [62]:

$$F_{ijk}^{sta} = \sum_{\tau \in T} (\mu_s^{a\tau} + K_{\frac{R}{P_i}} \sigma_s^{a\tau}) + \sum_{st \in E_V^a} \sum_{\substack{\{uv \in E_S: \\ uv \in P_{ijk}^{sta}\}}} (\mu_{st}^a + K_{\frac{R}{P_{uv}}} \sigma_{st}^a) + \sum_{\tau \in T} (\mu_t^{a\tau} + K_{\frac{R}{P_j}} \sigma_t^{a\tau}) \tag{B.3}$$

A common multiplicative fitness function example is a probability of the physical resources' availability, which are used to compose a service chain [148, 82]. This probability can be calculated as $\mathbb{P}\left[P_i \cdot \bigcap_{ij \in E_S} P_{ij} \cdot P_j\right] = P_i \cdot \prod_{\substack{\{uv \in E_S: \\ uv \in P_{ijk}^{sta}\}}} P_{uv} \cdot P_j$. To maximize this probability we can compose it with a $\log$ function and then minimize the following fitness function:

$$F_{ijk}^{sta} = -\log(P_i) - \sum_{st \in E_V^a} \sum_{\substack{\{uv \in E_S: \\ uv \in P_{ijk}^{sta}\}}} \log(P_{uv}) - \log(P_j) \tag{B.4}$$

Note that symbols and notations of sets, parameters, variables and functions are shown in Appendix B.1.1.

Bibliography

[1] Simon Knight, Hung X Nguyen, Nick Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[2] Ramakrishnan Durairajan, Subhadip Ghosh, Xin Tang, Paul Barford, and Brian Eriksson. Internet atlas: a geographic database of the internet. In *ACM workshop on HotPlanet*, pages 15–20. ACM, 2013.

[3] Brian Eriksson, Ramakrishnan Durairajan, and Paul Barford. Riskroute: a framework for mitigating network outage threats. In *ACM conference on Emerging networking experiments and technologies*, pages 405–416. ACM, 2013.

[4] António Fonseca, André Camões, and Teresa Vazão. Geographical routing implementation in ns3. In *ICST Conference on Simulation Tools and Techniques*, pages 353–358. ICST, 2012.

[5] Aruna Prem Bianzino, Claude Chaudet, Federico Larroca, Dario Rossi, and Jean-Louis Rougier. Energy-aware routing: a reality check. In *IEEE Workshop on GLOBECOM*, pages 1422–1427. IEEE, 2010.

[6] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. Achieving high utilization with software-driven

wan. In *ACM SIGCOMM Computer Communication Review*, pages 15–26. ACM, 2013.

[7] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM Computer Communication Review*, pages 3–14. ACM, 2013.

[8] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM Computer Communication Review*, pages 251–262. ACM, 1999.

[9] Floodlight sdn controller. `http://www.projectfloodlight.org`. Accessed: May, 2019.

[10] Neighborhood method prototype. `https://bitbucket.org/naas_cloud_computing_project/nm-of-controller/overview`. Accessed: May, 2019.

[11] Metapath-based reliable service chain orchestration repository. `https://bitbucket.org/duman190/mpsc_orchestration`. Accessed: May, 2019.

[12] George Thoma, Sameer Antani, Michael Gill, Glenn Pearson, and Leif Neve. People locator: A system for family reunification. *IT Professional*, 14(3):13–21, 2012.

[13] Rengarajan Pelapur, Sema Candemir, Filiz Bunyak, Mahdieh Poostchi, Guna Seetharaman, and Kannappan Palaniappan. Persistent target tracking using likelihood fusion in wide-area and full motion video sequences. In *International Conference on Information Fusion*, pages 2420–2427. IEEE, 2012.

[14] Nathan Schurr, Janusz Marecki, Milind Tambe, Paul Scerri, Nikhil Kasinadhuni, and John P Lewis. The future of disaster response: Humans working with multiagent teams using defacto. In *AAAI spring symposium: AI technologies for homeland security*, pages 9–16, 2005.

[15] Joshua C Klontz and Anil K Jain. A case study on unconstrained facial recognition using the boston marathon bombings suspects. *Michigan State University, Tech. Rep*, 119(120):1, 2013.

[16] Rasha Gargees, Brittany Morago, Rengarajan Pelapur, Dmitrii Chemodanov, Prasad Calyam, Zakariya Oraibi, Ye Duan, Guna Seetharaman, and Kannappan Palaniappan. Incident-supporting visual cloud computing utilizing software-defined networking. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(1):182–197, 2017.

[17] Dmitrii Chemodanov, Prasad Calyam, and Flavio Esposito. A near optimal reliable composition approach for geo-distributed latency sensitive service chains. In *IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2019.

[18] John Gillis, Prasad Calyam, Ashley Bartels, Mihai Popescu, Stephen Barnes, Jennifer Doty, Dena Higbee, and Salman Ahmad. Panacea's glass: Mobile cloud framework for communication in mass casualty disaster triage. In *IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 128–134. IEEE, 2015.

[19] B. Morago, G. Bui, and Y. Duan. Integrating LiDAR range scans and photographs with temporal changes. *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 732–737, 2014.

[20] Dmitrii Chemodanov, Flavio Esposito, Andrei Sukhov, Prasad Calyam, Huy Trinh, and Zakariya Oraibi. Agra: Ai-augmented geographic routing approach for iot-based

incident-supporting applications. *Future Generation Computer Systems*, 92:1051–1065, 2019.

[21] Huy Trinh, Prasad Calyam, Dmitrii Chemodanov, Shizeng Yao, Qing Lei, Fan Gao, and Kannappan Palaniappan. Energy-aware mobile edge computing and routing for low-latency visual data processing. *IEEE Transactions on Multimedia*, 20(10):2562–2577, 2018.

[22] Sarita Chung, C Mario Christoudias, Trevor Darrell, Sonja I Ziniel, and Leslie A Kalish. A novel image-based tool to reunite children with their families after disasters. *Academic emergency medicine*, 19(11):1227–1234, 2012.

[23] Davis E King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10(Jul):1755–1758, 2009.

[24] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *British Machine Vision Conference*, page 6, 2015.

[25] M. Kwan and D. Ransberger. LiDAR assisted emergency response: Detection of transport network obstructions caused by major disasters. *Computers, Environment and Urban Systems*, pages 179–188, 2010.

[26] R. Hartley and A. Zisserman. *Multiple View Geometry*. Cambridge University Press, 2010.

[27] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 246–253, 1999.

[28] K. Palaniappan, R. Rao, and G. Seetharaman. Wide-area persistent airborne video: Architecture and challenges. *Distributed Video Sensor Networks*, pages 349–371, 2011.

[29] A. Hafiane, G. Seetharaman, K. Palaniappan, and B. Zavidovique. Rotationally invariant hashing of median patterns for texture classification. *Lecture Notes in Computer Science (LNCS)*, pages 619–629, 2008.

[30] H. Aliakbarpour, K. Palaniappan, and G. Seetharaman. Robust camera pose refinement and rapid SfM for multiview aerial imagery Without RANSAC. *IEEE Geoscience and Remote Sensing Letters (GRSL)*, pages 2203–2207, 2015.

[31] A. Hafiane, K. Palaniappan, and G. Seetharaman. UAV-Video registration using block-based features. In *IEEE International Symposium on Geoscience and Remote Sensing (IGARSS)*, pages 1104–1107, 2008.

[32] K. Palaniappan, F. Bunyak, P. Kumar, I. Ersoy, S. Jaeger, K. Ganguli, A. Haridas, J. Fraser, R. Rao, and G. Seetharaman. Efficient feature extraction and likelihood fusion for vehicle tracking in low frame rate airborne video. *IEEE Conference on Information Fusion (FUSION)*, pages 1–8, 2010.

[33] Josiah Burchard, Dmitrii Chemodanov, John Gillis, and Prasad Calyam. Wireless mesh networking protocol for sustained throughput in edge computing. In *IEEE International Conference on Computing, Networking and Communications (ICNC)*, pages 958–962. IEEE, 2017.

[34] Andrej Cvetkovski and Mark Crovella. Hyperbolic embedding and routing for dynamic graphs. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1647–1655. IEEE, 2009.

[35] Brad Karp and Hsiang-Tsung Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *ACM International Conference on Mobile Computing and Networking*, pages 243–254. ACM, 2000.

[36] E Kranakis, H Singh, and J Urrutia. Compass routing on geometric networks. In *IEEE Conference on Computational Geometry*. IEEE, 1999.

[37] AM Sukhov and D Yu Chemodanov. The neighborhoods method and routing in sensor networks. In *IEEE Conference on Wireless Sensor (ICWISE)*, pages 7–12. IEEE, 2013.

[38] Robert Kleinberg. Geographic routing using hyperbolic space. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1902–1909. IEEE, 2007.

[39] Ben Leong, Barbara Liskov, and Robert Morris. Geographic routing without planarization. In *NSDI*, volume 6, page 25, 2006.

[40] Michal Król, Eryk Schiller, Franck Rousseau, and Andrzej Duda. Weave: Efficient geographical routing in large-scale networks. In *EWSN*, pages 89–100, 2016.

[41] Simon S Lam and Chen Qian. Geographic routing in d-dimensional spaces with guaranteed delivery and low stretch. *IEEE/ACM Transactions on Networking (TON)*, 21(2):663–677, 2013.

[42] Sahel Sahhaf, Wouter Tavernier, Didier Colle, Mario Pickavet, and Piet Demeester. Experimental validation of resilient tree-based greedy geometric routing. *Computer Networks*, 82:156–171, 2015.

[43] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[44] Aws lambda. `https://aws.amazon.com/lambda/`. Accessed: May, 2019.

[45] Google cloud functions. `https://cloud.google.com/functions/`. Accessed: May, 2019.

[46] Microsoft azure functions. `https://azure.microsoft.com/en-us/services/functions/`. Accessed: May, 2019.

[47] Ibm openwhisk. https://www.ibm.com/cloud/functions. Accessed: May, 2019.

[48] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in nfv: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[49] Yipei Niu, Fangming Liu, and Zongpeng Li. Load balancing across microservices. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 198–206. IEEE, 2018.

[50] Ruozhou Yu, Guoliang Xue, and Xiang Zhang. Application provisioning in fog computing-enabled internet-of-things: A network perspective. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 783–791. IEEE, 2018.

[51] Richard Cziva, Christos Anagnostopoulos, and Dimitrios P Pezaros. Dynamic, latency-optimal vnf placement at the network edge. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 693–701. IEEE, 2018.

[52] Dmitrii Chemodanov, Prasad Calyam, Flavio Esposito, and Andrei Sukhov. A general constrained shortest path approach for virtual path embedding. In *International Symposium on Local and Metropolitan Area Networks*, pages 1–7. IEEE, 2016.

[53] Leonardo Lozano and Andrés L Medaglia. On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384, 2013.

[54] Xinming Chen, Hao Cai, and Tilman Wolf. Multi-criteria routing in networks with path choices. In *International Conference on Network Protocols*, pages 334–344. IEEE, 2015.

[55] Piet Van Mieghem and Fernando A Kuipers. Concepts of exact qos routing algorithms. *IEEE/ACM Transactions on Networking*, 12(5):851–864, 2004.

[56] Rosario G Garroppo, Stefano Giordano, and Luca Tavanti. A survey on multi-constrained optimal path computation: Exact and approximate algorithms. *Computer Networks*, 54(17):3081–3107, 2010.

[57] Jeffrey M Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14(1):95–116, 1984.

[58] Klara Nahrstedt and Shigang Chen. Coexistence of qos and best-effort flows. In *Multimedia Communications*, pages 175–188. Springer, 1999.

[59] Xin Yuan. Heuristic algorithms for multiconstrained quality-of-service routing. *IEEE/ACM Transactions on Networking (TON)*, 10(2):244–256, 2002.

[60] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on Networking*, 20(1):206–219, 2012.

[61] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, and Raouf Boutaba. A path generation approach to embedding of virtual networks. *IEEE Transactions on Network and Service Management*, 12(3):334–348, 2015.

[62] Yang Wang, Qian Hu, and Xiaojun Cao. A branch-and-price framework for optimal virtual network embedding. *Computer Networks*, 94:318–326, 2016.

[63] Rahul Potharaju and Navendu Jain. Demystifying the dark side of the middle: a field study of middlebox failures in datacenters. In *Proceedings of Internet Measurement Conference*, pages 9–22. ACM, 2013.

[64] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. Adaptive vnf scaling and flow routing with proactive demand prediction. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 486–494. IEEE, 2018.

[65] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[66] Andrea Tomassilli, Frédéric Giroire, Nicolas Huin, and Stéphane Pérennes. Provably efficient algorithms for placement of sfcs with ordering constraints. In *IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2018.

[67] Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. Near optimal placement of virtual network functions. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1346–1354. IEEE, 2015.

[68] Yu Sang, Bo Ji, Gagan R Gupta, Xiaojiang Du, and Lin Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9. IEEE, 2017.

[69] Hao Feng, Jaime Llorca, Antonia M Tulino, Danny Raz, and Andreas F Molisch. Approximation algorithms for the nfv service distribution problem. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9. IEEE, 2017.

[70] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[71] Aurojit Panda, Wenting Zheng, Xiaohe Hu, Arvind Krishnamurthy, and Scott Shenker. Scl: Simplifying distributed sdn control planes. In *NSDI*, pages 329–345, 2017.

[72] Diego Ongaro and John K Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*, pages 305–319, 2014.

[73] Flavio Esposito. Catena: A distributed architecture for robust service function chain instantiation with guarantees. In *IEEE Conference o Network Softwarization*, pages 1–9. IEEE, 2017.

[74] Flavio Esposito, Donato Di Paola, and Ibrahim Matta. On distributed virtual network embedding with guarantees. *IEEE/ACM Transactions on Networking*, 24(1):569–582, 2016.

[75] Wolfgang KH Panofsky and Melba Phillips. *Classical electricity and magnetism*. Courier Corporation, 2005.

[76] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 14(14):527, 2008.

[77] Charles Perkins, Elizabeth Belding-Royer, and Samir Das. Ad hoc on-demand distance vector (aodv) routing. Technical report, 2003.

[78] Guido R Hiertz, Dee Denteneer, Sebastian Max, Rakesh Taori, Javier Cardona, Lars Berlemann, and Bernhard Walke. Ieee 802.11 s: the wlan mesh standard. *IEEE Wireless Communications*, 17(1), 2010.

[79] Cynthia Barnhart, Amr Farahat, and Manoj Lohatepanont. Airline fleet assignment with enhanced revenue modeling. *Operations research*, 57(1):231–244, 2009.

[80] Marshall L Fisher. The lagrangian relaxation method for solving integer programming problems. *Management science*, 27(1):1–18, 1981.

[81] Vishv Jeet and Erhan Kutanoglu. Lagrangian relaxation guided problem space search heuristics for generalized assignment problems. *European Journal of Operational Research*, 182(3):1039–1056, 2007.

[82] Bart Spinnewyn, Ruben Mennes, Juan Felipe Botero, and Steven Latré. Resilient application placement for geo-distributed cloud networks. *Journal of Network and Computer Applications*, 85:14–31, 2017.

[83] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[84] Spyros Gidaris and Nikos Komodakis. Object detection via a multi-region and semantic segmentation-aware cnn model. In *IEEE International Conference on Computer Vision*, pages 1134–1142, 2015.

[85] Mark Berman, Jeffrey S Chase, Lawrence Landweber, Akihiro Nakao, Max Ott, Dipankar Raychaudhuri, Robert Ricci, and Ivan Seskar. Geni: A federated testbed for innovative network experiments. *Computer Networks*, 61:5–23, 2014.

[86] Blesson Varghese and Rajkumar Buyya. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 79:849–861, 2018.

[87] Flavio Esposito, Ibrahim Matta, and Vatche Ishakian. Slice embedding solutions for distributed service architectures. *ACM Computing Surveys*, 46(1):6, 2013.

[88] Sandra Herker, Ashiq Khan, and Xueli An. Survey on survivable virtual network embedding problem and solutions. In *International Conference on Networking and Services*, 2013.

[89] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[90] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A survey on sensor networks. *IEEE Communications magazine*, 40(8):102–114, 2002.

[91] Tariq Umer, Muhammad Amjad, Muhammad Khalil Afzal, and Muhammad Aslam. Hybrid rapid response routing approach for delay-sensitive data in hospital body area sensor network. In *ACM International Conference on Computing Communication and Networking Technologies*, page 3. ACM, 2016.

[92] Fayaz Akhtar, Mubashir Husain Rehmani, and Martin Reisslein. White space: Definitional perspectives and their role in exploiting spectrum opportunities. *Telecommunications Policy*, 40(4):319–331, 2016.

[93] Athar Ali Khan, Mubashir Husain Rehmani, and Abderrezak Rachedi. Cognitive-radio-based internet of things: Applications, architectures, spectrum related functionalities, and future research directions. *IEEE Wireless Communications*, 24(3):17–25, 2017.

[94] Claude E Shannon. Prediction and entropy of printed english. *Bell Labs Technical Journal*, 30(1):50–64, 1951.

[95] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[96] Sangsu Jung, Malaz Kserawi, Dujeong Lee, and JK K Rhee. Distributed potential field based routing and autonomous load balancing for wireless mesh networks. *IEEE Communications Letters*, 13(6), 2009.

[97] Mehdi Kalantari and Mark Shayman. Routing in wireless ad hoc networks by analogy to electrostatic theory. In *IEEE International Conference on Communications*, volume 7, pages 4028–4033. IEEE, 2004.

[98] Nam T Nguyen, An-I Andy Wang, Peter Reiher, and Geoff Kuenning. Electric-field-based routing: a reliable framework for routing in manets. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8(2):35–49, 2004.

[99] Prosenjit Bose, Pat Morin, Ivan Stojmenović, and Jorge Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[100] Young-Jin Kim, Ramesh Govindan, Brad Karp, and Scott Shenker. Geographic routing made practical. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 217–230. USENIX Association, 2005.

[101] Jiangwei Zhou, Yu Chen, Ben Leong, and Pratibha Sundar Sundaramoorthy. Practical 3d geographic routing for wireless sensor networks. In *ACM Conference on Embedded Networked Sensor Systems*, pages 337–350. ACM, 2010.

[102] Fragkiskos Papadopoulos, Dmitri Krioukov, Marián Boguñá, and Amin Vahdat. Greedy forwarding in dynamic scale-free networks embedded in hyperbolic metric spaces. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–9. IEEE, 2010.

[103] Pejman Khadivi, Shadrokh Samavi, and Terence D Todd. Multi-constraint qos routing using a new single mixed metrics. *Journal of Network and Computer Applications*, 31(4):656–676, 2008.

[104] Alpar Juttner, Balazs Szviatovski, Ildikó Mécs, and Zsolt Rajkó. Lagrange relaxation based method for the qos routing problem. In *IEEE International Conference on Computer Communications (INFOCOM)*, volume 2, pages 859–868. IEEE, 2001.

[105] Z Zhang and Jon Crowcroft. Qos routing for supporting resource reservation. *IEEE Journal on Selected areas in Communications*, 14(7):1228–1234, 1996.

[106] Ron Widyono. *The design and evaluation of routing algorithms for real-time channels*. International Computer Science Institute Berkeley, 1994.

[107] Linqi Guo, John Pang, and Anwar Walid. Joint placement and routing of network function chains in data centers. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 612–620. IEEE, 2018.

[108] Xincai Fei, Fangming Liu, Hong Xu, and Hai Jin. Towards load-balanced vnf assignment in geo-distributed nfv infrastructure. In *International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2017.

[109] National oceanic and atmospheric organization. `https://geodesy.noaa.gov/storm_archive/storms/joplin/index.html`. Accessed: May, 2019.

[110] Robert Osserman. The isoperimetric inequality. *Bulletin of the American Mathematical Society*, 84(6):1182–1238, 1978.

[111] Dazhi Chen and Pramod K Varshney. A survey of void handling techniques for geographic routing in wireless networks. *IEEE Communications Surveys & Tutorials*, 9(1):50–67, 2007.

[112] Erik Kuiper and Simin Nadjm-Tehrani. Geographical routing in intermittently connected ad hoc networks. In *International Conference on Advanced Information Networking and Applications-Workshops*, pages 1690–1695. IEEE, 2008.

[113] Anders Lindgren, Avri Doria, and Olov Schelen. Probabilistic routing in intermittently connected networks. In *Service Assurance with Partial and Intermittent Resources*, pages 239–254. Springer, 2004.

[114] Mokhtar S Bazaraa, John J Jarvis, and Hanif D Sherali. *Linear programming and network flows*. John Wiley & Sons, 2011.

[115] Marco Chiesa, Gábor Rétvári, and Michael Schapira. Lying your way to better traffic engineering. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 391–398. ACM, 2016.

[116] Stefano Vissicchio, Olivier Tilmans, Laurent Vanbever, and Jennifer Rexford. Central control over distributed routing. In *ACM SIGCOMM Computer Communication Review*, pages 43–56. ACM, 2015.

[117] Emilie Danna, Subhasree Mandal, and Arjun Singh. A practical algorithm for balancing the max-min fairness and throughput objectives in traffic engineering. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 846–854. IEEE, 2012.

[118] Dritan Nace, Linh Nhat Doan, Olivier Klopfenstein, and Alfred Bashllari. Max–min fairness in multi-commodity flows. *Computers & Operations Research*, 35(2):557–573, 2008.

[119] David Eppstein. Finding the k shortest paths. *SIAM Journal on computing*, 28(2):652–673, 1998.

[120] Edoardo Amaldi, Stefano Coniglio, Arie MCA Koster, and Martin Tieves. On the computational complexity of the virtual network embedding problem. *Electronic Notes in Discrete Mathematics*, 52:213–220, 2016.

[121] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *ACM Workshop on Virtualized Infrastructure Systems and Architectures*, pages 81–88. ACM, 2009.

[122] Jorge Londono and Azer Bestavros. Netembed: A network resource mapping service for distributed applications. In *IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8. IEEE, 2008.

[123] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[124] Yong Zhu and Mostafa H Ammar. Algorithms for assigning substrate network resources to virtual network components. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1–12, 2006.

[125] Michael Till Beck, Andreas Fischer, Juan Felipe Botero, Claudia Linnhoff-Popien, and Hermann de Meer. Distributed and scalable embedding of virtual networks. *Journal of Network and Computer Applications*, 56:124–136, 2015.

[126] Fady Samuel, Mosharaf Chowdhury, and Raouf Boutaba. Polyvine: policy-based virtual network embedding across multiple domains. *Journal of Internet Services and Applications*, 4(1):6, 2013.

[127] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[128] Turgay Korkmaz and Marwan Krunz. Multi-constrained optimal path selection. In *IEEE International Conference on Computer Communications (INFOCOM)*, volume 2, pages 834–843. IEEE, 2001.

[129] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[130] Mikkel Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, 1999.

[131] Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[132] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. Brite: An approach to universal topology generation. In *IEEE Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 346–353. IEEE, 2001.

[133] Bernard M Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.

[134] Ibm cplex solver. `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`. Accessed: May, 2019.

[135] Ketan Bhardwaj, Ming-Wei Shih, Pragya Agarwal, Ada Gavrilovska, Taesoo Kim, and Karsten Schwan. Fast, scalable and secure onloading of edge functions using airbox. In *IEEE/ACM Symposium on Edge Computing*, pages 14–27. IEEE, 2016.

[136] Amedeo Sapio, Ibrahim Abdelaziz, Abdulla Aldilaijan, Marco Canini, and Panos Kalnis. In-network computation is a dumb idea whose time has come. In *ACM Workshop on Hot Topics in Networks*, pages 150–156. ACM, 2017.

[137] John R Birge and Francois Louveaux. *Introduction to stochastic programming*. Springer Science & Business Media, 2011.

[138] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network flows*. Elsevier, 2014.

[139] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[140] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.

[141] Robert Ricci, Eric Eide, and CloudLab Team. Introducing cloudlab: Scientific infrastructure for advancing cloud architectures and applications. *Login: the magazine of USENIX & SAGE*, 39(6):36–38, 2014.

[142] Thomas Blaschke. Object based image analysis for remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(1):2–16, 2010.

[143] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[144] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[145] Docker containers. `https://www.docker.com/`. Accessed: May, 2019.

[146] Open vswitch. `http://openvswitch.org`. Accessed: May, 2019.

[147] Anna Ellis, Ali Shahrokni, and James Michael Ferryman. Pets2009 and winter-pets 2009 results: A combined evaluation. In *International Workshop on Performance Evaluation of Tracking and Surveillance*, pages 1–8. IEEE, 2009.

[148] Ali Hmaity, Marco Savi, Francesco Musumeci, Massimo Tornatore, and Achille Pattavina. Virtual network function placement for resilient service chain provisioning. In *International Workshop on Resilient Networks Design and Modeling*, pages 245–252. IEEE, 2016.

VITA

Dmitrii Chemodanov was born in Samara, Russia. There, he attended the Samara State Aerospace University where he received both his B.S. and M.S. degrees from the Department of Computer Science in Applied Math and Physics in 2012 and 2014, respectively.

In July 2019, he will be completing his Ph.D. in Computer Science at the University of Missouri. Since August 2014, Dmitrii has joined Prof. Prasad Calyam's Virtualization, Multimedia and Networking (VIMAN) Lab. While at VIMAN, Dmitrii has done presentations, demos and tutorials in near 10 professional forums worldwide and published near 20 peer-reviewed publications in reputed conferences and journals including IEEE INFOCOM, Elsevier FGCS (IF: 4.64), IEEE TNSM (IF: 3.29), IEEE TSC (IF: 4.42), IEEE TMM (IF: 3.98), IEEE TCSVT (IF: 3.56), Elsevier PMC (IF: 2.95) and others. Moreover, Dmitrii's research on visual cloud computing and reliable network virtualization helped Prof. Prasad Calyam (Principle Investigator) to secure $\sim$1 Million in Sponsored-research Funding from the National Science Foundation. During graduate school, Dmitrii has also mentored students in Cloud Computing I and II courses of Prof. Prasad Calyam (CS 4530/7530 and CS 8001) research projects as well as given guest lectures.

In August 2019, Dmitrii will be joining Network Infrastructure Group of Google Cloud Platform at Google, Sunnyvale, CA as a Software Engineer/Researcher. His current research interests include distributed and cloud computing; network management; routing; design, implementation and evaluation of algorithms and protocols for service-based architectures, such as Software Define Networks (SDN); modeling and performance evaluation of wireless, and peer-to-peer networks. In addition to aforementioned practical areas, he is also interested in (more visionary) theoretical areas such as traffic engineering in dynamic scale-free network models for next-generation IoT, Edge and Internet infrastructures as well as in designing new Quantum Computing algorithms tailored to his area of interests.