Distributed Provisioning of 5G Service Requests

Ying Rao

A Thesis
in
Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of
Master's of Computer Science at
Concordia University
Montréal, Québec, Canada

February 2022

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By:	**Ying Rao**
Entitled:	**Distributed Provisioning of 5G Service Requests**

and submitted in partial fulfillment of the requirements for the degree of

Master's of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Tse Hsun Chen	Chair
Dr. Olga Ormandjieva	Examiner
Dr. Tse Hsun Chen	Examiner
Dr. Brigitte Jaumard	Thesis Supervisor
Dr. Kim Khoa Nguyen	Thesis Supervisor

Approved by ______________________________
Dr. Leila Kosseim, Graduate Program Director

February 21, 2022 ______________________________
Dr. Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Distributed Provisioning of 5G Service Requests

Ying Rao

Service function chain (SFC) plays a prominent role in realizing 5G Slicing and next-generation networks. Supported by the emerging techniques such as Software-Defined Networking (SDN) and Network Function Virtualization (NFV), network operators can freely define and configure a variety of complex network services or SFCs based on business needs, policies, and quality of service (QoS) requirements.

Additionally, 5G networks are expected to be highly dynamic, constantly upgrading and scaling, in which case a plug-and-play mechanism is needed to avoid high operational and management costs, for which the flexible and scalable distributed system is well suited.

This thesis investigates the distributed SFC provisioning problem, and propose a fully distributed algorithm that runs with dynamic traffic aiming to find and reserve the best suited network and compute resources for each SFC request with manageable messaging costs.

We implemented the resulting algorithm in the OMNET++ environment and conducted a series of experiments with different dynamic traffic instances running on a distributed network. We then compare its performance with a classical centralized resource constraint shortest path (RCSP) algorithm, the results show that our 5G provisioning algorithm obtains a similar performance to the centralized RCSP in terms of throughput and acceptance rate.

Acknowledgments

I would like to express my deepest gratitude to my supervisor Prof. Brigitte Jaumard and Prof. Kim Khoa Nguyen. Their attentive guidance and unremitting support at each step throughout my study have deeply motivated and inspired me. Their profound knowledge, rigorous academic spirit and constructive suggestions have been and will always be of great value to me.

In addition, I would like to express my special thanks of gratitude to Mitacs and Ciena, for their generous financial and technical support on this work.

Also, I am very grateful to my lab colleagues for their help and support. Especially Dr. Quang Huy Duong, he is very knowledgeable and patient.

Lastly, I would like to thank my family and my boyfriend for giving me as much moral support and encouragement as they could.

Contents

Acronyms

2P-DSP 2-phase distributed SFC provisioning. 12

CAPEX capital expenditure. 2

COTS commercial off-the-shelf. 2

DAG directed acyclic graph. 48

DMR distributed multi-constrained routing. 7

E2E End-to-End. 4

IDS Intrusion Detection Systems. 2

IPS Intrusion Prevention Systems. 2

LB Load Balancing. 2

NFV Network Function Virtualization. iii, 1

OPEX operating expenditure. 2

QoS quality of service. iii, 2

RCSP resource constraint shortest path. iii

SDN	Software-Defined Networking. iii, 1
SFC	service function chain. iii, 2, 47, 48
SSSP	single-source shortest paths. 6
VNF	Virtualized Network Functions. 2, 8, 47

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Networking Background: 5G Slicing

Over the past few decades, mobile networks have evolved to provide us with voice, messaging, video, data and a variety of other services that have revolutionized the way we live. Compared to its predecessors, 5G will open up a new era of "Internet of Everything" [8], providing ultra-fast, reliable and ubiquitous connectivity with negligible latency, and connecting a massive number of individuals and devices [5]. These connected devices come from different industrial fields with different characteristics and needs, in other words, they have different requirements for the network, such as security, reliability, latency and speed. Therefore, 5G networks must be as flexible, convenient and versatile as possible.

In this context, network slicing is proposed as a solution. Network slicing divides the physical network into multiple independent logical network slices, each virtual slice corresponding to a different type of applications [23], such that network operators can select features required for each slice to meet different service requirements, such as less latency, higher throughput, higher traffic capacity and connection density. Software-Defined Networking (SDN) and Network Function Virtualization (NFV) are the key enabler technologies of network slicing.

NFV promises a flexible and cost-efficient way to deploy network services by replacing the traditional dedicated network hardware devices with the software applications that are running on the commercial off-the-shelf (COTS) servers. These software applications are called Virtualized Network Functions (VNF)s, VNFs can be freely defined to implement various network functions such as IDS /IPS, firewalls, LB, etc. By deploying these VNFs on the industry-standard commercial servers, NFV significantly reduces capital expenditure (CAPEX) and operating expenditure (OPEX), while increasing flexibility by removing location constraints and allowing network functions to be deployed anywhere on demand[9]. Meanwhile, SDN architecture decouples control plane from data plane, providing centralized programmatic control over network functions and the traffic flow.

In more complex cases, where a data stream needs to pass through multiple network services or VNF instances in a specific or partially specific order to finally reach its destination, a service function chain (SFC) is created. In network slicing, SFC plays a key role in providing various types of complex services for each network slice. On the other hand, how to effectively deploy SFCs to satisfy QoS while maximizing the revenue of virtual operators is a complex and challenging task, and has been an active topic in 5G networks.

Given the prominent value of SFC in implementing 5G slicing and the future generation network, there is a growing number of researches focus on the design and implementation SFC provisioning techniques[9]. However, most of the existing study focus on centralized provisioning of SFC with very few of them discuss this problem in distributed manner. On the other hand, the fast-evolving 5G network is envisioned to be highly dynamic and the future can be hard to predict [15], i.e., new components and devices are constantly introduced to expand or upgrade current services and performance of the system. In this case, a more flexible and efficient way of service deployment is needed to manage the high operational costs, error-prone and time-consuming situations. From this perspective, distributed systems are better suited for this plug-and-play manner, as they exhibit higher scalability than centralized systems, and are more adaptable therefore more robust in handling dynamic changes, especially in large-scale networks.

However, it is not easy to realize distributed provisioning for SFC requests. In the absence of a global view or centralized supervision, it poses several major challenges, such as synchronization, fault handling and provisioning efficiency.

1.2 Our Research Project: Distributed SFC Provisioning

In this thesis, we investigate on the designing of a distributed SFC provisioning algorithm that runs in real-time dynamic network. Here we give a simple example to illustrate the problem graphically.

Inputs

The physical network is represented by Fig.1, node R1$\sim$8 represent routers scattered in different geographical locations, node S1$\sim$5 represent servers or data centers (DCs), these nodes are connected by a set of links as shown. Each DC provides certain available VNF types and a certain amount of computing resources, including CPU, RAM, and storage. Every link is associated with certain latency and bandwidth capacity. All the nodes within the network only communicate with their direct neighbouring nodes.

Each VNF type has different computing resource requirements in terms of CPU, RAM and storage, and it takes a certain amount of processing time to process a data packet on a data center node. In the given example, for simplicity, we set the VNF requirements as follows: VNF1 requires 2 CPUs and 1 ms processing time, VNF2 requires 50GB of storage and 2 ms processing time, VNF3 requires 2 RAMs and 3 ms processing time.

There are incoming and departing requests in the network at any time. Each request is defined by its characteristics, including: its source and destination, its live-time from its arrival, bandwidth and E2E latency requirements, and the SFC that specifies the required VNFs and their execution order. Take the two SFC requests in Fig.1 as examples.

Note that the E2E delay requirement includes the link delay and the VNF processing time. For each SFC request that arrives at the system, the operator needs to make a decision in real-time, and in order to grant a request, the operator needs to provide a routing path and reserve network and compute resources for that request.

For each request, we intend to find a path in the given network that goes through particular nodes following the VNF sequence in SFC, while subject to the computational and connectivity constraints. For example, for request 1, a feasible path will be: Path 1 = (R1(SRC), R2(book VNF1), R8(book VNF3), R4, R6(DST)), similarly, for request 2, the path can be: Path 2 = (R5(SRC, book VNF1), R3, R8, R4(book VNF2, DST)). Path 1 and 2 are highlighted purple and blue in Fig.1.

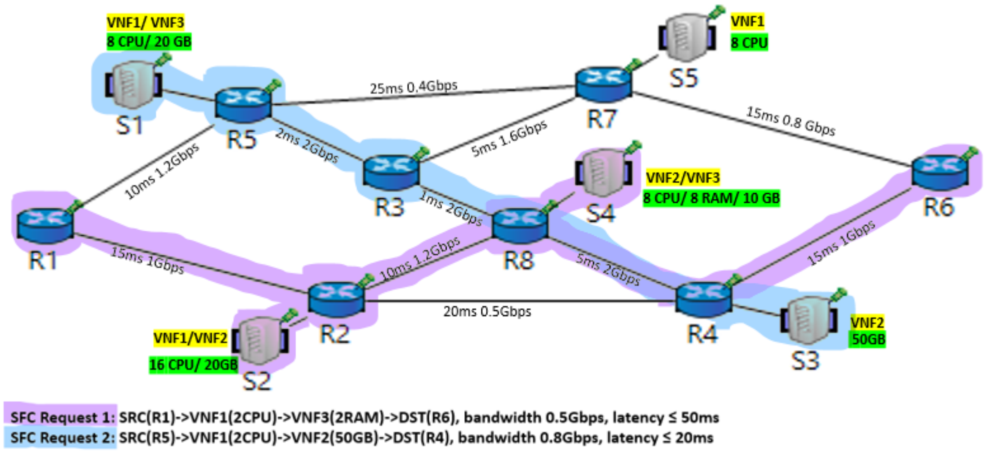


Figure 1: Example distributed SFC request provisioning

Outputs

Based on the inputs described in the previous section, our goal is to design an algorithm that runs on each node while, through cooperation, the nodes search for resources in the network for each request and assign those resources to that request, subject to the following objectives:

- Maximizing the overall number of granted requests in the system,
- Minimizing communication complexity, i.e., the average number generated messages to serve a request,

1.3 Key References

Our survey on existing literature mainly goes in two directions: distributed SFC provisioning and related graph theory algorithms.

1.3.1 Distributed SFC Provisioning

The SFC provisioning problem is gaining popularity in this area and many different approaches have been proposed, while most focus on centralized supervision, i.e., SDN controllers, as listed out in [14]. Few studies have addressed distributed SFC routing and allocation problems, this can be further confirmed in [20] and [11]. [20] systematically reviews the progress of SFC placement in distributed scenarios over the last decade; however, the distributed scenarios presented are either SDN-enabled distributed clouds, multiple domains, or geographically distributed data centers, that are essentially computing SFC routing paths by using centralized (shortest) path algorithms. The closet effort to our problem is [11] which can be considered the most recent work, where the authors propose a central-distributed SFC path selection mechanism that computes routing paths distributed under the monitoring of a central controller, the central controller will re-compute a new path for a request when the path gets too costly.

On the other hand, we have seen some worthwhile learning from some centralized approaches. In [3] and [17], the authors proposed a solution to the management of VNF chaining by running the shortest path algorithm on a multi-layer graph so that the SFC routing path will traverse each VNF hosting node in the specified order, which also inspires us to apply the layered-graph paradigm to the distributed SFC routing path computation problem.

1.3.2 Graph Theory Algorithms

Next, we move on to studies of the related graph theory algorithms, including distributed shortest path problem with and without constraint management.

Distributed Shortest Path Algorithms

The work of M.Elkin *et al.* [4] has been considered to be influential and a breakthrough in solving the exact distributed single-source shortest paths (SSSP) problem, in which they propose an algorithm that solves the exact SSSP problem in the CONGEST model in sub-linear time, that is $O((n \log n)^{5/6})$ time for $D = O(\sqrt{n} \log n)$, and $O(D^{1/3} \times (n \log n)^{2/3})$ time for larger D. The main idea behind it is to first construct a hop-set G'' of the skeleton graph G' constructed from the original graph G, and then perform Bellman-Ford exploration in $G'' \bigcup G'$, with the difference that M. Elkin did not sample and construct G' in the first step, but G' was built on the fly in the second step, such that the resulting G' is exact rather than approximated. M. Elkin's work has led to a number of advances, i.e., U. Agarwal *et al.*[1] presents a Õ($n^{4/3}$) round algorithm for the distributed weighted all pairs shortest paths (APSP) problem in both un-directed and directed graphs, which improved the previous Õ($n^{3/2}$) rounds. M. Ghaffari *et al.* [7] propose a new single-source shortest path(SSSP) algorithm with complexity Õ($n^{3/4}$ $D^{1/4}$) compared to M.Elkin's Õ($n^{5/6}$) bound, and for larger D value, the complexity is Õ($n^{3/4+o(1)}$ + min($n^{3/4}$ $D^{1/6}$,$n^{6/7}$)+D) compared to M.Elkin's Õ($n^{5/6}$+$n^{2/3}$ $D^{1/3}$+D) bound, where D is the hop-diameter of the graph.

However, from a technical point of view, applying M. Elkin's algorithm and similar approaches to our distributed SFC routing problem has several issues, such as difficulty in adding constraints, the complexity gap between the synchronous model being used in M. Elkin's algorithm and the asynchronous model in our problem, and no existing implementation.

Distributed Multi-constrained Shortest Path Algorithm

J.J. Garcia-Lunes-Aceves *et al.* extensively research on the problem of distributed loop-free routing [6], [13], [12]. In [12], they propose a distributed multi-constrained routing (DMR) protocol that uses distance vector to construct multiple feasible paths for each destination, moreover, it offers a customizable optimization function that can be defined based on a case-by-case basis. In addition, DMR supports multi-path selection and guarantees loop-free. DMR is a distributed Bellman-Ford (DBF) based algorithm where each node exchanges the logical shortest path(s) with their direct peers, then computes and updates the local shortest path(s) by applying the Bellman-Ford equation. The time and communication complexity of DMR was evaluated by comparing it with typical distance-vector and link-state routing protocols that are widely used in Internet.

1.4 Our Contributions

Our major contributions are listed as following:

1. We propose a fully distributed SFC request allocation algorithm, including computation of routing paths and reservation of resources, that operates in real-time dynamic networks. Each node within the network only communicates with its connected peers and then computes and decides independently. To the best of our knowledge today, there is very limited research discussing the distributed SFC provisioning problem.

2. For each SFC request, the path exploration algorithm computes the shortest path from

source to destination while satisfying all proposed constraints, including compute resources constraints such as RAM, CPU and storage, sequence constraint of VNFs in SFC, routing constraints such as bandwidth and E2E delay requirements.

3. We discuss in detail the candidate path selection and different influencing factors, we present and further analyze the overall performance loss when we collect different numbers of candidate paths for each request.

4. We conducted a series of experiments to closely simulate different network scenarios, where the traffic are affected by a set of factors, including the population of the city and the geographical distance between the source and destination cities, peak hours and concurrent requests, etc. The performance evaluation of our proposed algorithm shows that the algorithm can obtain a very close overall performance compared with the optimal baseline, regardless of the traffic fluctuations and concurrent requests.

1.5 Plan of the Thesis

The thesis is organized as follows: Chapter 1 describes the general background, including 5G slicing, SFC paradigm and the high level description of the research project on distributed SFC provisioning. Chapter 2 details the research problem and proposed a 2-phase scheme for SFC provisioning in form of a submitted manuscript. Chapter 3 concludes the research and future work.

Chapter 2

Distributed Provisioning for 5G Service Requests

The chapter is submitted to the IEEE Access for review, titled "Distributed Provisioning for 5G Service Requests", written by H. Duong, B. Jaumard and Y. Rao.

2.1 Abstract

Thanks to Software-Defined Networks (SDN) and Network Function Virtualization (NFV), 5G Networks can flexibly utilize networking and computing resources. Particularly, virtualized network functions (VNFs) are freely defined as long as general-purpose data centers (DC) can host them. In consequence, NFV offers network operators with diverse service structures, i.e., a high number of VNFs with linear or non-linear combinations to form a service structure. To ease and automate the service provisioning in 5G Networks with highly complex service structures, the Service Function Chaining (SFC) paradigm is introduced. SFC paradigm provisions a service request in 5G Networks takes into account bandwidth requirements and Quality of Service (QoS) constraints as in traditional request

network provisioning, jointly identifies nodes along the routing path with the required compute resources for VNFs specified in the associated with a service request.

In addition, 5G and Beyond 5G Networks are planned to be highly dynamic, i.e., new network components and devices are constantly introduced into the networks to expand or upgrade the coverage and performance. Thus it required a plug-and-play mechanism to avoid expensive management and time-consuming deployment. Distributed operating systems are suitable for this plug-and-play perspective.

In this paper, we investigate the designing of an efficient distributed SFC request provisioning algorithm. For each request, our distributed algorithm consists of two phases: i) the exploration phase (EXPL) explores a set of candidate paths, then ii) a resource reservation (RESV) phase confirms necessary resources for one of the candidate paths.

We use several dynamic traffic data sets to experiment and compare the proposed distributed algorithm with a well-known centralized resource-constrained shortest. The experimental results show that the distributed algorithm reaches a similar acceptance rate as the centralized algorithm, with reasonable amount of message exchanges per request.

2.2 Introduction

5G is ushering in a new era of "Internet of Everything" and will penetrate into a diverse range of emerging industries, it is envisioned to provide ubiquitously connected, highly reliable and ultra-low latency services for a massive number of individuals and devices. The unprecedented increase in active users and network dynamics bring more challenges to existing network techniques and industrial practices. Further more, the unpredictable evolving directions of the future requires a more flexible and efficient way of service deployment. From this perspective, distributed systems exhibit higher scalability, where each member of the system computes and makes decisions independently from the changes in other members, and thus are generally more resilient to dynamic changes than centralized systems, especially in large-scale dynamic networks. On the other hand, in support of the fast-evolving 5G network, the NFV technology has transformed the traditional deployment of network functions on dedicated hardware into a more flexible, cost-efficient way by enabling the allocation of VNFs on commodity hardware. An SFC is created when a network service consists of a set of VNFs connected by virtual links following a specified execution order along with multiple requirements on computing resources, bandwidth and delays. Determining a feasible path on a physical network to allocate an SFC while satisfying all these requirements is complex and has proved to be NP-hard [10], while a carefully designed SFC provisioning strategy is of key importance to ensure overall network performance, especially for large-scaled dynamic 5G networks. Fig. 2 gives a simple example of such SFC provisioning where the candidate hosts for each type of VNF is marked in the same color as the VNF, the red line paves a feasible path for the given SFC request.

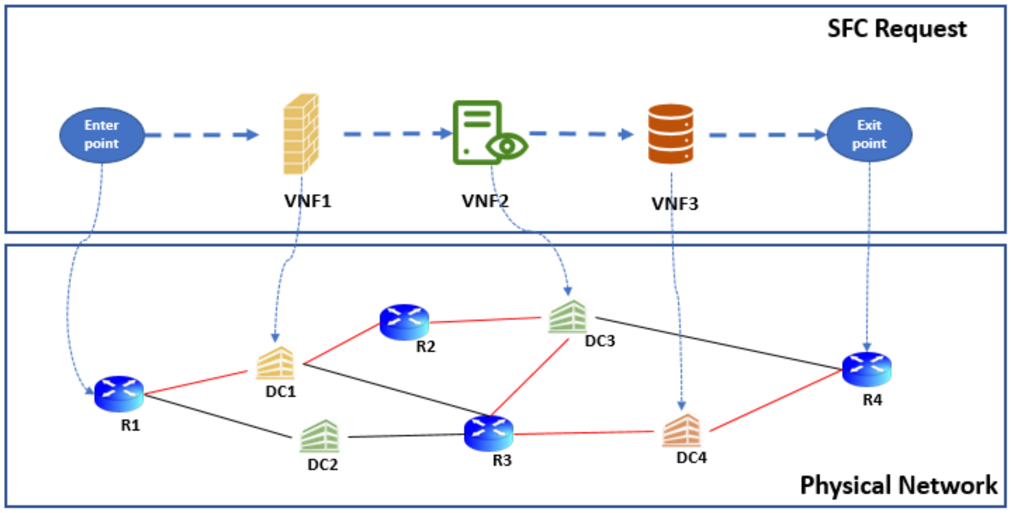


Figure 2: Example SFC request allocation.

Centralized provisioning of SFC requests takes the entire network as input, then outputs the globally optimal routing path for each request according to the performance metrics defined by the network operator. However, centralized provisioning has several drawbacks in certain cases, for example, the path computation can be very burdensome for the central controller when the network topology is large, and synchronization can be very expensive when the network is highly-dynamic. On the other hand, the high adaptability of distributed systems to large dynamic networks and to future changes motivates us to consider distributed provisioning for SFC requests.

We present a 2-phase distributed SFC provisioning (2P-DSP) algorithm aiming at computing and deciding the best-fit path for an SFC in a distributed network subject to its resource, bandwidth and delay constraints. The main idea of our algorithm is to employ the layered graph paradigm (e.g., in [3]) to convert the complex SFC allocation problem into a simpler routing problem on a multi-layered graph, then solve the simplified routing problem by applying a loop-free multi-constrained shortest path algorithm as of [12].

We show that layered graph can be built on-fly in a distributed algorithm where nodes within the network do not require a centralized supervision. They only communicate with connected peers, process and decide locally, and therefore conserving the scalability and adaptability of a distributed plug-and-play system. The allocation of an SFC has two phases: path exploration and resource reservation, the former one explores feasible paths and decide the best fit candidate, then the latter reserves networking and computing resources along the path. Simulation was conducted on OMNET++, and the results show that the distributed algorithm reaches a similar acceptance rate and throughput as of the centralized algorithm.

The rest of the paper is organized as follows. Section 2.3 surveys the related works including distributed shortest path algorithms and distributed SFC provisioning studies. Section 2.4 details the dynamic request provisioning problem. Section 2.5 demonstrates our distributed routing algorithm for each request as well as reservations. Section 2.6 presents the data generation process and experimental results of algorithms basing on the generated data sets. The last section concludes the research and discusses future work.

2.3 Literature Review

Numerous papers have been published on the provisioning of service requests in 5G networks, with very few of them discussing the distributed network requirements. It adds difficulty in terms of convergence and in terms of optimizing the number of message exchanges: a compromise needs to be found in terms of network resource requirements and quality of the network provisioning.

We therefore focus below on studies including the distributed system concern.

2.3.1 Distributed Shortest Path Algorithms

Firstly, we discuss the literature of distributed routing algorithm for a connection request with only bandwidth requirement. In other words, there are neither compute/storage resource constraints nor SFC specification.

M. Ghaffari *et al.* [7] abstracted the synchronous message-passing model to an undirected weighted graph where each vertical only has the information of its neighbors and the weights of adjacent links. they discussed previous work and proposed a new single-source shortest path(SSSP) algorithm with complexity Õ($n^{3/4}$ $D^{1/4}$) compared to previously Õ($n^{5/6}$), and complexity Õ($n^{3/4+o(1)}$ + min($n^{3/4}$ $D^{1/6}$,$n^{6/7}$)+D) compared to previous Õ($n^{5/6}$+$n^{2/3}$ $D^{1/3}$+D) for larger D value, where D is the hop-diameter of the graph. M.Elkin *et al.* [4] extensively discussed solving exact SSSP problem in sublinear time. That is $O((n \log n)^{5/6})$ time, for $D = O(\sqrt{n} \log n)$, and $O(D^{1/3} \times (n \log n)^{2/3})$ time, for larger D. U. Agarwal *et al.* [1] presented their algorithm for distributed weighted all pairs shortest paths (APSP) in both un-directed and directed graphs, which runs in Õ($n^{4/3}$) rounds in the Congest models on graphs with arbitrary edge weights, improved on the previous Õ($n^{3/2}$).

2.3.2 Distributed Multi-Constrained Shortest Path Algorithms

Secondly, we discuss distributed routing algorithms w.r.t. compute/storage resource constraints.

In [6], the author proposes a loop-free distance vector protocol to solve distributed routing problem. However, this work assumes that resources are unlimited. In [12], the authors extended the previous work in [6] by including resource constraints of paths. The authors present the distributed multi-constrained routing (DMR) protocol (algorithm) that finds a set of feasible paths from a given request that satisfy a set of link and path requirements, e.g., bandwidth, reliability, end-to-end delay. In addition, DMR protocol is able to use a non-closed form objective function for paths, by which feasible paths are ranked. The loop free property of the DMR protocol is guaranteed as long as the path cost function is monotonic and isotonic. However, this work does not provide a complete time complexity for DRM protocol. The performance of the DRM protocol is only evaluated by other well-known link-state routing protocols.

[18] proposes a distributed Optimized Multi Constrained Routing (OMCR) protocol using vector transform method and addressing effectively count-to-infinity issues. The OMCR protocol gains slightly better performance compared to the DMR protocol.

Lastly, to the best of our knowledge, there is limited work discussing distributed SFC provisioning problem. The algorithm in [11] can be considered as the state-of-the-art for distributed SFC provisioning algorithm. However, in [11], the authors does not take into account resource constraints (e.g., delay). Thus, the problem in this work is more complex than one of [11].

2.4 Problem Statement

The physical network is represented as a directed graph $G = (V, L)$ where V represents routers and DCs within the network connected by a set of directed links L, e.g., fiber links in an optical network. Nodes representing DCs are assigned with available VNF types such as firewall, IDS, VPN, etc., initial capacity of compute resources including CPU, RAM and storage as well as their current availability. Each link $\ell \in L$ has delay δ_ℓ, bandwidth capacity c_ℓ and available bandwidth a_ℓ such that total bandwidth on ℓ being reserved by SFC requests that passing through link ℓ will not exceed the physical limit c_ℓ, in other words, $a_\ell \geq 0$ at any time t. Table 1 lists out notations being used in our problem statement in detail.

The set F represents all available VNF types in the system, for each VNF $f \in F$, there are resource requirement rate in terms of CPU, RAM and storage, i.e., $(\text{RAM}_{kf}, \text{CPU}_{kf}, \text{STO}_{kf})$. Note that the compute resource requirements and the processing time of an VNF depend on the VNF types and on the bandwidth requirement of the request k, see, e.g., [19] for some examples numbers. Meanwhile, in this paper, we assume that one node can host multiple VNFs as long as it has enough resources available, while one VNF can only be hosted on the same node.

Let K be a set of requests, indexed by k. Each request k is characterized by:

- Its source and destination nodes, SRC_k and DST_k.
- Its arrival time T_k^{Arr} and its finishing time T_k^{End}.
- Its bandwidth requirement BW_k.
- Its E2E delay requirement δ_k including link delay and node delay. Link delay is the physical latency of a link, and node delay refers to processing time on a node, i.e., packet processing time of a VNF instance on its hosting node. In practice, node delay is real-time and it varies by hosting nodes and VNF types. In our simulations,

different types of VNFs are given different processing delays. Per bandwidth unit, all nodes share the same processing delay for the same VNF type.

- Its service chain c_k: an ordered sequence of n_k^c VNFs, i.e., $(f_1, f_2, \ldots, f_{n_k^c})$.

When a request arrives at the system, the operator must accept or reject the request; in other words, requests arrive in real time rather than being pre-scheduled, and the future requests are unknown. If the request is accepted, the operator must provide the request with routing information consisting of links and resource reservations.

We intent to design an fully-distributed algorithm that searches for the desired resources in the network under presented constraints to allocate requests subject to the objectives of

- Maximizing the the accepted requests and throughput overall,

- Minimizing the average message exchanging per request.

Table 1: Notations and vocabulary

Parameters	**Descriptions**
Physical Network	
$G = (V, L)$	Directed graph representing original network.
V_R, V_{DC}	Nodes representing routers and DCs, $V_R \bigcup V_{DC} = V$.
$C_v^{cpu}, C_v^{ram}, C_v^{sto}$	Resources capacity on DC $v \in V_{DC}$.
$A_v^{cpu}, A_v^{ram}, A_v^{sto}$	Resources availability on DC $v \in V_{DC}$.
F	Set of all possible VNFs in the system.
A_v^{VNF}	Set of available VNF types on DC $v \in V_{DC}$.
δ_v^f	Processing delay per unit of bandwidth for VNF f on hosting node v.
c_ℓ, a_ℓ	Bandwidth capacity and availability of link $\ell \in L, a_\ell \geq 0$.
δ_ℓ	Delay of link $\ell \in L$.
SFC Requests	
K, k	Request set and request index, $k \in K$.
SRC_k, DST_k	Arriving node and exiting node of request k.
$\text{T}_k^{Arr}, \text{T}_k^{End}$	Arrival time and exiting time of k.
δ_k, BW_k	Connection (E2E delay and bandwidth) requirements of k, E2E delay $\delta_k = \sum$ link delay $+ \sum$ node processing delay.
c_k	SFC of k, it specifies VNFs needed and their execution order.
$\text{RAM}_{kf}, \text{CPU}_{kf}, \text{STO}_{kf}$	Compute resources required by VNF $f \in c_k$

2.5 2P-DSP Algorithm

Distributed routing for SFC requests in 5G network can be processed in two schemes: The 1-phase scheme reserves resources in parallel with path exploration, while the 2-phase scheme computes candidate paths first and then decide which path to reserve. Since the 1-phase scheme will reserve resources for each candidate path, network resources can be exhaustively reserved easily in different situations. For example, when the number of requests or the number of candidates per request is large, or when the demand for connectivity or computational resources of individual requests is relatively high. In consequence, it inhibits requests to be routed efficiently. On the other hand, the 2-phase scheme does not run out of resources due to overbooking, but it may end up with no candidate paths that can be successfully reserved. In this case, a request must restart its search.

Considering that the number of requests in 5G networks can be enormous, we propose a 2-phase scheme for the distributed SFC routing problem, also named as 2P-DSP algorithm in this paper. The first phase explores and aims to obtain multiple feasible paths under network and compute resources constraints, after that the second phase will select and reserve a chosen path including link bandwidth, computing resources and VNFs.

On the other hand, given the fact that requests arrive in real time and future requests are unknown, it is impossible to pre-schedule resources for the requests, so we adopt a best-effort approach, i.e., we try to allocate every current request as much as possible, meanwhile we prioritize routing paths with fewer hops to save more link bandwidth for future requests.

2.5.1 Distributed Bellman-Ford-based Path Exploration Method

The exploration phase of the 2P-DSP algorithm is based on the idea of applying the distributed multi-constrained routing (DMR) algorithm in [12, 13] on a layered graph built from the physical network [3]. Take Fig. 3 as an example, to build a layered graph for request k, for each VNF f indexed $i(0 \leq i < n_k^c)$ by sequence number in SFC c_k, we abstract a copy labeled $Layer_i$ from the original network, i is layer number, for each candidate hosting node v for f, build a virtual link (v^i, v^{i+1}) connecting $Layer_i$ to $Layer_{i+1}$, v^i is the copy of v at i^{th} layer, such that eventually, for request k, the goal becomes to compute a shortest path from the abstract source node of 0th layer SRC_k^0 to the destination copy of the last layer $\text{DST}_k^{n_k^c}$ in the layered graph.

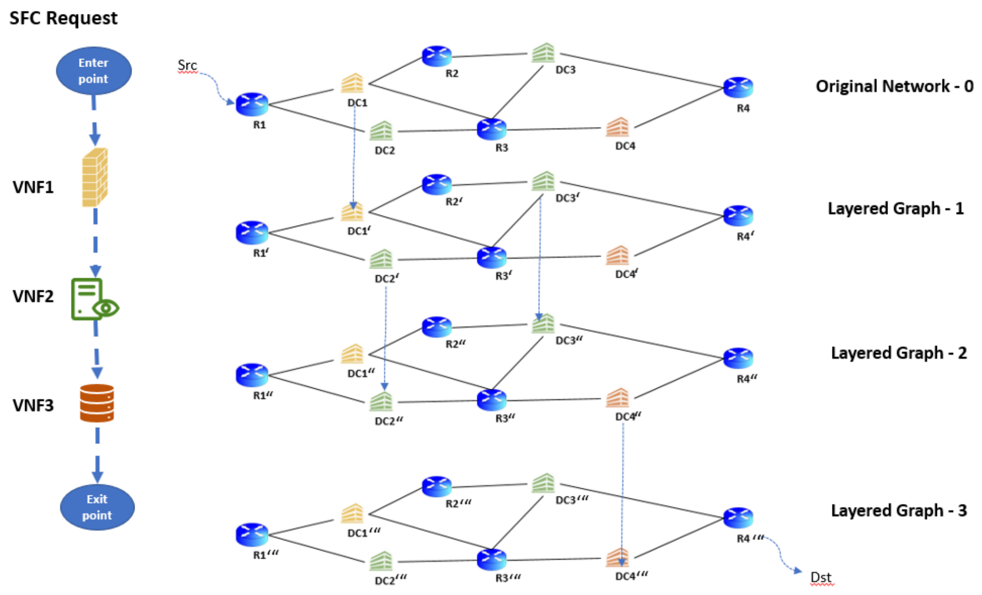


Figure 3: SFC request provisioning in layered graph.

We then employ DMR to compute the shortest paths from SRC_k^0 to $\text{DST}_k^{n_k^c}$ in the layered graph described above. Similar to the Distributed Bellman-Ford (DBF) algorithm, DMR exchanges distances only amongst directly connected peers, every node receives updates from its neighbouring nodes, then computes the latest shortest distance to each destination using Bellman-Ford equation and updates back to its neighbours. After certain rounds of network state updates, the shortest distance between any node pair converges and maintained locally. DMR converges in finite time and had been proved in [12]-Appendix B. Additionally, DMR defines optimization functions tailored to the practical situations and maintains a non-dominated path set for each destination to support multi-constrained path selection.

In the following description of the 2P-DSP algorithm, the notations being used are listed in Table 2.

Table 2: Notations in 2P-DSP

Parameters	**Descriptions**
Path Exploration Phase	
$G'_k = (V'_k, L'_k)$	Layered graph for request k built from original network $G = (V, L)$.
V^i, v^i	Copy of physical node set V at i^{th} layer, $v^i \in V^i$.
L^i, ℓ^i	Copy of physical link set L at i^{th} layer, $\ell^i \in L^i$, also labeled "horizontal links" in this work.
$\delta_{\ell^i}, A_{bw_{\ell^i}}$	Delay and available bandwidth of link $\ell^i \in L^i$, $\delta_{\ell^i} = \delta_\ell, A_{bw_{\ell^i}} = a_\ell$, where $\ell \in L$ is the original physical link.
(v^i, v^{i+1})	Virtual link that connecting v^i and v^{i+1} vertically between two layers, also labeled "vertical links" in this work.
$\delta_{(v^i,v^{i+1})}, A_{bw_{(v^i,v^{i+1})}}$	Delay and available bandwidth of link $\{(v^i, v^{i+1}) : v^i \in V^i, v^{i+1} \in V^{i+1}\}$, $\delta_{(v^i,v^{i+1})} = \delta_v^f \times \mathrm{BW}_k, A_{bw_{(v^i,v^{i+1})}} = \infty$.
$N^v_{\text{IN}}(t), N^v_{\text{OUT}}(t)$	In-coming and out-going neighbour set of node $v \in V'_k$
Resource Reservation Phase	
x	A constant number that indicates up to how many paths will be collected at path exploration phase.
W_k	Waiting time budget indicating up to how long it will wait upon the completion of first feasible path.
$Q_{forward}, Q_{backward}$	Two Queue data structures that used to record reservation status when reserving a path.

Constraint Management

To manage the VNF sequence constraint, for each request k, we define the so-called layered directed graph $G'_k = (V'_k, L'_k)$ where:

$$
\begin{aligned}
V'_k &= \bigcup_{i=0,1,2,\ldots,n^c_k} V^i \\
&\quad \text{where } V^i \text{ is a copy of } V. \\
L'_k &= \bigcup_{i=0,1,2,\ldots,n^c_k} L^i \cup \{(v^i, v^{i+1}) : v^i \in V^i, v^{i+1} \in V^{i+1}, \\
&\quad \text{where } L^i \text{ is a copy of } L. \\
&\quad v \text{ is a candidate node that has sufficient resources to host the } i\text{th VNF} \\
&\quad \text{denoted by } f, \text{ requesting resources } (\mathrm{RAM}_{kf}, \mathrm{CPU}_{kf}, \mathrm{STO}_{kf})\}.
\end{aligned} \tag{1}
$$

Note that v^i in above formulations is the abstract node of physical node $v \in V$ at i-th layer of layered graph G'_k. In other words, the layered graph is made of $n^c_k + 1$ copies of the original network, and copies of the network graph are connected by directed links from one compute node to the same compute node in the next layer, assuming node hosting the VNF has enough compute resources.

For horizontal link $\ell^i \in L^i$, its bandwidth and latency are the current bandwidth and latency of its original physical link $\ell \in L$. While for a vertical link (v^i, v^{i+1}) in Formulation 1, its delay is the processing time of VNF f on node v, i.e., $\delta_{(v^i,v^{i+1})} = \delta^f_v \times \mathrm{BW}_k$. Vertical link (v^i, v^{i+1}) has unlimited bandwidth.

Note that, in the layered graph G'_k, the neighbor sets of a node $v \in V'_k$ include links bridging two consecutive layers, namely the outgoing neighbor set

$$N^v_{\mathrm{OUT}}(t) = \{(v, v') : (v, v') \in L'_k \text{ and } \delta_{(v,v')} < \infty\},$$

and the incoming neighbor set

$$N^v_{\text{IN}}(t) = \{(v', v) : (v', v) \in L'_k \text{ and } \delta_{(v',v)} < \infty\}.$$

$\delta_{(v,v')} < \infty$ and $\delta_{(v',v)} < \infty$ verify that the link exists in the layer graph and is passable. Meanwhile, the neighbor set of an abstract node is not fixed as the physical links and nodes can be changed over time, therefore the neighbor sets are accommodated by a time index.

Connection constraints including E2E delay and bandwidth requirements are validated during path computation. A path p from node u to node v in layered graph G'_k is characterized by its total delay, total number of hops, set of link bandwidth usage, and link sequence, namely

$$p = \{(\sum_{(u,v)\in p} \delta_{(u,v)}, \#ofhops), \{(u, v) \in L'_k : (u, v) \in p\}\}. \quad (2)$$

To form a new path p' in the layered graph G'_k from path p by concatenating a new link $\ell \in L'_k$ to p, (e.g., operation $\oplus$ in Appendix A.2), the new path p' has to satisfy a set of constrains:

- Total delay of p' will not exceed the E2E delay requirement of request k:

$$\sum_{(u,v)\in p'} \delta_{(u,v)} \leq \delta_k.$$

- Every link $\ell \in L'_k$ that p' passes through must satisfy bandwidth requirement of request k:

$$A_{bw_\ell} > \text{BW}_k.$$

- The times h that path p' passes through the same physical link ℓ should satisfy that the total bandwidth consumed on link ℓ will not exceed its current bandwidth availability:

$$h \times \mathrm{BW}_k < a_\ell.$$

Path p dominates path q only when the total delay and hops of p is less than that of q. The distance of path p is defined as:

$$f^p = \text{\# of hops of } p \tag{3}$$

Stopping Conditions

If given long enough time for request k to search for feasible paths from SRC_k^0 to $\mathrm{DST}_k^{n_k^c}$ in layered graph G'_k described in previous paragraph using DMR, $\mathrm{DST}_k^{n_k^c}$ will either collect a set of non-dominated path(s) or none, before message exchanging over the network ends in finite time after the last network state update, given the proof in [12] Appendix B. On top of this, we added E2E delay and bandwidth constraints to limit timely live messages in the network, i.e., a path exploration message will stop propagating upon detecting current path exceeds E2E delay requirement or current stop does not provide a valid connecting link that meets bandwidth requirement. Meanwhile, at destination $\mathrm{DST}_k^{n_k^c}$, we set a constant number x and a time budget W_k to limit path exploration time detailed in Section 2.5.2.

Algorithm Specification

The pseudo-code of the path exploration algorithm are detailed in Appendix A.2. In our algorithm, the layered graph is constructed on-the-fly and updated dynamically. For request k, when it first arrives at node SRC_k, SRC_k will initiate a local record for abstract node of the SRC_k at the 0th layer, i.e., R1 in Fig. 3, and send UPDATE messages to all its outgoing neighbors, these UPDATE messages will later trigger their receivers to check and initiate an local record at the same horizontal layer for request k if there is none exists yet. The same initialization applies to every other nodes within the layer graph. A node will add a self copy of the next layer to its out-going neighbour set when it discovers itself as a candidate node of current layer, then it will send out an UPDATE message to this copy and trigger the construction of the next layer, and so on.

All layers are synchronized at anytime, this is easy to deliver because all the abstract copies at different layers of the same node are technically stored at the physical node, therefore the physical states including resources status, link delay and bandwidth are shared real-time.

2.5.2 Distributed Resource Reservation

Path Collection Strategies

A successful path exploration for request k will end up with a set of candidate paths coming in chronological order at destination node DST_k. In a distributed network with concurrent requests, trying multiple candidate paths in a single reservation attempt presents a number of challenges including synchronization and deadlock issues. To ensure the reservation status of the same request remains consistent all the time across the network, in each reservation attempt, we only proceed with one candidate path.

We keep multiple candidate paths in case the reservation of a current path fails. Meanwhile, given the fact that a best-fit path is a comprise between multiple constraints, the first arriving path does not always produce the best overall outputs, while collecting all feasible paths will take much longer time at the risk of losing current ones, we therefore set a constant number x to decide how many candidate paths will be collected and a waiting time budget W_k, W_k starts after DST_k receives the first feasible path p^k_{1st}, such that:

- DST_k can have enough time to collect up to x candidate paths,
- DST_k will not wait too long for perspective paths, especially when there are concurrent requests, e.g., in our simulation, we set W_k to be 1 time slot.

At DST_k, reservation phase starts either when it finishes collecting x paths or when the waiting time exceeds the budget W_k.

Distributed Path Reservation

In this work, paths with less hops are given priority on reservation decisions, then the total delay of the path will be used to break the tie. Reservation starts with selecting the path with least hops among all candidate paths, if reservation is successful, request k will be connected and remain online till T_k^{End}, otherwise, current path will be removed from candidates and the next one with least hops will be selected, and so on. Request k will be rejected when no more candidate path available.

Technically, the distributed reservation of a path is delivered mainly by two queue data structures, $Q_{forward}$ and $Q_{backward}$, that are updated and passed on through message exchanges at each node along the path. Starts from DST_k, each path being selected for reservation will be first decoded from the virtual path in the layered graph to a hop-by-hop path in the physical network, and then stored in queue $Q_{forward}$ which indicates the path ahead to be reserved, each node receives the reservation message will reserve link bandwidth and/or compute resources on nodes indicated in $Q_{forward}$, then pop out the record of itself from $Q_{forward}$ and push to $Q_{backward}$, $Q_{backward}$ records current progress, i.e., bandwidth on links and compute resources on nodes that have been reserved so far. When there is a reservation failure at an intermediate node, this node will first abort reservation and then initiate an cancellation procedure along $Q_{backward}$ to cancel previous reservations on links and nodes, until it travels back to DST_k and select next path to start another reservation attempt.

2.6 Numerical Results

In this section, we discuss the evaluation of our algorithm in four sub-sections: Section 2.6.1 explains experimental setups including configuration of the simulation environment, generating scheme of data sets, and design of the comparison algorithm. Section 2.6.2 compares the performance of our distributed algorithm 2P-DSP with the classical centralized Resource Constrained Shortest Path (RCSP) algorithm w.r.t throughput and acceptance rate. Section 2.6.3 discusses the number of message exchanges during EXPL phase and RESV phase. Section 2.6.4 compares different path collection strategies, we name each path collection strategy by "P" and the x value defined in Section 2.5.2, e.g., "P3" indicates that EXPL will collect first 3 paths before deciding which path to reserve.

2.6.1 Experimental Setups

Simulation Environment and Data Set

We implemented the 2P-DSP algorithm on OMNET++ v 5.6.2[16], a discrete-event simulator with powerful and complete GUI interfaces that has been widely used in network simulation for both academic and industrial purposes, simulations are conducted on a computer with Intel(R) Xeon(R) CPU E3-1271 v3 3.60GHz and 32 GB RAM. The network was built based on the CORONET Continental United States (CONUS) topology[21], from which we removed 2-degree nodes to simply the topology, meanwhile, 10 DCs were selected manually so that all regions on the map are covered, as shown in Fig. 4, there are 4 types of predefined VNFs that will be randomly placed on the 10 DCs, each DC has $1 \sim 4$ available VNF types. During each simulation, initial capacity of compute resources on each DC, bandwidth of each link are calculated corresponding with the traffic load of each data set, detailed in the following paragraph.

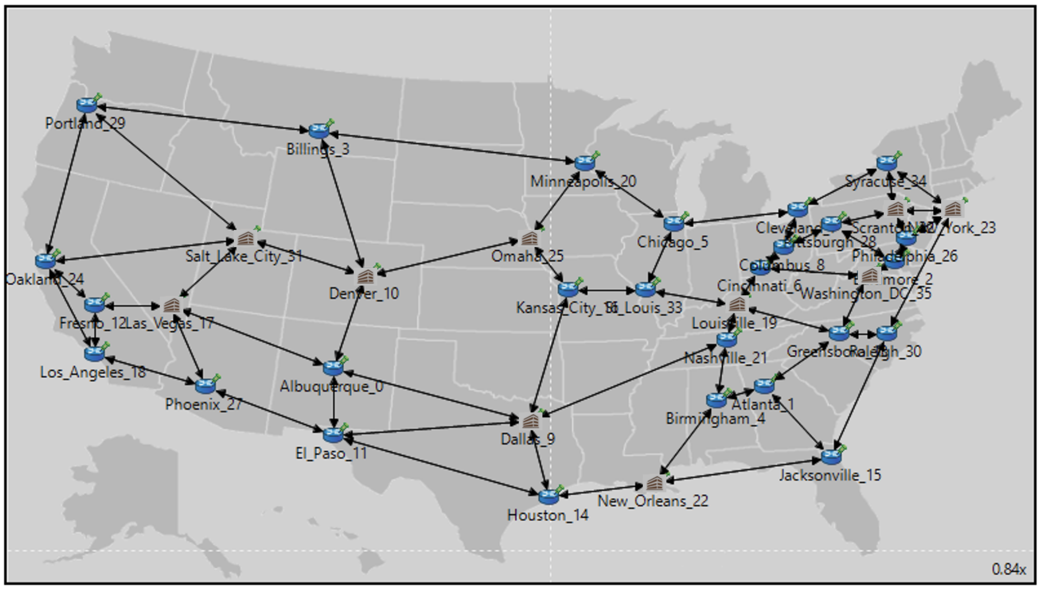


Figure 4: Simulation network topology(36 nodes, 120 links, built on OMNET++).

To simulate the real-world network scenarios as close as possible, each data set is generated in 3 steps:

- First, we create an unbounded network using the same topology while assigning unlimited computing and networking resources to all DCs and links, such that this network can meet any size of demand.

- Next, we generate randomized SFC requests between all node pairs in the unbounded network. For "Uniformed" traffic, the distribution of such node pairs, or source and destination of the request, will be the same amongst all pairs. On the other hand, for "Non-uniformed" traffic, the distribution will be calculated based on the population and geometric distance of two cities, similar to [2]. That is, assuming the total traffic load of the whole network is K, the traffic load from citie i to city j is

$$K \times \frac{R_{ij}}{\sum R},$$
$$\text{where } R_{ij} = \left(\log_{10}\left(\frac{P_i \times P_j}{d_{ij}} + 10\right) - 1\right),$$
$$\sum R \text{ is the sum of the values of all node pairs by applying the formula of } R_{ij}. \tag{4}$$

where P_i and P_j are the populations of the two cities, the population data is obtained from Wikipedia[24], taken for the year 2020. d_{ij} is the geographical distance between this two cities. The $\log$ function is used to avoid this impractical unbalanced distribution caused by high-population cities. If we do not use the $\log$ function, then the distribution is really unbalanced, i.e., two biggest city attract all the requests as show in Data 8 and 9 of Fig. 7.

Let T be the duration of a simulation, represented by the number of time slots, each time slot is equivalent to one second. At each time slot, there can be one or several requests arrives at the start of the time slot. At this stage, the E2E delay requirements of all requests are set to infinite, bandwidth requirements range evenly between 1 $\sim$ 10 Gbps. SFC of each request is randomly picked from 10 pre-defined SFCs, the pre-defined SFCs are constructed out of the 4 types of VNFs randomly. The live time of the SFC requests is a geometric random variable with the average of 2000 time slots. Each SFC request will be routed along a random path among a set of paths computed by the k-shortest-path algorithm (k=10) on the layered graph w.r.t. E2E delay. During this process, the maximum usage of compute and connect resources on each DC and link will be recorded, denoted by C, meanwhile, the E2E delay of the routing path for each request will be recorded and denoted by D.

- Finally, we use the resource usage and request data collected in the previous step to estimate and control the resource capacity and traffic in the simulation environment. Every SFC request generated previously will be updated with a new E2E delay requirement based on its associated D, i.e.,

$$\text{E2E Delay requirement of request } k = D_k \times d\%. \tag{5}$$

the updated request set will then be shuffled and prepared for the experiments. Meanwhile, the previously recorded maximum usage C on each node/link will be used to estimate its resource capacities respectively in the experimenting network, i.e.,

$$\text{Link/Node Capacity} = C \times Random(a\%, b\%). \tag{6}$$

In Table 3, we list out all the parameters of above settings, including variables a, b and d, for each data set. Intuitively, if the request set is uniformly reordered on the network with $a = b = d = 100\%$, then one can expect that routing algorithms will get an acceptance rate close to 100%.

Additionally, we introduce fluctuations to the traffic to simulate peaks and dips of the traffic at different times. To do that, we introduce 1000-time-slot fluctuation periods. There are two type of periods alternatively happen. The first one is "peak" period where we randomly add more connection requests to these time slots, while during a "dip" period, some connection requests will be selected randomly and their live times will be shortened. In Fig. 5, it presents an example of a original offered load ('stable') and the derived offered load after the original one is reodered and fluctuated by the above process.

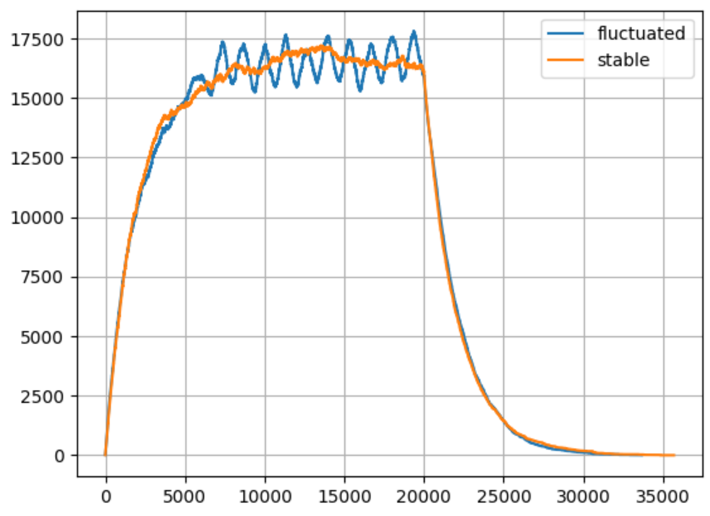


Figure 5: Original and fluctuated offered load examples

Note that traffic, reflected from throughput, during a simulation is divided into 3 stage as shown in Fig. 6: the "Warming-up" stage starts with an empty network where most new requests are embraced by sufficient resources and therefore has an acceptance rate close to 100%, "Stable" state is when resources being reserved and released by current requests in the network reached dynamic balance, which is of the most interest to observe the performance of request provisioning. At "Cool-down" stage, there's no incoming new requests, while existing requests start to release resources and eventually exit the network. In the results being presented, we will mainly focus on performance of the algorithms at "Stable" stage.

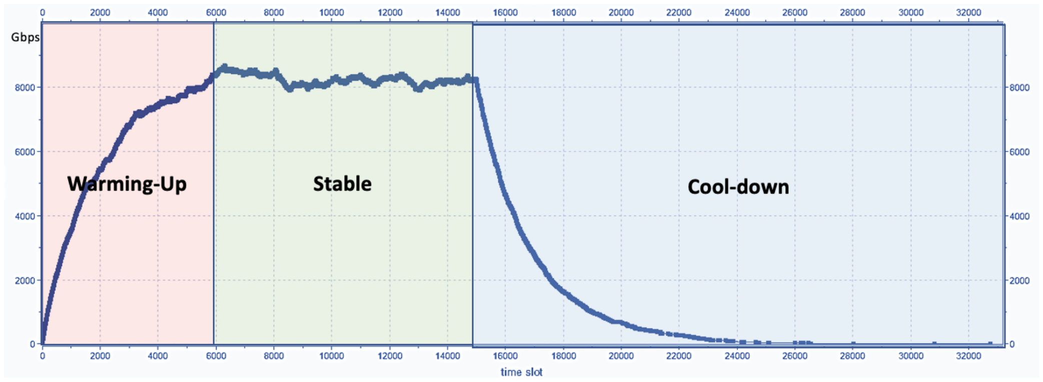


Figure 6: Example of throughput over time during simulation.

The "Stable" state of all the data sets being experimented in following sections starts from the 6000th time slot.

Table 3: Characteristics of data sets

Data-set No.	Uniformed (Y/N)	Fluctuated (Y/N)	Total Req. / Time Slots	Max. Req. per Time Slot	Computing Scaling	Bandwidth Scaling	Delay Scaling
7	Y	N	15000/ 15000	1	[-10%, +20%]	[-10%, +20%]	20%
8	N	N	15000/ 15000	1	[-10%, +20%]	[-10%, +20%]	20%
9	N	Y	22095/ 20000	2	[-10%, +20%]	[-10%, +20%]	20%
13	N	Y	22100/ 20000	2	RAM: [-10%,+10%] CPU,STO:[-10%,+20%]	[-10%, +20%]	20% if hops > 4, [20%,30%] otherwise.
15	N	Y	22100/ 20000	2	RAM:[-10%,+10%] CPU,STO:[-10%,+20%]	[-10%, +20%]	20% if hops > 4, [20%,30%] otherwise.
16	N	Y	33054/ 20000	2 [30%]	RAM: [-10%, +10%] CPU,STO:[-10%,+20%]	[-10%, +20%]	20% if hops > 4, [20%,30%] otherwise.
17	N	Y	33025/ 20000	2 [30%]	RAM: [-10%, 0%] CPU,STO:[-10%, +5%]	[-20%, 0%]	20% if hops > 4, [20%,30%] otherwise.
18	N	Y	34061/ 20000	2 [40%]	RAM: [-10%, 0%] CPU,STO:[-10%, +5%]	[-20%, 0%]	20% if hops > 4, [20%,30%] otherwise.

Equation 6 is applied to Computing Scaling and Bandwidth Scaling, Equation 5 is applied to Delay Scaling.

Baseline Algorithm

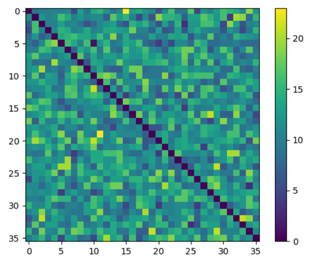
For comparison, we use a centralized resource-constrained shortest path algorithm, i.e., algorithm RCSP in [22]. RCSP assumes that communication between the centralized controller and nodes within the network is instantaneous without cost or delay. Similar to 2P-DSP, RCSP defines the length of a path by hop count and uses E2E latency to break the tie. The difference is that for each request, RCSP instantly obtains the globally shortest feasible path and immediately reserves the path. For 2P-DSP, on the other hand, message exchanging obeys link delay, every node asynchronously communicates with its peers and calculates the current shortest path based on locally available information. For a single request, if the 2P-DSP is given a long enough time to collect all feasible paths, the 2P-DSP will theoretically end up with the same shortest path as the RCSP, but this becomes uncertain in dynamic networks where there may be more requests competing for the same resources and time becomes critical. Therefore, in our simulations, the RCSP is primarily designed to provide an baseline to evaluate 2P-DSP for real-time dynamic network scenarios.

Data Analysis

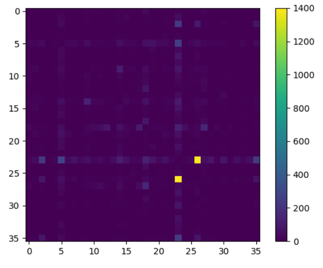
On the other hand, the behavior of the RCSP can provide deeper insights into the characteristics of each data set. First Fig. 7 illustrates the total number of requests (also called traffic load) between node pairs within the network in the whole process. As mentioned earlier in Equation 4, in data set 8 and 9, two cities attract all the traffic when we do not apply the log function, while the other data sets are more reasonable, which also shows the validity of our distribution function.

Second, Fig. 8 summarizes resources usage during a simulation as well as the throughput rate. We can see that, in these data sets, the usage of computing resources is higher than the usage of bandwidth, which coincides with the real-life situation that network computing resources are generally more scarce than bandwidth.

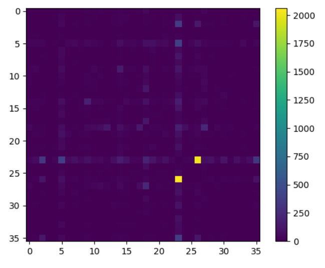
Last, in Fig. 9, we define "Delay Stress Level" to be the percentage of the latency of the RCSP chosen path for a request divided by the allowable latency or E2E delay requirement of that request, such that, from the "Delay Stress Level" histogram, we can roughly estimate the latency budget for each request and the ease to collect multiple feasible paths, i.e., as shown in the histogram, the majority of the requests in each data set generally use up $80 \sim 90\%$ of the given delay budget, indicating that the given allowed latency settings are not easy for path exploration and, at the same time, they are reasonable.



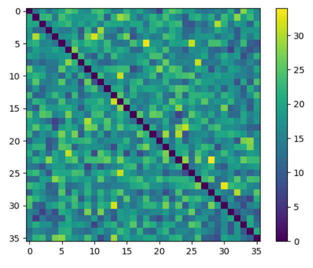
(a) Dataset-7



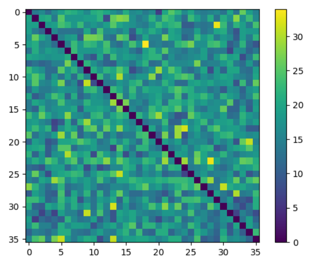
(b) Dataset-8



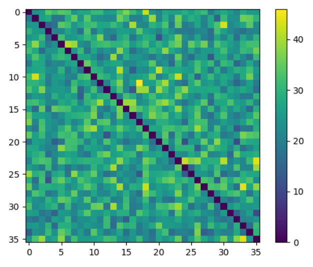
(c) Dataset-9



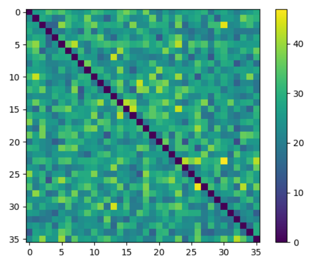
(d) Dataset-13



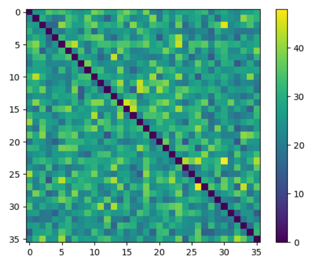
(e) Dataset-15



(f) Dataset-16

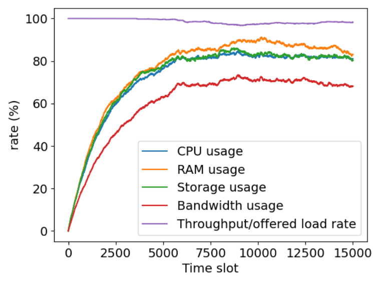


(g) Dataset-17

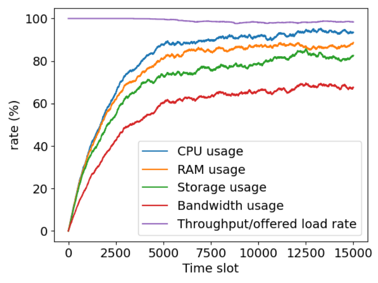


(h) Dataset-18

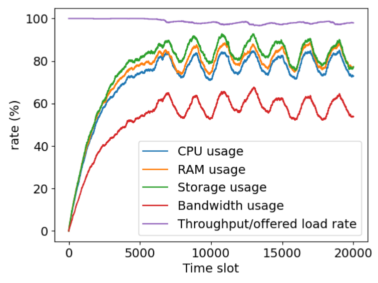
Figure 7: Traffic distribution heat map



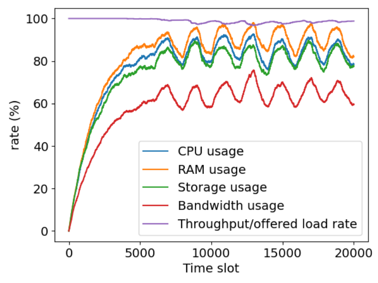
(a) Dataset-7



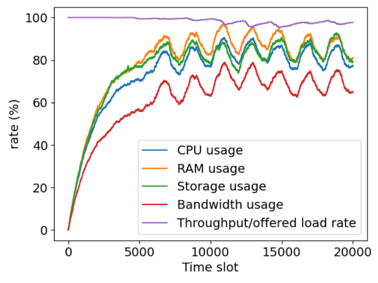
(b) Dataset-8



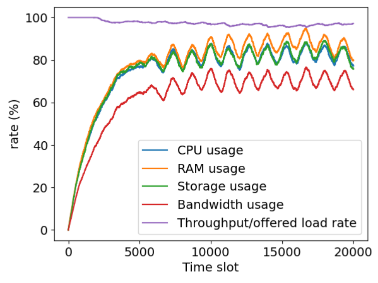
(c) Dataset-9



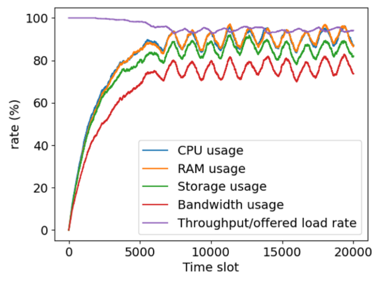
(d) Dataset-13



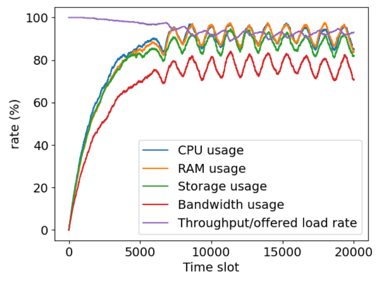
(e) Dataset-15



(f) Dataset-16

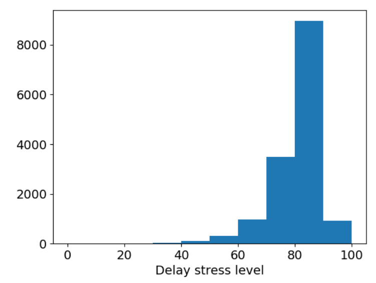


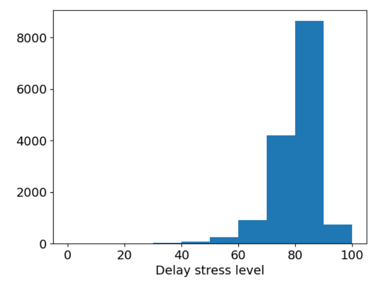
(g) Dataset-17

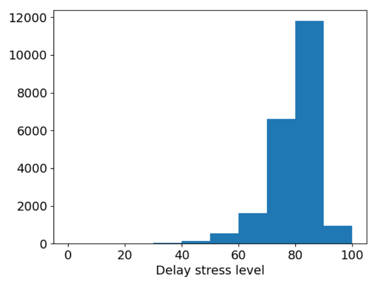


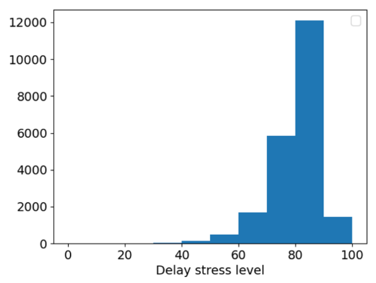
(h) Dataset-18

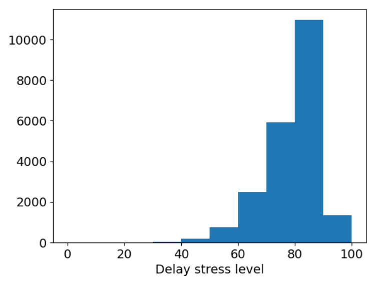
Figure 8: Resources utilization and throughput of RCSP

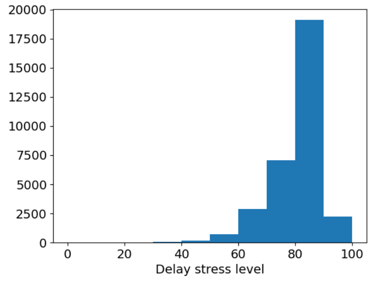

(a) Dataset-7

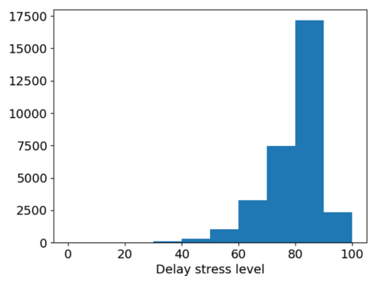

(b) Dataset-8


(c) Dataset-9


(d) Dataset-13


(e) Dataset-15


(f) Dataset-16


(g) Dataset-17

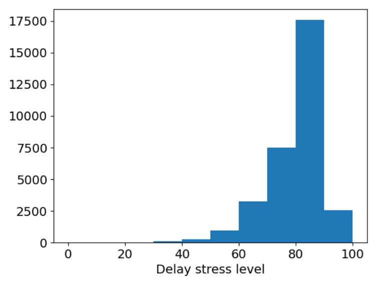

(h) Dataset-18

Figure 9: Delay stress level of RCSP

2.6.2 RCSP vs. 2P-DSP

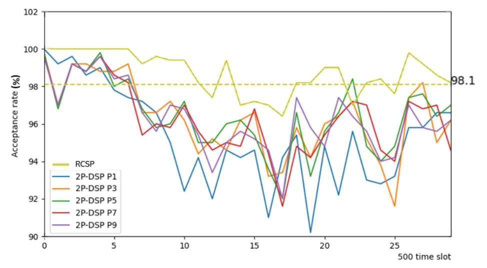
In Table 4 and Fig. 10, we evaluation RCSP and 2P-DSP w.r.t acceptance rate, in Fig. 11 $\sim$ 18, we compare the performance of the two algorithms w.r.t throughput.

Table 4 presents the overall acceptance rate of the two algorithm at stable state, we can see that, with appropriate path collection strategy which are highlighted in gray color, 2P-DSP obtains an overall acceptance rate that is very close to that of RCSP in all data sets, with the difference of 0.59% $\sim$ 4.34%.

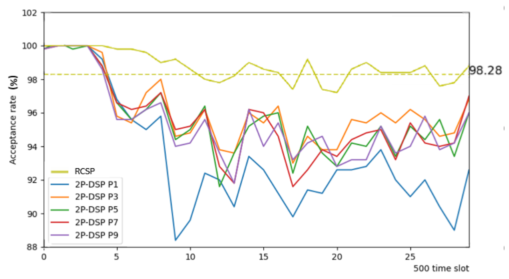
Table 4: Overall acceptance rate at stable stage (Time slot from 6000 to 20000)

Dataset No.	RCSP	2P-DSP				
		P1	P3	P5	P7	P9
7	98.10%	94.03%	95.34%	95.59%	95.36%	95.33%
8	98.28%	91.71%	95.04%	94.31%	94.17%	94.15%
9	97.7%	94.91%	96.22%	95.99%	96.09%	95.97%
13	97.91%	95.18%	95.78%	95.47%	95.39%	95.20%
15	97.68%	96.73%	96.14%	96.09%	96.02%	96.07%
16	97.20%	96.61%	96.05%	95.45%	95.19%	95.06%
17	94.84%	89.15%	90.50%	90.42%	90.46%	90.30%
18	93.37%	91.78%	92.77%	92.30%	92.09%	92.21%

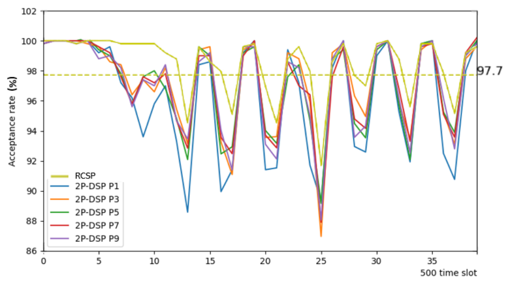
In Fig. 10, we plot acceptance rate by time window of 500 time-slots, consistent with Table 4, we observe that the plots of 2P-DSP are very close to RCSP in Dataset-15 and Dataset-16, and the two almost overlap in Dataset-18. In Dataset-7,8,9 and 13, the acceptance rate of 2P-DSP stayed within 5% of that of RCSP in the best case. In Dataset-17, the gap between 2P-DSP and RCSP gets relatively bigger, reaching a maximum of around 10% at best level for 2P-DSP.



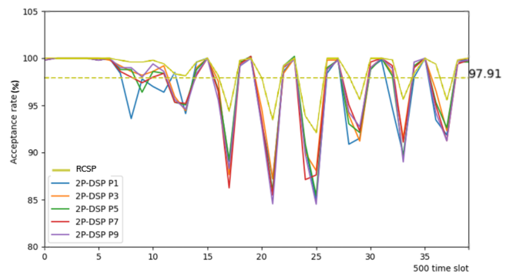
(a) Dataset-7



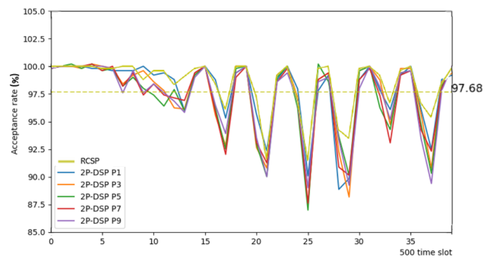
(b) Dataset-8



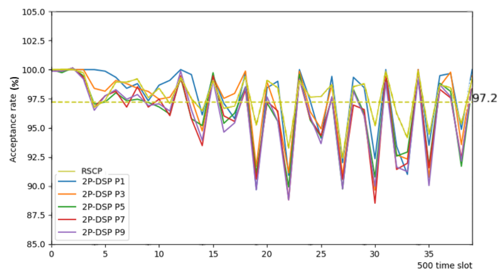
(c) Dataset-9



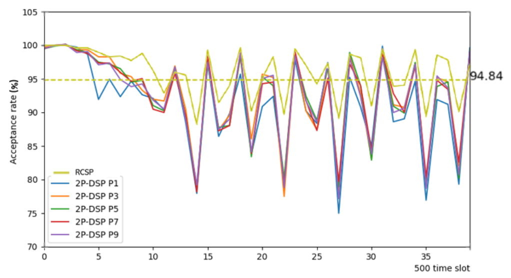
(d) Dataset-13



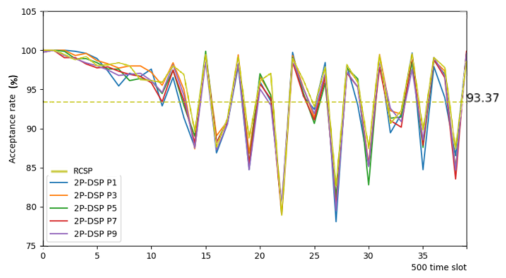
(e) Dataset-15



(f) Dataset-16

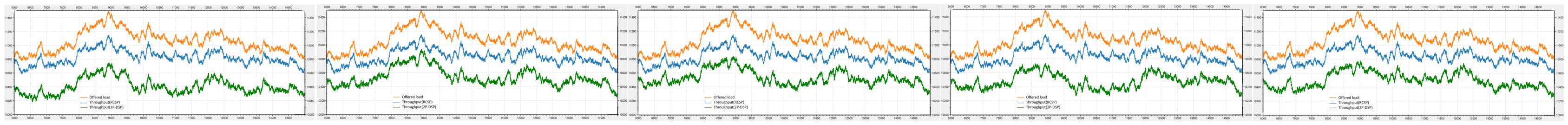


(g) Dataset-17



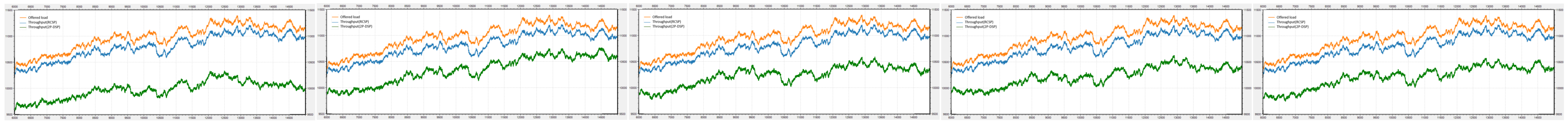
(h) Dataset-18

Figure 10: Acceptance rate by time window 500 time-slots



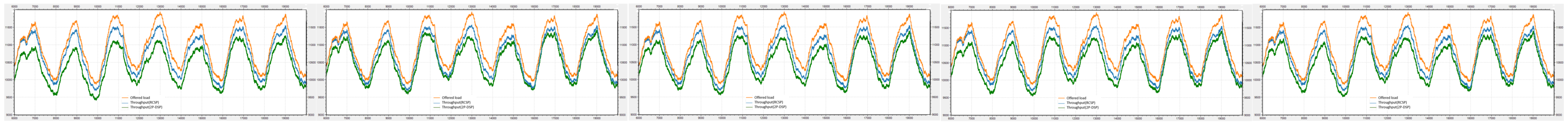
(a) P1 (b) P3 (c) P5 (d) P7 (e) P9

Figure 11: 2P-DSP vs. RCSP w.r.t throughput when choosing different path collection strategies(Dataset-7)



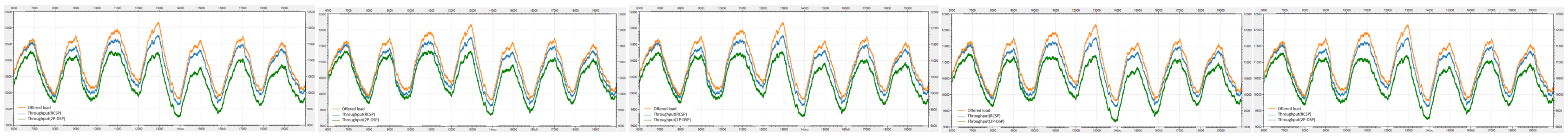
(a) P1 (b) P3 (c) P5 (d) P7 (e) P9

Figure 12: 2P-DSP vs. RCSP w.r.t throughput when choosing different path collection strategies(Dataset-8)



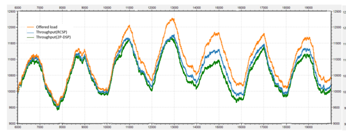
(a) P1 (b) P3 (c) P5 (d) P7 (e) P9

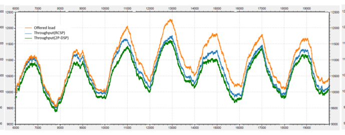
Figure 13: 2P-DSP vs. RCSP w.r.t throughput when choosing different path collection strategies(Dataset-9)

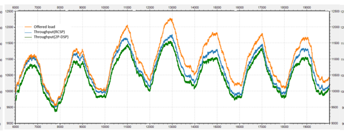


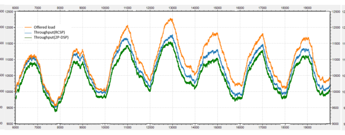
(a) P1 (b) P3 (c) P5 (d) P7 (e) P9

Figure 14: 2P-DSP vs. RCSP w.r.t throughput when choosing different path collection strategies(Dataset-13)


(a) P1


(b) P3


(c) P5


(d) P7

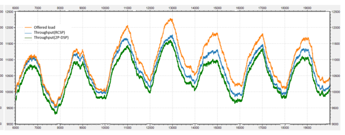
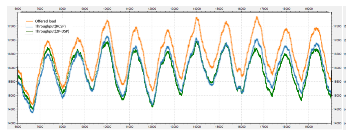
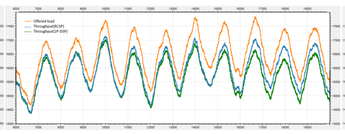
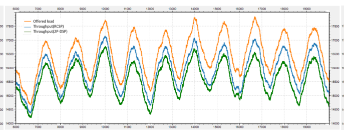
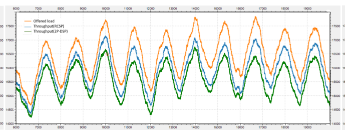

(e) P9

Figure 15: 2P-DSP vs. RCSP w.r.t throughput when choosing different path collection strategies(Dataset-15)


(a) P1


(b) P3


(c) P5


(d) P7

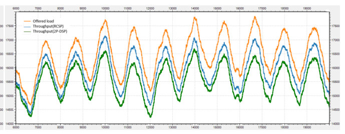
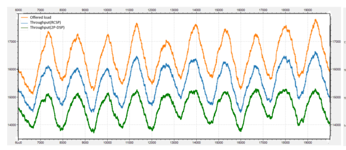
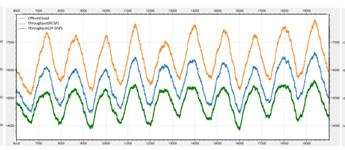
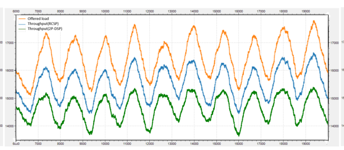
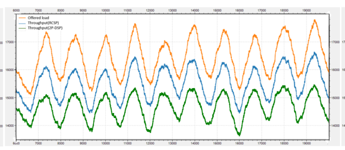

(e) P9

Figure 16: 2P-DSP vs. RCSP w.r.t throughput when choosing different path collection strategies(Dataset-16)


(a) P1


(b) P3


(c) P5


(d) P7

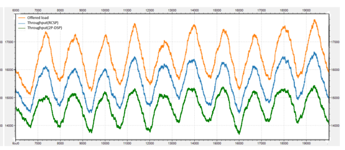
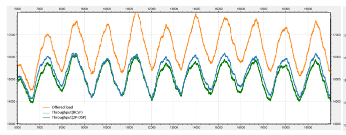
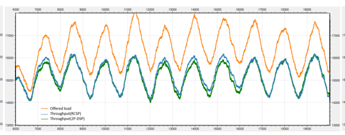
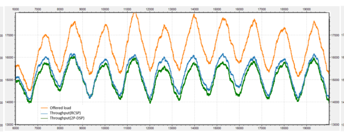
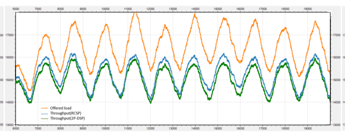

(e) P9

Figure 17: 2P-DSP vs. RCSP w.r.t throughput when choosing different path collection strategies(Dataset-17)


(a) P1


(b) P3


(c) P5


(d) P7

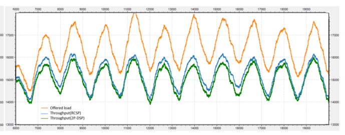

(e) P9

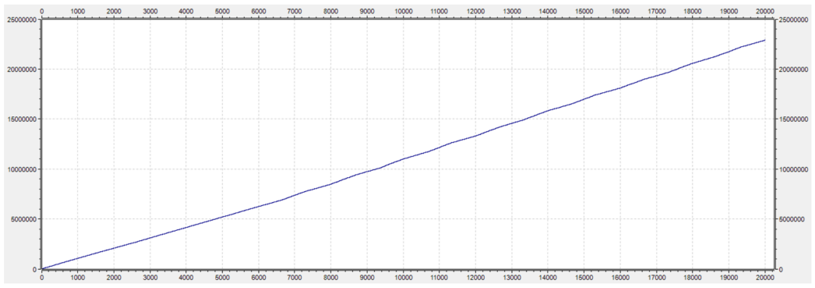
Figure 18: 2P-DSP vs. RCSP w.r.t throughput when choosing different path collection strategies(Dataset-18)

Fig. 11 $\sim$ 18 compares 2P-DSP with RCSP in terms of throughput. In accordance with their performance as of acceptance rate, at the best level of performance, 2P-DSP still approaches RCSP closely regardless of the peaks in traffic in all the data sets except Dataset-17. On the other hand, in Dataset-16,17 and 18, we introduced more concurrent requests at stable state, from where we can see that 2P-DSP is handling well without any observable loss in performance.

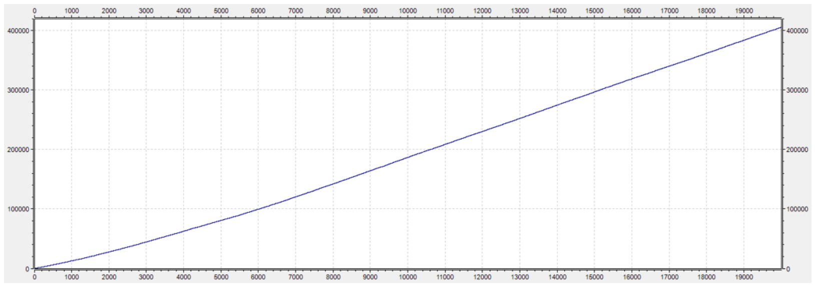
2.6.3 Message Exchanging

We evaluate the complexity of 2P-DSP in terms of the amount of messages exchanged, i.e., the average number of messages generated to serve a request.

Our simulations on OMNET++ show that, in all the data sets, the number of messages generated grows almost linearly with time, take Fig. 19 as an example.



(a) EXPL Message Growth



(b) RESV Message Growth

Figure 19: Messages generated during a simulation (Dataset-17 P1)

Table 5 lists out average message volume per request for each data set.

Table 5: Avg. message exchanging per request during 2 phases with different path collection strategies

Data-set No.	P1		P3		P5		P7		P9	
	EXPL	RESV	EXPL	RESV	EXPL	RESV	EXPL	RESV	EXPL	RESV
7	742.24	12.94	685.56	14.52	673.46	14.25	671.94	14.20	672.29	14.14
8	680.23	10.63	603.83	12.62	593.99	12.26	593.26	12.14	594.03	12.28
9	690.25	10.73	609.77	12.44	600.24	12.02	597.16	11.89	597.48	11.87
13	701.59	12.75	650.38	14.29	639.85	13.97	637.22	13.81	638.02	13.82
15	677.65	13.45	646.83	14.74	640.57	14.36	638.42	14.22	638.01	14.26
16	705.73	12.92	640.02	14.26	624.55	13.85	622.53	13.78	623.29	13.77
17	692.97	12.73	623.72	14.90	608.18	14.65	606.68	14.55	605.49	14.52
18	690.07	12.53	616.14	14.12	599.92	13.70	599.52	13.62	600.21	13.59

We observe that:

- Under the current environment settings, the average number of EXPL messages generated to serve a request in "Uniformed" traffic is 671 $\sim$ 742 as shown in Dataset-7, for "Non-uniformed" traffic is 593 $\sim$ 706. This is because, after applying the traffic equation, two nodes that are geometrically closer have higher probability to receive more connection requests, in other words, "Uniformed" traffic receives more requests between two long-distant cities.
- This EXPL message amount is in the same order of magnitude of the number of links in the network. Thus, it shows that the distributed algorithm only requires reasonable number of messages for routing algorithm.
- The average number of RESV messages is 10 $\sim$ 15 per request. It is in the same order with the width of the network, it implies that the majority of the requests are granted within 1 $\sim$ 2 reservation attempts.

2.6.4 Path Collection Strategies

To compare different path collection strategies, we do a cross-sectional analysis in terms of acceptance rate, throughput and message exchanging from the previous tables and graphs.

In Table 4, 2P-DSP performed at the best level w.r.t overall acceptance rate when P3 is selected as the path collection strategy, followed by P5 and P7. P1 yields the lowest acceptance rate in all data sets except Dataset-15 and Dataset-16, mainly because there is no chance of trial and error in selecting P1, and once the current path fails, the request will be rejected. While in Dataset-15 and 16, not only P1, we can see that all other path collection strategies obtained very similar and high acceptance rate, which indicates that most requests obtained the optimal path from the first few candidate paths collected, meanwhile, with sufficient resources, most requests are able to be granted, in this case, choosing P1 saves time instead of waiting to collect multiple paths.

The performance of different path collection strategies varies depending on the network and traffic settings, however, the best path collection strategy is a compromise between multiple factors, including the quality of candidate paths and the time to collect these paths. In our simulations, the main reason why P3 outperforms other path collection strategies is that, for most requests in these data sets, they can get the shortest path from the first few candidate paths, i.e., 3 to 5. Compared with P5, P7, and P9, P3 saves more waiting time at the expense of choosing longer paths for a small number of requests, which we can roughly infer from the acceptance rates in Table 4 and the average number of reservation messages in Table 5. To this end, We see that these path collection strategies are very close to each other except for P1.

In terms of message exchanging, given the nature of distributed system and DBF algorithm, message exchanging during the path exploration phase will not vary much using different path collection strategies, because for other nodes than the DST, they won't be notified if DST has collected enough paths, they will only follow their routine of processing and updating to peers whenever receives a message.

In our implementation, the propagation of EXPL messages is limited by two constraints: the bandwidth requirement of the request and its E2E delay requirement. From this perspective, bandwidth availability of the links within the network can have a impact on message propagation during path exploration. For example, 2P-DSP generates more EXPL messages when choosing P1, mainly because P1 maintains lower acceptance rate overall, intuitively, with less requests being granted, there are more bandwidth available during a simulation, EXPL messages can propagate further.

The number of generated RESV messages is mainly affected by the hop count of the chosen path and number of reservation attempts. We can see that P1 has the least RESV message exchanges overall, mainly because P1 only has one booking attempt, without cancelling or re-booking processes, while on the other hand, it can also explain why the other path collection strategies takes relatively more RESV messages. From P3 to P9, we observe a slight decrease in RESV message, this is because, while getting more candidate paths, 2P-DSP gets higher probability to find path with less hops.

2.7 Conclusion

In this work, we study the distributed SFC provisioning problem in the context of 5G networks, and then propose a novel fully distributed 2-phase SFC request allocation scheme, aiming at maximizing the total number of granted requests while minimizing the communication cost. For routing path computation, we transform the original complex routing problem by running a DBF-based shortest path algorithm on a layered graph. For path reservation, we discuss the selection of candidate paths and compare different path collection strategies. From our simulations, we observe that in most of the cases, our 2P-DSP algorithm obtains very close performance in terms of both acceptance rate and throughput compared to the optimal baseline RCSP, and that 2P-DSP still performs well during traffic peaks and when concurrent requests are introduced. In addition, the message load generated during the simulation is within the reasonable range.

Acknowledgment

This work is supported by a joint Mitacs-Ciena (SOF project) grant.

Chapter 3

Conclusion and Future Work

In this thesis, we investigate the problem of SFC provisioning in distributed 5G networks, the proposed solution and the results are detailed in Chapter 2.

We demonstrate that the layered graph can be constructed on-the-fly in a distributed dynamic system, and the integration of the layered graph paradigm and the multi-constraint DBF-based shortest path algorithm can well manage the provisioning constraints of the SFC requests. The layered graph ensures that the routing paths follow the compute resources constraint and the sequence constraint of VNFs in SFC. Meanwhile, routing constraints, including bandwidth and E2E delay requirements, are checked during path formation processes. Additionally, we show that the message exchanging ends in finite time.

For resource reservation, we discuss the selection of candidate paths and compare different path collection strategies, i.e., in a dynamic network with limited resources, time is critical and the optimal path collection strategy is a compromise under the combination of various factors, including the time budget for collecting multiple paths and the quality of these paths. For example, we see an obvious performance degradation when we only collect the first complete path as the candidate, because there is no chance for trial and error. While on the other hand, collecting too many candidate paths, say 9, can also lead to an overall performance loss, as the collected paths could have their resources taken up by other requests due to long waiting time.

Future research can extend the current work in several directions, including, e.g.,:

- introducing larger network topology and more concurrent requests. In all our simulations in Chapter 2, we used the same network topology with 36 nodes and 120 links, and the maximum number concurrent requests per time slot is 2. It would be interesting to test with a larger topology, say 75 nodes, and a larger number of concurrent requests, say at least $10 \sim 20$.

- investigating more complex path definitions, i.e., take resource defragmentation into account so that those paths with lower degree of resource fragmentation are prioritized, thus allowing the system to save resources to fit in more requests with smaller resource requirements and ultimately maximize the number of granted requests.

- extending current 5G SFC provisioning problem to partial-order SFC provisioning, i.e., SFC structures are DAGs defining precedence constraints.

Bibliography

[1] U. Agarwal and V. Ramachandran. Faster deterministic all pairs shortest paths in congest model. In *32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 11 – 21, 2020.

[2] A. Betker, C. Gerlach, R. Hülsermann, M. Jäger, M. Barry, S. Bodamer, J. Späth, C. Gauger, and M. Köhn. Reference transport network scenarios. *MultiTeraNet Report*, 2003.

[3] A. Dwaraki and T. Wolf. Adaptive service-chain routing for virtual network functions in software-defined networks. In *Workshop on Hot topics in Middleboxes and Network Function Virtualization (HotMIddlebox)*, pages 32–37, 2016.

[4] M. Elkin. Distributed exact shortest paths in sublinear time. *J. ACM*, 67(3), May 2020.

[5] Ericsson. This is 5G. `https://www.ericsson.com/49f1c9/assets/local/5g/documents/07052021-ericsson-this-is-5g.pdf`, 2021. (accessed: 2021-12-17).

[6] J. J. Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM transactions on networking*, 1(1):130–141, 1993.

[7] M. Ghaffari and J. Li. Improved distributed algorithms for exact shortest paths. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2018, page 431–444, New York, NY, USA, 2018. Association for Computing Machinery.

[8] GSA. 5G network slicing for vertical industries. `https://www-file.huawei.com/-/media/corporate/pdf/news/gsa-5g-network-slicing-for-vertical-industries.pdf?la=zh`, 2017. (accessed: 2021-12-17).

[9] K. Kaur, V. Mangat, and K. Kumar. A comprehensive survey of service function chain provisioning approaches in sdn and nfv architecture. *Computer Science Review*, 38:100298, 2020.

[10] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai. Deploying chains of virtual network functions: On the relation between link and server usage. *IEEE/ACM Transactions on Networking*, 26(4):1562–1576, 2018.

[11] D. Li, J. Lan, and Y. Hu. Central control over distributed service function path. *KSII Transactions on Internet and Information Systems (TIIS)*, 14(2):577–594, 2020.

[12] Z. Li and J. Garcia-Luna-Aceves. Loop-free constrained path computation for hop-by-hop qos routing. In *Computer Networks*, volume 51, pages 3278–3293, 2007.

[13] Z. Li and J. J. Garcia-Luna-Aceves. A distributed approach for multi-constrained path selection and routing optimization. In *Proceedings of the 3rd International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks*, QShine '06, page 36–es, New York, NY, USA, 2006. Association for Computing Machinery.

[14] L. Liu, S. Guo, G. Liu, and Y. Yang. Joint dynamical vnf placement and sfc routing in nfv-enabled sdns. *IEEE Transactions on Network and Service Management*, 18(4):4263–4276, 2021.

[15] K. Mahdi. A self-optimizing fabric for the 5G era. `https://media.ciena.com/documents/A_Self-Optimizing_Fabric_for_the_5G_Era_WP.pdf`, 2020. (accessed: 2020-11-11).

[16] OMNeT++. OMNeT++ discrete event simulator. `https://omnetpp.org/`. (accessed: 12.03.2021).

[17] J. Pei, P. Hong, K. Xue, and D. Li. Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system. *IEEE Transactions on Parallel and Distributed Systems*, 30(10):2179–2192, 2019.

[18] P. S. Prakash and S. Selvan. Optimized multi constrained path quality of service routing protocol. *WSEAS Trans. Info. Sci. and App.*, 8(2):80–95, Feb. 2011.

[19] S. V. Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester. Profile-based resource allocation for virtualized network functions. *IEEE Transactions on Network and Service Management*, 16:1374 – 1388, Dec. 2019.

[20] G. L. Santos, D. d. F. Bezerra, É. d. S. Rocha, L. Ferreira, A. L. C. Moreira, G. E. Gonçalves, M. V. Marquezini, Á. Recse, A. Mehta, J. Kelner, et al. Service function chain placement in distributed scenarios: A systematic review. *Journal of Network and Systems Management*, 30(1):1–39, 2022.

[21] J. Simmons. CORONET continental united states (CONUS) topology. `http://www.monarchna.com/topology.html`. (accessed: 20.04.2021).

[22] T. D. Tran, B. Jaumard, H. Duong, and K.-K. Nguyen. Joint service function chain embedding and routing in cloud-based nfv: A deep q-learning based approach. In *2021 IEEE 4th 5G World Forum (5GWF)*, pages 171–175, 2021.

[23] S. Wijethilaka and M. Liyanage. Survey on network slicing for internet of things realization in 5g networks. *IEEE Communications Surveys Tutorials*, 23(2):957–994, 2021.

[24] Wikipedia. List of United States cities by population. `https://en.wikipedia.org/wiki/List_of_United_States_cities_by_population`. (accessed: 04.10.2021).

Appendix A

Algorithm Specification

The pseudo code is mainly taken from [12] with some further refinements. For the ease of reading, the major function in [12] has been divided into several sub-functions.

A.1 Notation

Note that, in the notation list, s refers to the source node in the layered graph of request k, i.e., SRC_k^0 at 0-th layer.

For each abstract node $v \in V_k'$ in the layered graph, it maintains the following information:

- $S_v^{s,k}(t)$: the successor set chosen by node v for s according to the loop-free condition at time t, and $S_v^{s,k}(t) \subset N_{IN}^v(t)$.
- $QS_v^{s,k}(t)$: the set of neighbors from whom a query has been received, but a reply has not been sent.
- $state_v^k(t)$: state of node v for the processing of request k at time t.
- $LD_v^{s,k}(t)$: the shortest logical distance from node s to node v that is known by node v at time t for request k.

- $Pset_v^s$: the non-dominated path set from s to v maintained (known) by node v at time t.

- $SLD_v^s(t)$: the shortest logical distance from s to v through $S_v^s(t)$ at time t.

- $RLD_v^s(t)$: the logical distance from s to v that node v reports to all its neighbors at time t.

- $FLD_v^{s,k}(t)$: the feasible logical distance used by node v to evaluate whether $o-LFC$ (i.e., loop-free condition) can be satisfied when choosing $S_v^{s,k}$.

- $\widetilde{LD}_{v,n}^{s,k}(t)$: the shortest logical distance from node s to node n that is maintained (known) by node v at time t for request k, where $n \in N_{\text{IN}}^v(t)$.

- $\widetilde{PSet}_{v,n}^{s}(t)$: the non-dominated path set from s to n – a neighbor of v, that is maintained (known) by v at time t.

- $r_{v,n}^s(t)$: this flag is true if node v has sent a query for s to neighbor n but has not received a reply from n, and false otherwise.

- $lc_u^v(t)$: cost when using link (u, v) at time t. For the time being, the cost to use a link at anytime is the delay of the link. When a link is removed or available amount of CPU/RAM/DISK of a node is no longer enough for the corresponding VNF, then its corresponding is the layered graph is set to ∞.

A.2 Pseudo-code

Algorithm 1 Initialize information for source SRC_k^0 at node v

```
1: procedure INITIALIZE_NODE(v, k)
2:     $LD_v^{\text{SRC}_k^0,k} \leftarrow (v = \text{SRC}_k^0 ? 0 : \infty)$
3:     $SLD_v^{\text{SRC}_k^0,k} \leftarrow (v = \text{SRC}_k^0 ? 0 : \infty)$
4:     $FLD_v^{\text{SRC}_k^0,k} \leftarrow (v = \text{SRC}_k^0 ? 0 : \infty)$
5:     $RLD_v^{\text{SRC}_k^0,k} \leftarrow (v = \text{SRC}_k^0 ? 0 : \infty)$
6:     $S_v^{\text{SRC}_k^0,k} \leftarrow \emptyset$
7:     $QS_v^{\text{SRC}_k^0,k} \leftarrow \emptyset$
8:     $state_v^{\text{SRC}_k^0,k} \leftarrow PASSIVE$
9:     if $v = \text{SRC}_k^0$ then
10:        for each $n \in N_{\text{OUT}}^v$ do
11:            send $\{UPDATE, v, \bar{0}, < null >, k\}$ to $n$    ▷ $n.PROCESS\_VECTOR(...)$
12:        end for
13:    else
14:        for each $n \in N_{\text{IN}}^v$ do
15:            $\widetilde{LD}_{v,n}^{\text{SRC}_k^0,k} \leftarrow \infty$
16:            $r_{v,n}^{\text{SRC}_k^0,k} \leftarrow false$
17:        end for
18:    end if
19: end procedure
```

When the abstract node of the source node receives a request or when an abstract node gets involved with the request for the first time, the abstract node initializes a local record in its database using Algorithm 1. In other words, only node SRC_k^0 starts the process by sending an $UPDATE$ message to every its outgoing neighbor. In [12], INITIALIZE_NODE procedure, line 10-13, they are indented without any block indications. However, based on their algorithm's line 12, it indicates that only destination node sends out $UPDATE$ messages at the beginning.

Algorithm 2 Node v receives information of source SRC_k^0 from its neighbor n.

```
1: procedure PROCESS_VECTOR($mt, n, rld, < d^p, w^p >, k$)
2:     $s \leftarrow \mathrm{SRC}_k^0$
3:     if $s = v$ then
4:         if $mt = QUERY$ then
5:             send $\{REPLY, v, \bar{0}, < null >, k\}$ to $n$
6:         else
7:             return
8:         end if
9:     end if
10:    if $mt = REPLY$ then
11:        $r_{v,n}^{s,k} \leftarrow false$
12:    end if
13:    $\widetilde{PSet}_{v,n}^{s,k} \leftarrow \widetilde{PSet}_{v,n}^{s,k} \uplus p \triangleright p =< d^p, w^p >$ ▷ non-dominated updating operation $\uplus$
14:    $\widetilde{LD}_{v,n}^{s,k} \leftarrow rld$                ▷ $rld$ is always more up-to-date than $\tilde{LD}_{v,n}^{s,k}$
15:    update_successor_set($v, k$)
16:    $LD_v^{s,k} \leftarrow \min\{\widetilde{LD}_{v,n}^{s,k} \oplus lc_v^n | n \in N_{\text{IN}}^v\}$
17:    $w_{LD_v^{s,k}} \leftarrow$ the path weight associated with $LD_v^{s,k}$
18:    if $state_v^{s,k} = PASSIVE$ then
19:        process_passive_mode($mt, n, rld, < d^p, w^p >, k$)
20:    else
21:        process_active_mode($mt, n, rld, < d^p, w^p >, k$)
22:    end if
23: end procedure
```

Whenever an abstract node v receives a message of any types from its neighbor n, that message is processed by Algorithm 2. To ease the reading, s is used as an alias of the abstract node representing the source node of the request, i.e., $s \leftarrow \mathrm{SRC}_k^0$ in this algorithm.

Firstly, let's consider when v is the source node (SRC_k^0). If its neighbor n is querying for the logical distance then it simply replies back an distance of 0 and an empty path. If a message is not a $QUERY$ message, then it is an $UPDATE$ message. Since v is the source node, it does not need to update its distance in any cases.

Next, if the message's type is $REPLY$ then v simply marks that n has replied to its query.

Line 13-14 of Algorithm 2, v updates its known information about n ($\widetilde{LD}_{v,n}^{s,k}$ and $\widetilde{PSet}_{v,n}^{s,k}$) according to the message's information. In Line 13, $\uplus$ means that the set add p if it is not dominated by any path in the set and remove paths that are dominated by p. In Line

In Algorithm 2-line 16, v recompute its known shortest logical distance ($LD_v^{s,k}$) using current conditions of its incoming links.

- mt: message type.
- n: the sender of this message.
- rld: logical distance from s to n that node n reports to its neighbors.
- $< d^p, w^p >$: a path and its logical distance and delay.
- k: connection request which is considered.

Algorithm 3 Update successor set for source SRC_k^0 of node v

```
1: procedure UPDATE_SUCCESSOR_SET(v, k)
2:     $s \leftarrow \mathrm{SRC}_k^0$
3:     $S_v^{s,k} \leftarrow \{n \in N_{\mathrm{IN}}^v : \widetilde{LD}_{v,n}^{s,k} \prec FLD_v^{s,k}\}$
4:     for each $n \in S_v^{s,k}$ do
5:         for each $p \in \widetilde{PSet}_{v,n}^{s,k}$ do
6:             $p \leftarrow p \oplus lc_v^n$
7:             $PSet_s^{v,k} \leftarrow PSet_s^{v,k} \uplus p$                ▷ non-dominated paths
8:         end for
9:     end for
10:    $SLD_v^{s,k} \leftarrow \min\{\widetilde{LD}_{v,n}^{s,k} \oplus lc_v^n : n \in S_v^{s,k}\}$      ▷ $SLD_v^{s,k} \leftarrow \infty$ if $S_v^{s,k} = \emptyset$
11:    $w_{SLD_v^{s,k}} \leftarrow$ the path associated with $SLD_v^{s,k}$
12:    for each $n \in N_{\mathrm{IN}}^v \setminus S_v^{s,k}$ do
13:        $\widetilde{PSet}_{v,n}^{s,k} \leftarrow \emptyset$
14:    end for
15: end procedure
```

Using Algorithm 3, an abstract node v evaluates its successor set according to its current incoming-link conditions.

Algorithm 4 Node v process a vector in passive mode

```
1: procedure PROCESS_PASSIVE_MODE($mt, n, rld, < d^p, w^p >, k$)
2:     $s \leftarrow \mathrm{SRC}_k^0$
3:     if $LD_v^{s,k} \prec SLD_v^{s,k}$ or $S_v^{s,k} = \emptyset$ then        ▷ $S_v^{s,k}$ does not provide optimal path
4:         $state_v^{s,k} \leftarrow ACTIVE$        ▷ become active
5:         if $mt = QUERY$ then
6:             $QS_v^{s,k} \leftarrow QS_v^{s,k} \cup n$
7:         end if
8:         $RLD_v^{s,k} \leftarrow SLD_v^{s,k}$
9:         for each $h \in N_{\mathrm{OUT}}^{v,k}$ do
10:            $r_{v,h}^{s,k} \leftarrow true$
11:            send $\{QUERY, v, RLD_v^{s,k}, < SLD_v^{s,k}, w_{SLD_v^{s,k}} >, k\}$ to $h$
12:        end for
13:    else        ▷ stay in passive
14:        $state_v^{s,k} \leftarrow PASSIVE$
15:        $FLD_v^{s,k} \leftarrow \min\{LD_v^{s,k}, RLD_v^{s,k}\}$        ▷ $FLD$ may decrease
16:        update_successor_set(v,k)        ▷ update according to new $FLD$
17:        $flag \leftarrow (RLD_v^{s,k} \neq LD_v^{s,k} ? true : false)$
18:        $RLD_v^{s,k} \leftarrow LD_v^{s,k}$
19:        if $mt = QUERY$ then
20:            send $\{REPLY, v, RLD_v^{s,k}, < LD_v^{s,k}, w_{LD_v^{s,k}} >, k\}$ to $n$
21:        end if
22:        if $flag = true$ then
23:            for each $h \in N_{\mathrm{OUT}}^{v,k}$ do
24:                send $\{UPDATE, v, RLD_v^{s,k}, < LD_v^{s,k}, w_{LD_v^{s,k}} >, k\}$ to $h$
25:            end for
26:        end if
27:    end if
28: end procedure
```

When an abstract node v is in a passive state ($state_v^{s,k} = PASSIVE$), there are two cases. In the first case, the successor set of v at the current time can provide the currently known shortest logical distance ($LD_v^{s,k}$). In this case, node v first update its feasible logical distance to maintain loop free condition (Source Node Condition (SNC) [6]) (Line 15, Algorithm 4). In consequence, its successor set is updated according to its new feasible logical distance.

Since current $LD_v^{s,k}$ value is the most up-to-date value, it is used as the value to report to neighbors of v. Note that if a neighbor n is querying v (i.e., asking v to update its successor set according to the current successor set of n), then v must reply. Otherwise, v only sends notifications to its neighbor if the reporting value ($RLD_v^{s,k}(t)$) is different from the last one before t.

In the second case, the current successor set cannot provide the shortest logical distance considering all of its incoming neighbors (incoming links towards v). In this case, node v becomes active to notify nodes upon v that they must update their successor set because successor set of v is going to be modified. In this case, v must fist set its state to active ($state_v^{s,k} \leftarrow ACTIVE$). Then it sends $QUERY$ messages to its outgoing neighbors (outgoing links of v) to ask them to correct their successor sets under condition that v has not changed its successor set.

Algorithm 5 Node v process vector in active mode

```
1:  procedure PROCESS_IN_ACTIVE_MODE($mt, n, rld, < d^p, w^p >, k$)
2:      $s \leftarrow \mathrm{SRC}_k^0$
3:      if $r_{v,h}^{s,k} = false, \forall h \in N_{OUT}^v$ then                ▷ receive all replies
4:          $FLD_v^{s,k} \leftarrow \min\{LD_v^{s,k}, RLD_v^{s,k}\}$    ▷ $FLD$ can increase only at the end of an
    active phase
5:          update_successor_set($k$)
6:          if $LD_v^{s,k} \prec SLD_v^{s,k} \vee (S_v^{s,k} = \emptyset \wedge RLD_v^{s,k} \prec \bar{\infty})$ then
7:              $state_v^{s,k} \leftarrow ACTIVE$                    ▷ start a new active phase
8:              if $mt = QUERY$ then
9:                  $QS_v^{s,k} \leftarrow QS_v^{s,k} \cup n$
10:             end if
11:             $RLD_v^{s,k} \leftarrow SLD_v^{s,k}$
12:             for each $h \in N_{OUT}^v$ do
13:                 $r_{v,h}^{s,k} \leftarrow true$
14:                 send $\{QUERY, v, RLD_v^{s,k}, < SLD_v^{s,k}, w_{SLD_v^{s,k}} >, k\}$ to $h$
15:             end for
16:         else
17:             $state_v^{s,k} \leftarrow PASSIVE$                   ▷ return to passive
18:             $flag \leftarrow (RLD_v^{s,k} \neq LD_v^{s,k} ? true : false)$
19:             $RLD_v^{s,k} \leftarrow LD_v^{s,k}$
20:             for each $h \in N_{OUT}^v$ do
21:                 if $h \in QS_v^{s,k} \vee (h = n \wedge mt = QUERY)$ then
22:                     send $\{REPLY, v, RLD_v^{s,k}, < LD_v^{s,k}, w_{LD_v^{s,k}} >, k\}$ to $h$
23:                 else if $flag = true$ then
24:                     send $\{UPDATE, v, RLD_v^{s,k}, < LD_v^{s,k}, w_{LD_v^{s,k}} >, k\}$ to $h$
25:                 end if
26:             end for
27:             $QS_v^{s,k} \leftarrow \emptyset$
28:         end if
```

When a node v in the active state, it mean that v is waiting for replies from its outgoing neighbors ($REPLY$ messages). If v is aware that its does not have all required replies, then it only considers query messages sent to it. Note that when the shortest logical distance of v current $SLD_v^{s,k}$ is larger than its value used to report $RLD_v^{s,k}$, it means that its shortest distance via its successor set has increased. Then, in that case, v puts n to its waiting list.