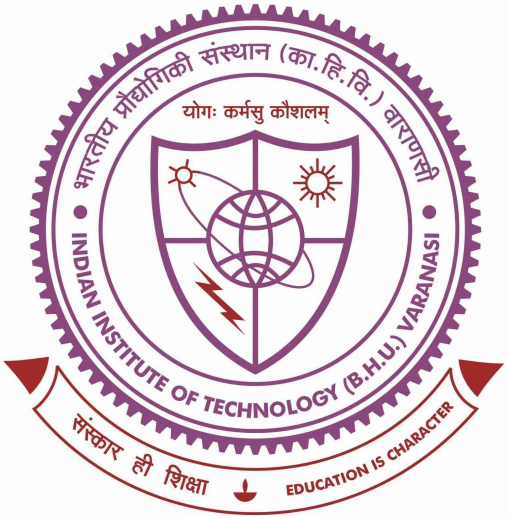
Cost-Aware and Resilient Placement of Parallel Service Function Chains Using Multi-Stage Graph Modeling



Thesis submitted in partial fulfilment

for the Award of

INTEGRATED DUAL DEGREE (B.TECH.+ M.TECH.)

in

COMPUTER SCIENCE

by

HIMANSHU YADAV

DEPARTMENT OF COMPUTER SCIENCE

INDIAN INSTITUTE OF TECHNOLOGY

(BANARAS HINDU UNIVERSITY)

VARANASI – 221 005

ROLL NUMBER
20074016

YEAR OF SUBMISSION
2025

CERTIFICATE

It is certified that the work contained in the thesis titled **Cost-Aware and Resilient Placement of Parallel Service Function Chains Using Multi-Stage Graph Modeling** by **HIMANSHU YADAV** has been carried out under my/our supervision and that this work has not been submitted elsewhere for a degree.

It is further certified that the student has fulfilled all the requirements of Comprehensive Examination, Candidacy and SOTA for the award of **Integrated Dual Degree (B.Tech.+ M.Tech.)**.

Supervisor
Prof. Dr. Vignesh Sivaraman
DEPARTMENT OF COMPUTER SCIENCE
Indian Institute Of Technology
(Banaras Hindu University)
Varanasi – 221 005

DECLARATION BY THE CANDIDATE

I, **HIMANSHU YADAV**, certify that the work embodied in this thesis is my own bona fide work and carried out by me under the supervision of **Prof. Dr. Vignesh Sivaraman** from **January 2024** to **May 2025**, at the **DEPARTMENT OF COMPUTER SCIENCE**, Indian Institute of Technology (Banaras Hindu University), Varanasi. The matter embodied in this thesis has not been submitted for the award of any other degree/diploma. I declare that I have faithfully acknowledged and given credits to the research workers wherever their works have been cited in my work in this thesis. I further declare that I have not willfully copied any other's work, paragraphs, text, data, results, *etc.*, reported in journals, books, magazines, reports dissertations, theses, *etc.*, or available at websites and have not included them in this thesis and have not cited as my own work.

Date: June 8, 2025
Place: Varanasi, India

Signature of the Student

CERTIFICATE BY THE SUPERVISOR

It is certified that the above statement made by the student is correct to the best of my/our knowledge.

Supervisor
Prof. Dr. Vignesh Sivaraman
DEPARTMENT OF COMPUTER SCIENCE
Indian Institute Of Technology
(Banaras Hindu University)
Varanasi – 221 005

Signature of Head of Department

COPYRIGHT TRANSFER CERTIFICATE

Title of the Thesis: Cost-Aware and Resilient Placement of Parallel Service Function Chains Using Multi-Stage Graph Modeling

Name of the Student: HIMANSHU YADAV

COPYRIGHT TRANSFER

The undersigned hereby assigns to the Indian Institute of Technology (Banaras Hindu University), Varanasi all rights under copyright that may exist in and for the above thesis submitted for the award of the Integrated Dual Degree (B.Tech.+ M.Tech.) in COMPUTER SCIENCE.

Date: June 8, 2025
Place: Varanasi, India

Signature of the Student

Note: However, the author may reproduce or authorize others to reproduce material extracted verbatim from the thesis or derivative of the thesis for author's personal use provided that the source and the Institute's copyright notice are indicated.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Dr. Vignesh Sivaraman for his invaluable guidance, constant support, and insightful feedback throughout the course of this project. His mentorship was instrumental in shaping the direction and execution of this work, and his inputs were critical at every stage of the research.

I am also deeply thankful to my parents for their unwavering encouragement, motivation, and emotional support throughout this journey. Their belief in me has been a continuous source of strength.

Finally, I would like to acknowledge the faculty, staff, and peers at IIT (BHU), Varanasi, for providing a stimulating academic environment and for their cooperation and support during the completion of this thesis.

Date: June 8, 2025
Place: Varanasi, India

Signature of the Student

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

With the advent of 5G, IoT, and latency-sensitive cloud-native services, modern communication networks increasingly rely on dynamic and programmable service infrastructures. Two fundamental technologies enabling this shift are Software Defined Networking (SDN) and Network Function Virtualization (NFV). Together, they allow traditional network functions—such as firewalls, NAT, load balancers, and intrusion detection systems—to be decoupled from proprietary hardware and implemented in software, known as Virtual Network Functions (VNFs).

To enforce end-to-end policies, security, and performance requirements, VNFs are deployed in ordered sequences, forming what is known as a *Service Function Chain (SFC)*. These SFCs are essential for realizing service-level agreements (SLAs) across multi-tenant or operator networks. Typical SFC deployment problems include deciding where to place each VNF and how to route the traffic between them while satisfying resource constraints (CPU, bandwidth) and Quality of Service (QoS) requirements like latency or reliability.

Traditionally, SFCs are deployed in a strictly sequential fashion (s-SFC), where each VNF processes packets in a serialized manner. However, sequential chaining often leads to high end-to-end latency and inflexible deployment, especially when the number of VNFs is

large or traffic volumes are high. This is particularly problematic for applications such as real-time monitoring, autonomous control, and remote industrial communication, where low latency and high availability are mandatory.

To address this, the concept of *Parallel SFCs (p-SFCs)* has emerged. Instead of a linear VNF chain, a p-SFC is modeled as a Directed Acyclic Graph (DAG) in which certain VNFs that are functionally independent can operate in parallel. This parallelism has the potential to significantly reduce packet processing delays, improve throughput, and better utilize available resources across the network. However, enabling parallel deployment introduces several new challenges, such as ensuring processing correctness, managing duplicated traffic across split branches, and bounding the extra resource overhead introduced by parallel flows.

Furthermore, in production-grade networks, reliability is a critical requirement. To handle node failures or overloads, each VNF placement must be supported by a disjoint *backup path* that satisfies the same resource and delay constraints. In addition, to ensure visibility and control, monitoring mechanisms must be deployed—typically as Virtual Monitoring Functions (VMFs)—to observe all active and backup paths without introducing excessive redundancy.

These requirements collectively form a complex placement problem that must jointly consider parallel SFC decomposition, active and backup path selection, resource reservation, and observability, all while keeping the total deployment cost low. Designing a practical and scalable solution to this problem is the core motivation of this thesis.

1.2 Problem Overview

This thesis addresses the following two core problems:

1. **Cost-Aware and Reliable Parallel SFC Placement:** Given a set of SFCs, each decomposed into a sequence of parallel entities (PEs), the goal is to place both active and disjoint backup instances of these PEs onto a physical substrate network

while minimizing the total deployment cost. The solution must satisfy CPU and bandwidth constraints, as well as strict end-to-end delay requirements.

2. **Efficient VMF Placement:** The second objective is to place Virtual Monitoring Functions (VMFs) such that every VNF instance (in both active and backup paths) is monitored, while minimizing the number of monitoring nodes and the communication overhead.

Both problems are inherently NP-hard and require resource-aware, delay-constrained, and reliability-aware optimization strategies.

1.3 Proposed Approach

To solve the parallel SFC placement problem, we propose a graph-based heuristic that constructs a *Multi-Stage Graph* (MSG) for each SFC. Each stage corresponds to a parallel entity, and each node in the MSG represents a feasible placement of that PE on a physical node. Edges between stages capture connectivity with acceptable delay and bandwidth characteristics. The algorithm finds cost-efficient paths through this MSG for both active and backup instances, ensuring disjointness and compliance with all constraints.

For the VMF placement problem, we formulate it as a *facility location problem*. Each physical node hosting a PE acts as a client that must be monitored. The goal is to select a minimal set of monitoring nodes (facilities) such that all PEs are covered and the communication cost between PEs and VMFs is minimized. A greedy heuristic is proposed to solve this efficiently.

1.4 Contributions

The primary contributions of this thesis are as follows:

- We define a novel system model for parallel SFC deployment, incorporating parallel entities, active and backup paths, delay constraints, and observability through monitoring.

- We develop a cost-aware and delay-constrained heuristic algorithm for reliable placement of parallel SFCs using multi-stage graph construction, ensuring disjoint backup provisioning.
- We propose a greedy algorithm for reliable VMF placement that minimizes the number of monitoring nodes while ensuring full coverage of all VNFs in both active and backup paths.
- We implement a full simulation framework to evaluate the proposed algorithms under realistic physical topologies and varying service loads, and demonstrate performance improvements in cost, delay satisfaction, and monitoring overhead.

Chapter 2

Literature Review

The deployment of Service Function Chains (SFCs) in softwarized networks has been widely studied across domains such as Network Function Virtualization (NFV), cloud-native orchestration, and software-defined networking (SDN). This chapter presents a review of foundational and state-of-the-art approaches to SFC placement, with particular attention to works on parallel SFC decomposition, reliability-aware deployment, and network monitoring using Virtual Monitoring Functions (VMFs).

2.1 Sequential SFC Placement and Optimization

Early research efforts primarily focused on mapping linear (sequential) SFCs to physical infrastructure. These methods formulate placement as a resource-constrained optimization problem, with objectives ranging from cost minimization to latency satisfaction.

Mehraghdam et al. [10] modeled SFCs using a tuple-based structure and solved placement using Mixed Integer Linear Programming (MILP). Luizelli et al. [7] presented a joint placement and chaining framework using precomputed paths to speed up decision making. However, such approaches do not scale well in large networks and cannot exploit the parallelism inherent in many real-world SFCs.

To overcome complexity, heuristic and approximation methods have been proposed. For

example, Bari et al. [1] used a greedy approach to place VNFs near data sources, improving delay but sacrificing optimality.

Mijumbi et al. [11] explored mapping and scheduling algorithms that balance performance and feasibility in resource-constrained networks. A comprehensive survey by Wang et al. [14] compared various SFC orchestration techniques, emphasizing challenges in QoS provisioning and scalability.

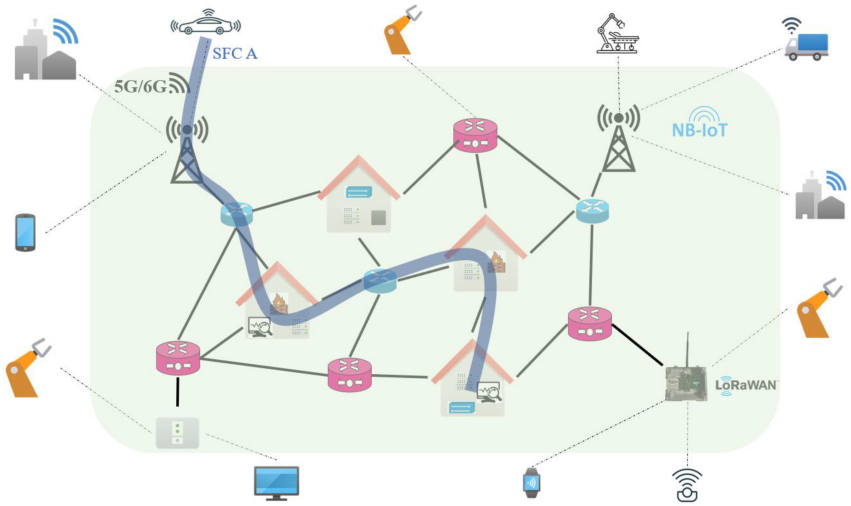


Figure 2.1: Illustration of the Sequential SFC Placement

2.2 Parallel SFC Placement and Multi-Stage Modeling

Traditional sequential Service Function Chains (s-SFCs) impose a rigid linear structure on VNF traversal. While this is simple to manage, it becomes inefficient for latency-sensitive or high-throughput applications. Several recent works have explored the use of *Parallel SFCs (p-SFCs)*, which enable multiple VNFs to be executed concurrently when their functions are not order-dependent.

The modeling of p-SFCs often takes the form of a Directed Acyclic Graph (DAG), where VNFs can be distributed across multiple parallel branches. As shown in Figure 2.2, each source flow (e.g., $S1$, $S2$) is split and routed through parallel virtual function paths.

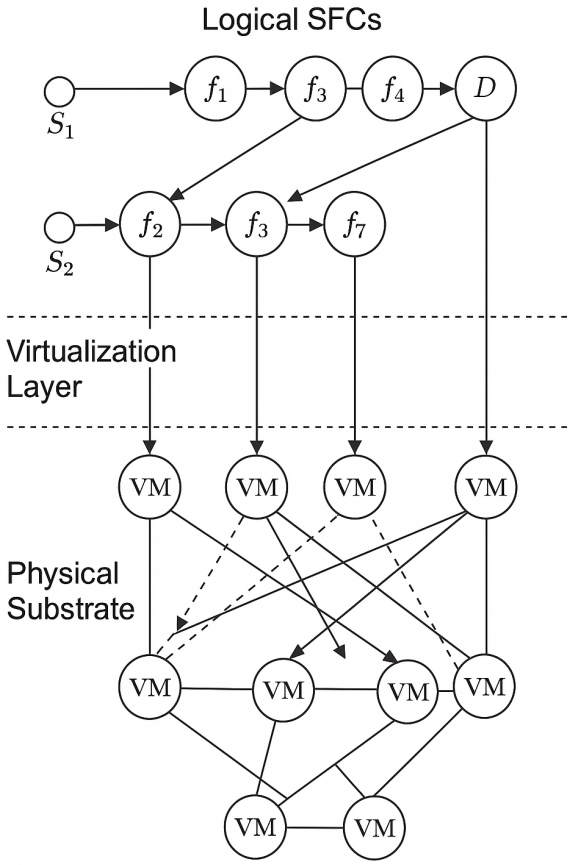


Figure 2.2: Illustration of Parallel SFC Modeling with Virtualized Topology and VNF Mapping

The ParaSFC model [9] introduces parallelization-aware chaining using layered forwarding graphs with embedded backup logic. LPulse [6] addresses large-scale SFC placement under delay constraints using label-based pulse pruning. SplitSFC [3] further refines p-SFC embeddings using latency-driven partitioning across substrate paths.

This multi-stage modeling introduces new challenges:

- Coordinating resource competition among parallel branches.
- Avoiding redundant resource use due to duplicated traffic across branches.
- Maintaining processing correctness across reordered and parallelized functions.
- Efficiently mapping each VNF stage to feasible physical nodes under resource and QoS constraints.

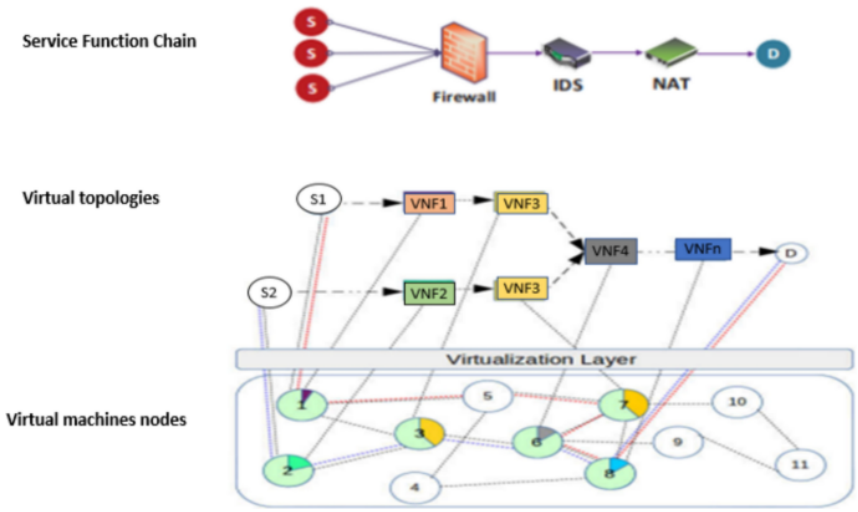


Figure 2.3: Illustration of Service Function Chain parallelization

2.3 Reliability-Aware and Backup Placement Strategies

Fault tolerance is a critical requirement in production-grade networks. Recent works have integrated disjoint path construction and backup VNF placement into the orchestration logic.

Xie et al. [15] proposed probabilistic models for failure-aware VNF deployment. Ghosh et al. [2] incorporated stochastic programming to minimize risk under failure events. Lin et al. [5] proposed robust placement strategies under uncertain failure scenarios using protective mapping.

In the ParaSFC extension [8], node-disjoint backups are enforced by removing active path nodes from the solution space and penalizing delay overhead in backup detours.

2.4 VMF Placement and Observability in NFV

Service observability is indispensable for SLA validation and fault detection in NFV infrastructures. Virtual Monitoring Functions (VMFs) provide fine-grained traffic oversight by being co-located with VNFs or placed strategically in the substrate.

Pan and Lin [12] formulated a joint optimization of VMF placement and reliability using

Integer Linear Programming. Ghosh et al. [2] minimized communication overhead by modeling VMF provisioning as a facility location problem.

Our work uses a greedy placement mechanism that ensures:

- All active and backup nodes are covered.
- Redundant deployments are avoided via VMF reuse.
- Placement is feasible on shared infrastructure without special hardware.

2.5 Research Gaps and Motivation

While existing works have addressed the following in isolation:

- Delay-aware SFC placement,
- Parallel SFC decomposition,
- Backup and failure-resilient mappings,
- Efficient monitoring using VMFs,

there is a clear absence of integrated frameworks that combine all these components.

This thesis bridges that gap by proposing:

- A unified placement system for parallel SFCs with disjoint backup paths.
- A scalable MSG abstraction for efficient deployment and rerouting.
- A greedy yet coverage-optimal VMF placement algorithm.

Recent studies like Qi et al. [13] have shown promise using deep learning, but lack integrated backup and monitoring logic. Our contribution unifies all aspects into one coherent framework designed for edge and telco-scale infrastructures.

Chapter 3

System Model

In this chapter, we describe the system architecture that models the physical substrate network, parallelized Service Function Chains (P-SFCs), and Virtual Monitoring Function (VMF) placement. The model captures the constraints and requirements necessary for cost-aware, reliable, and observable deployment of service chains.

3.1 Physical Substrate Network

The physical infrastructure is represented as an undirected graph:

$$G_p = (N, E)$$

where:

- N is the set of physical nodes (e.g., edge/cloud servers),
- $E \subseteq N \times N$ is the set of physical links.

Each physical node $k \in N$ is characterized by:

- CPU capacity C_k,
- A binary activation state $a_k \in \{0, 1\}$, which is 1 if the node is activated for hosting at least one function,

- Per-unit CPU deployment cost and fixed node activation cost.

Each physical link $e = (k_1, k_2) \in E$ has:

- Bandwidth capacity $B_{k_1 k_2}$,
- Delay $D_{k_1 k_2}$,
- Link traversal cost $L_{k_1 k_2}$ per unit of traffic.

3.1.1 Illustration of SFC Mapping to Physical Network

Figure 3.1 illustrates a high-level view of how a service function chain (SFC) is mapped onto a physical SDN/NFV network. The SFC request specifies a sequence of virtual functions (e.g., Firewall, DPS, NAT), which must be executed in order for traffic originating from an edge source (e.g., sensor) to reach its destination (e.g., operation center).

The underlying infrastructure comprises two types of nodes:

- **Function nodes:** Capable of hosting VNFs (shown as server icons).
- **Switch nodes:** Responsible for forwarding traffic (shown as router icons).

Multiple physical paths (P1, P2, P3) are shown that satisfy the same SFC request. Each path uses a different combination of VNF instances and switch nodes. This visualizes the flexibility and complexity of the placement problem — the system must not only choose feasible placements for each VNF but also construct end-to-end paths that satisfy delay and bandwidth constraints, while minimizing cost.

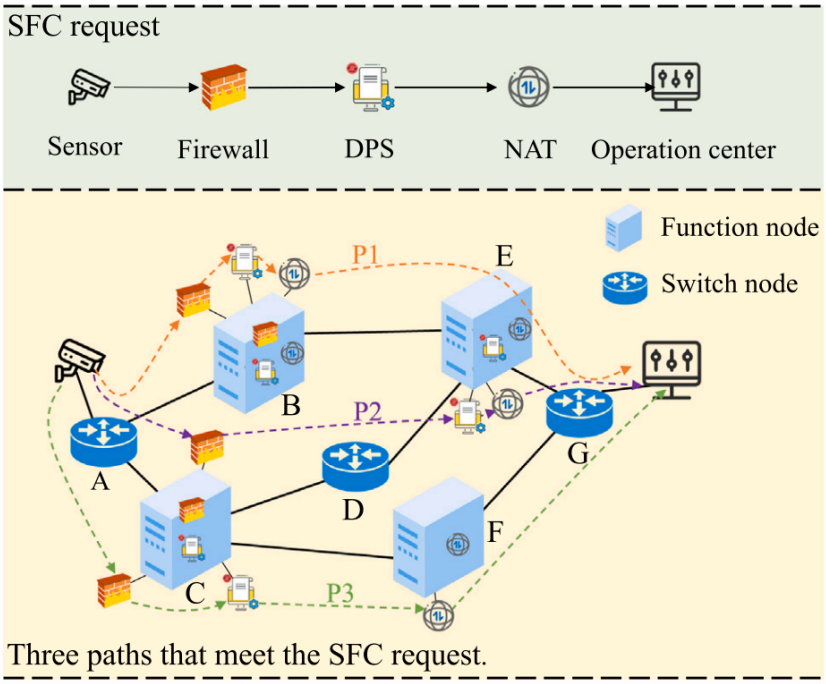


Figure 3.1: SFC request and its mapping over multiple physical paths in an SDN/NFV network.

3.2 Service Function Chains (P-SFCs)

Let $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ denote a set of service chain requests. Each service chain $S_j \in \mathcal{S}$ is defined as:

$$S_j = (src_j, dst_j, B_j^q, D_j^q, R_j^q, S_j)$$

where:

- $src_j, dst_j \in N$: Source and destination nodes,
- B_j^q: Required bandwidth for the chain,
- D_j^q: End-to-end delay budget,
- R_j^q: Resource demand vector (e.g., CPU per PE),
- $S_j = \{PE_j^1, PE_j^2, ..., PE_j^K\}$: A sequence of K *Parallel Entities* (PEs), each of which contains one or more VNFs that can operate concurrently.

Each Parallel Entity PE_j^k represents a processing stage that must be placed on a single physical node, and adjacent PEs are connected through physical links forming a logical chain. For reliability, each PE_j^k is deployed in both *active* and *backup* forms on disjoint physical nodes.

3.3 Cost Model and Constraints

The placement of Parallel Service Function Chains (P-SFCs) across a physical network involves multiple cost dimensions and is subject to strict resource and reliability constraints. The overall objective is to minimize the total deployment cost while ensuring feasible and delay-compliant placements for both active and backup paths.

3.3.1 Cost Model

We define four cost components contributing to the total system cost:

- **C_1: Node Activation Cost.** A fixed cost is incurred for each physical node that is activated to host at least one Parallel Entity (PE). This reflects operational expenses such as energy and management overhead. If a node is unused, it contributes no cost.

$$C_1 = \text{Total cost of all activated nodes}$$

- **C_2: PE Deployment Cost.** Each active and backup PE consumes CPU on its hosting node. This cost is proportional to the total CPU resources reserved for all SFC stages.

$$C_2 = \sum \text{CPU usage} \times \text{cost per unit}$$

- **C_3: Link Traversal Cost.** Service traffic between PEs incurs cost while traversing physical links. This is computed based on the number of hops, link usage, and required bandwidth.

$$C_3 = \sum (\text{bandwidth} \times \text{path length}) \times \text{link cost}$$

- **C_4: Backup Delay Cost.** The physical distance or delay between corresponding active and backup PEs adds to recovery overhead. This component penalizes far-separated placements.

$$C_4 = \sum \text{distance(active\_PE, backup\_PE)}$$

The final optimization objective is:

$$\text{Minimize } \mathbf{C} = \alpha_1 C_1 + \alpha_2 C_2 + \alpha_3 C_3 + \alpha_4 C_4$$

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ are weights that control the relative importance of each component.

3.3.2 Placement Constraints

To ensure feasibility and quality of service, the following constraints must be satisfied:

- **CPU Constraint:** Each physical node has a limited CPU capacity. The total demand from all mapped PEs (both active and backup) must not exceed its available capacity.

$$\text{Used CPU at node } v \leq \text{CPU capacity of } v$$

- **Bandwidth Constraint:** The sum of bandwidth used by active and backup paths on any link must be less than the link's total capacity.

- **Delay Constraint:** For each SFC, the total end-to-end delay, including link and processing delays, must be within the service's delay bound D_s.

$$\text{Total delay}_{\text{SFC}} \leq D_s$$

- **Disjointness Constraint:** Active and backup PEs of the same stage must be placed on different physical nodes to ensure fault tolerance. Similarly, their paths must not share links or nodes.

- **Mapping Constraint:** Each stage of a P-SFC must be assigned to exactly one feasible physical node. There should be no skipped or duplicated mappings.

These constraints guide the selection of placements in the Multi-Stage Graph (MSG) and ensure both efficiency and resilience in SFC deployment.

3.4 Virtual Monitoring Functions (VMFs)

To ensure observability and SLA compliance, each SFC placement must include Virtual Monitoring Functions (VMFs) deployed in the substrate network. VMFs serve as monitoring agents that oversee traffic passing through active and backup nodes.

Let $\mathcal{M} \subseteq N$ be the set of nodes chosen to host VMFs. Each VMF monitors a subset of nodes used in SFC placements. The monitoring model must satisfy:

- **Coverage Constraint:** Every node involved in active or backup placement must be monitored by at least one VMF.
- **Communication Overhead:** The number and location of VMFs must minimize the total monitoring communication cost.
- **Facility Model:** A node may host a VMF only once and incurs a fixed monitoring cost.

3.5 Objective Overview

Given a set of parallel SFCs $\mathcal{S} = \{S_1, ..., S_m\}$, and a physical network graph $G_p = (N, E)$, the objective is to find cost-effective and reliable placements of active and backup instances of all PEs, while ensuring complete monitoring coverage via VMFs.

Objective Function:

$$\min \left(C_{\text{active}} + C_{\text{backup}} + C_{\text{deploy}} + C_{\text{delay-backup}} + C_{\text{monitoring}}\right)$$

Where:

- C_{active}: Total path cost for active placement,
- C_{backup}: Path cost for disjoint backup placement,
- C_{deploy}: CPU-based deployment and node activation costs,
- $C_{\text{delay-backup}}$: Delay cost for backup state synchronization,
- $C_{\text{monitoring}}$: Total communication and activation cost of VMFs.

Subject to the following constraints:

- **Node Capacity:**

$$\sum_{\text{PEs on node } n} R_j^q \leq C_n, \quad \forall n \in N$$

- **Bandwidth:**

$$\text{Used bandwidth on } (u, v) \leq B_{uv}, \quad \forall (u, v) \in E$$

- **End-to-End Delay:**

$$\sum_{\text{path segments}} D_{uv} \leq D_j^q$$

- **Active/Backup Disjointness:**

$$\text{PE}_{active}^k \neq \text{PE}_{backup}^k, \quad \forall k$$

- **Monitoring Coverage:**

$$\forall \text{ used node } n, \exists \text{ VMF } m \text{ such that VMF covers } n$$

This formulation ensures resource feasibility, QoS satisfaction, reliability through disjoint backups, and complete observability via monitoring functions.

Chapter 4

Methodology

This chapter presents our methodology for cost-aware and reliable placement of Parallel Service Function Chains (P-SFCs), along with the deployment of Virtual Monitoring Functions (VMFs) to ensure observability.

We propose a scalable heuristic-based strategy that models the SFC placement problem as a layered graph traversal task, implemented through the construction of a **Multi-Stage Graph (MSG)**. This enables the selection of both active and disjoint backup paths under resource and delay constraints. A lightweight greedy strategy is employed to determine the optimal locations for monitoring functions. The methodology is modular and consists of four core steps: MSG construction, active path selection, backup path generation, and VMF placement.

Each of these steps is detailed below with algorithmic explanations, design motivations, and complexity insights.

4.1 Design Motivation

Limitations of ILP and Exhaustive Approaches

Traditional Integer Linear Programming (ILP) or brute-force solutions provide exact placement results but suffer from exponential complexity. For a large-scale network with mul-

tiple concurrent SFCs, the search space quickly becomes unmanageable due to:

- Combinatorial explosion in node-to-stage mapping.
- Difficulty in enforcing disjointness and end-to-end delay constraints.
- Infeasibility in real-time dynamic request scenarios.

To overcome these limitations, we propose a heuristic approach built around the concept of Multi-Stage Graphs.

What is a Multi-Stage Graph (MSG)?

A Multi-Stage Graph is a layered, directed acyclic graph (DAG) specifically constructed to model SFC placement. Each stage in the graph corresponds to a processing stage (PE) in the parallel SFC. The key properties of MSG include:

- **Stage-wise structure:** The graph is partitioned into K layers (stages), where K is the number of PEs in the SFC.
- **Candidate node filtering:** Each node in a stage represents a physical node in the substrate network that satisfies CPU requirements for the corresponding PE.
- **Inter-layer edges:** Directed edges connect nodes in consecutive stages if a feasible physical path (meeting delay and bandwidth constraints) exists between them.
- **Edge weights:** Each edge carries a cost and delay value, inherited from the corresponding physical link/path.

How Does MSG Work?

The MSG transforms the SFC placement problem into a shortest-path traversal over a constrained graph:

1. For each PE stage, candidate physical nodes are selected based on CPU availability.
2. Edges between nodes in adjacent stages are added only if a valid physical path exists and satisfies the delay bound.

3. Pathfinding algorithms (e.g., Dijkstra for DAGs) are used to find the least-cost route from the first to the last stage.

4. The selected path maps each PE to a distinct physical node, with guaranteed resource feasibility.

Why Use MSG?

MSG offers several advantages over traditional approaches:

- **Efficient pruning:** Infeasible mappings are excluded during graph construction, significantly reducing the search space.
- **Modularity:** MSG construction and traversal are decoupled, enabling reuse for both active and backup path planning.
- **Constraint embedding:** CPU, delay, and bandwidth constraints are embedded into node and edge selection, avoiding post-hoc validation.
- **Supports parallel SFCs:** Since MSG works independently per SFC, multiple requests can be processed in parallel or batched.

Greedy Heuristics for VMF Placement

While MSG handles the SFC routing and placement, the observability requirement is managed through a greedy VMF placement scheme:

- Every physical node used in any active or backup path must be monitored.
- Instead of solving a complex optimization problem, we deploy VMFs incrementally — only on nodes that are not already monitored.
- This approach ensures complete coverage while avoiding redundant monitoring deployments.

Overview of the Complete Pipeline

Each SFC is processed independently via the following pipeline:

1. Construct the active MSG and place the optimal active path.

2. Construct a backup MSG excluding used nodes, and place a disjoint backup path.

3. Store all physical nodes used.

4. After processing all SFCs, deploy VMFs on uncovered nodes using a greedy strategy.

This modular, layer-wise, and delay-aware design enables real-time applicability, scalability, and high reliability in the placement of SFC.

This chapter presents our methodology for cost-aware and reliable placement of parallel Service Function Chains (P-SFCs), along with the deployment of Virtual Monitoring Functions (VMFs) to ensure observability. We use a heuristic-based design centered on Multi-Stage Graph (MSG) construction for path selection and a greedy placement strategy for VMFs. This chapter is organized into step-wise blocks, each with its motivation, procedure, and algorithm.

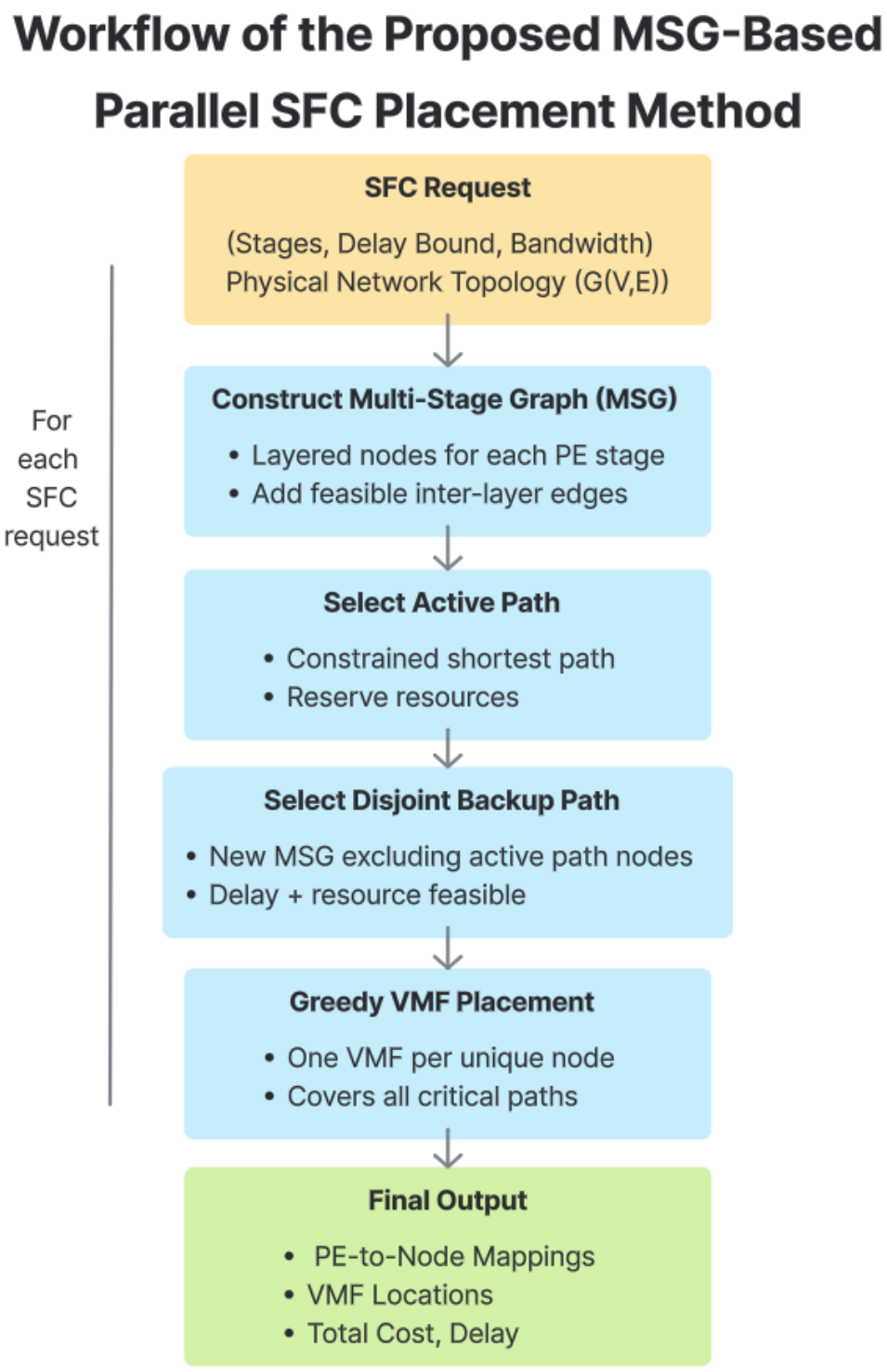


Figure 4.1: Workflow of the Proposed MSG-Based Parallel SFC Placement Method

4.2 Step 1: Multi-Stage Graph (MSG) Construction

4.2.1 Motivation

The first and most critical step in deploying an SFC is selecting where each Processing Element (PE) stage should be placed on the physical network. However, evaluating all combinations of PE-to-node mappings is computationally infeasible. We need a method that allows us to:

- Reduce the search space before placement.
- Respect CPU, delay, and bandwidth constraints.
- Structure the problem in a way that supports efficient path computation.

This motivates the construction of a **Multi-Stage Graph (MSG)** — a layered representation that mirrors the SFC structure while embedding physical feasibility.

4.2.2 Theory

Each SFC consists of K processing stages. The MSG is a directed acyclic graph (DAG) composed of K layers, where:

- Each layer i represents the i-th PE stage.
- Each node in layer i corresponds to a physical node v with enough CPU to host the PE stage i.
- Edges are added only between nodes in adjacent layers (from stage i to $i + 1$) if a valid physical path exists that satisfies the delay and bandwidth constraints.
- The edge weight is derived from the physical path's cost (typically a function of link cost or delay).

This construction transforms the SFC placement into a structured pathfinding problem in a restricted and constraint-compliant search space.

4.2.3 Procedure

1. **Stage-wise node selection:** For each PE stage, filter physical nodes that:
   - Are not in the exclusion list (used for disjoint backup)
   - Have residual CPU $\geq$ required for that PE

   These are added as virtual nodes to the corresponding MSG stage.

2. **Inter-layer edge construction:** For every pair of nodes (u, v) where u belongs to stage i and v to stage $i + 1$:
   - Check if a physical path exists from u to v
   - Ensure the path meets delay and bandwidth requirements
   - Add a directed edge with cost and delay metadata

3. **Return the MSG:** The output is a layered, delay- and resource-aware DAG that encodes all feasible PE placements for this SFC.

4.2.4 Algorithm

Algorithm 1 ConstructMSG($G_p, Stages, ExcludeNodes$)

```
Initialize empty MSG
for each stage i in Stages do
    for each physical node v ∈ G_p.nodes do
        if v ∉ ExcludeNodes and CPU[v] ≥ PE[i] then
            Add virtual node (i, v) to MSG
        end if
    end for
end for
for each pair of consecutive stages (i, i + 1) do
    for each node u in MSG stage i do
        for each node v in MSG stage i + 1 do
            if physical path from u → v exists and satisfies constraints then
                Compute edge weight based on path delay/cost
                Add directed edge (u, v) to MSG
            end if
        end for
    end for
end for
return MSG
```

4.2.5 Insight

By transforming PE placement into shortest-path traversal over a structured graph, we dramatically reduce the computational burden and embed constraints directly into the graph model. This step enables efficient downstream placement while guaranteeing feasibility.

4.3 Step 2: Active Path Placement

4.3.1 Motivation

Once the multistage graph (MSG) is constructed, we must identify a valid and efficient route through it that represents the placement of all PEs in the given SFC. This route determines which physical nodes will host the stages of the SFC. The placement must:

- Satisfy end-to-end delay constraints
- Minimize total cost (link + deployment)
- Ensure no resource violation on selected nodes or paths

Solving this through exact optimization (e.g., ILP) is computationally expensive. Instead, we use a shortest-path approach over the layered DAG (MSG) to identify a feasible and cost-effective active path.

4.3.2 Theory

Each layer of MSG corresponds to a PE stage, and each inter-layer edge represents a feasible link between physical nodes satisfying delay, bandwidth, and disjointness constraints.

Since MSG is a Directed Acyclic Graph (DAG), classical shortest-path algorithms (e.g., topological sort + relaxation or layered Dijkstra) can be efficiently applied:

- Source nodes are from stage 0; destination nodes from final stage K
- Edge weights represent link cost (e.g., latency or monetary cost)

- A path through MSG maps each PE to a unique physical node, ensuring valid deployment

Additionally, the feasibility of each candidate path is validated against the following:

- Node CPU availability
- Path delay constraints

4.3.3 Procedure

1. **Path Enumeration or Dijkstra:** Use shortest-path algorithms to compute the minimum-cost path from any node in stage 0 to any node in stage K.
2. **Feasibility Filtering:** Validate that:
   - All physical nodes on the path have sufficient CPU
   - The cumulative delay across hops and entry/exit paths is within the delay budget
3. **Path Selection and Resource Reservation:** Choose the valid path with the minimum total cost. Reserve CPU and bandwidth along the path to prevent future collisions.
4. **Return the active path:** The result is a valid PE-to-node mapping for active SFC deployment.

4.3.4 Algorithm

Algorithm 2 PlaceActivePath(MSG)

```
Initialize shortest-path tracker over MSG
for each node v in stage 0 do
    Run shortest-path from v to all stage-K nodes
    Track paths satisfying total delay and resource constraints
end for
Select path p_active with minimum total cost
Reserve CPU and bandwidth resources along p_active
return p_active
```

4.3.5 Insight

This step transforms PE-to-node mapping into a constrained graph traversal task. It ensures low-cost, resource-aware, and delay-compliant deployment, enabling highly scalable SFC provisioning in complex networks.

4.4 Step 3: Disjoint Backup Path Placement

4.4.1 Motivation

In mission-critical services, failure resilience is essential. If a node or link on the active path fails, traffic must be rerouted without violating service continuity or performance bounds. This requires the construction of a **backup path** that:

- Is completely disjoint from the active path (node-level disjointness).
- Satisfies the same resource and delay constraints.
- Can be pre-reserved to allow rapid failover.

Enforcing node-level disjointness enhances robustness against both node and link failures.

4.4.2 Theory

To build a disjoint path:

- We exclude all physical nodes used in the active path from candidate consideration.
- A new MSG is constructed over the reduced set of nodes.
- Pathfinding is again performed on this new graph.

The final backup path must:

- Use only unused physical nodes.
- Satisfy end-to-end delay and bandwidth constraints.
- Be reservable in advance to guarantee fast switchover.

We also compute a **backup delay cost**, which measures the cumulative delay from each active PE to its corresponding backup PE. This cost models the synchronization overhead during failover.

4.4.3 Procedure

1. **Exclude active nodes:** Extract the set of all physical nodes used in the active path and exclude them from MSG construction.
2. **Construct Backup MSG:** Build a new MSG using the filtered node set to enforce node-disjointness.
3. **Find disjoint path:** Use the same shortest-path procedure to find the minimum-cost valid backup path.
4. **Reserve and compute delay:** Reserve resources along the path and calculate the pairwise delay between each stage of the active and backup paths.
5. **Return:** Return the backup path along with its cost and synchronization delay metrics.

4.4.4 Algorithm

Algorithm 3 PlaceBackupPath($G_p, Stages, p_{\text{active}}$)

$ExcludeNodes \leftarrow$ physical nodes used in p_{active}
$MSG_{\text{backup}} \leftarrow$ ConstructMSG($G_p, Stages, ExcludeNodes$)
Compute all valid paths in MSG_{backup}
Select minimum-cost backup path p_{backup}
Reserve CPU and bandwidth along p_{backup}
Compute backup delay cost between p_{active} and p_{backup}
return p_{backup}

4.4.5 Insight

Disjoint backup planning is critical for high availability. By reconstructing the MSG after excluding used nodes, we maintain isolation between active and backup paths. This en-

sures robustness while preserving scalability due to the reuse of the efficient MSG framework.

4.5 Step 4: Greedy VMF Placement

4.5.1 Motivation

To ensure observability, it is essential to monitor all physical nodes involved in service delivery. Monitoring becomes especially important when backup paths are provisioned, as faults may require instant switchover.

Deploying a **Virtual Monitoring Function (VMF)** on every used node allows system administrators or automated systems to:

- Continuously track node status and performance.
- Quickly detect anomalies or failures.
- Enable fault diagnostics and proactive alerts.

However, deploying VMFs on every node indiscriminately is inefficient. Our goal is to ensure complete node coverage *without redundancy*, using a fast and memory-efficient strategy.

4.5.2 Theory

Each SFC placement (active and backup) uses a subset of the physical nodes. The union of all such nodes across all SFCs forms the set of ”critical nodes” that must be monitored.

Rather than solving a costly facility location problem, we adopt a greedy strategy:

- For every SFC, extract all used physical nodes.
- Maintain a global set of already-covered nodes.
- If a node has not yet been monitored, deploy a VMF.

This strategy guarantees:

- Coverage of every active and backup node exactly once.
- Linear-time execution across placements.
- Minimal monitoring overhead.

4.5.3 Procedure

1. **Initialization:** Create empty lists for deployed VMFs and for the set of nodes already monitored.
2. **Placement Iteration:** For every placement (active + backup) in the deployment history:
    - Loop through each physical node used in that placement.
    - If the node is not already monitored, deploy a VMF on it.
3. **Output:** Return the complete set of VMFs and their hosting nodes.

4.5.4 Algorithm

Algorithm 4 PlaceVMFs($Placements$)

```
Initialize VMFs ← ∅, Covered ← ∅
for each placement p in Placements do
    for each node v in p_active ∪ p_backup do
        if v ∉ Covered then
            Deploy VMF at v and add to VMFs
            Mark v as covered
        end if
    end for
end for
return VMFs
```

4.5.5 Insight

This greedy VMF placement algorithm delivers full observability while minimizing cost and avoiding redundant monitoring. Its simplicity and efficiency make it practical for real-world deployment across dynamically scaling or heterogeneous SFCs.

4.6 Summary of the Methodology

To summarize, our proposed approach enables scalable and resilient deployment of parallel Service Function Chains (P-SFCs) through a modular and constraint-aware heuristic. The process is composed of four primary steps:

1. **Multi-Stage Graph (MSG) Construction:** A layered representation of feasible PE-to-node mappings is constructed, embedding CPU, delay, and bandwidth constraints.
2. **Active Path Placement:** The minimum-cost feasible path through the MSG is selected to deploy the active instance of the P-SFC.
3. **Disjoint Backup Path Placement:** A secondary path is computed using a reduced MSG that excludes nodes from the active path, ensuring disjointness and resilience.
4. **Greedy VMF Placement:** Monitoring functions are deployed only once per physical node involved in any active or backup path, ensuring observability with minimal redundancy.

This pipeline achieves a balance between deployment efficiency, fault tolerance, and monitoring coverage. In the next section, we formally analyze the computational complexity of this methodology.

4.7 Time and Space Complexity Analysis

We analyze the computational complexity of both the brute-force and MSG-based heuristic algorithms in terms of time and space, highlighting the scalability improvements achieved by our approach.

4.7.1 MSG-Based Heuristic Algorithm

Time Complexity

Let:

- M: Average number of feasible candidate nodes per stage
- K: Number of stages
- E: Number of physical links
- N: Total number of physical nodes

The proposed algorithm performs the following operations per SFC:

1. **Candidate Filtering:** For each stage, all N nodes are scanned for CPU feasibility:

$$O(K \cdot N)$$

2. **MSG Edge Construction:** Each node in stage i is connected to all feasible nodes in stage $i + 1$ via path delay checks, leading to:

$$O(K \cdot M^2 \cdot (E + N \log N))$$

3. **Shortest Path Computation in MSG:** Once the MSG is built, shortest path across layers is computed using a DAG-aware version of Dijkstra:

$$O(K \cdot M \log M)$$

Therefore, the total time complexity per SFC is:

$$O\left(K \cdot M^2 \cdot (E + N \log N) + K \cdot M \log M\right)$$

This can be rounded and simplified as:

$$O\left(K \cdot M^2 \cdot (E + N \log N)\right)$$

since $M^2 \cdot (E + N \log N) \gg M \log M$ for moderately sized graphs.

VMF Placement Complexity

After all SFCs are placed, VMFs are deployed on physical nodes used in active and backup paths. A simple greedy algorithm checks for uniqueness and deploys one VMF per uncovered node.

- **Time Complexity:**

$$O(R \cdot K)$$

  where R is the number of SFCs, and each has K stages. In the worst case, every PE maps to a unique node, leading to linear growth.

- **Space Complexity:**

$$O(N)$$

  to store covered nodes and VMF metadata.

The VMF placement algorithm is fast and scalable with negligible overhead.

Space Complexity

The MSG stores:

- $K \cdot M$ virtual nodes across stages
- $K \cdot M^2$ inter-stage edges with weights (delay, cost)

Hence, the space complexity is:

$$O(K \cdot M^2)$$

4.7.2 Brute-Force Algorithm

Time Complexity

Let:

- N: Number of physical nodes
- K: Number of stages (i.e., PEs) in the SFC

- E: Number of physical links

The brute-force algorithm evaluates all valid mappings of K PEs to distinct physical nodes, resulting in:

$$\text{Number of combinations} = \frac{N!}{(N-K)!}$$

For each combination, it checks CPU feasibility and computes the cost of the active and backup paths using shortest-path algorithms such as Dijkstra. Hence, the total time complexity per SFC becomes:

$$O\left(\frac{N!}{(N-K)!} \cdot K \cdot (E + N \log N)\right)$$

Space Complexity

The algorithm may need to store all possible mappings and their associated cost and delay values. Thus, the worst-case space complexity is:

$$O\left(\frac{N!}{(N-K)!} \cdot K\right)$$

This factorial growth renders brute-force approaches impractical for realistic network sizes.

4.7.3 Comparison Summary

Table 4.1: Time and Space Complexity Comparison (Approximate)

Approach	Time Complexity	Space Complexity
MSG-Based Heuristic	$O(K \cdot M^2 \cdot (E + N \log N))$	$O(K \cdot M^2)$
Greedy VMF Placement	$O(R \cdot K)$	$O(N)$
Brute-Force Search	$O\left(\frac{N!}{(N-K)!} \cdot K \cdot (E + N \log N)\right)$	$O\left(\frac{N!}{(N-K)!} \cdot K\right)$

4.8 Approximation Insight

To evaluate how close the MSG-based method comes to the optimal solution, we derive a formal approximation bound for the total cost incurred.

Notation

- Let MSG be the total cost incurred by the MSG-based placement method.
- Let OPT be the optimal cost achieved by an ILP or brute-force approach.
- Let K be the number of stages (PEs) in an SFC.
- Let δ be the *maximum path stretch factor*, i.e., the worst-case ratio between the cost of an edge selected in MSG and the cost of the optimal edge at each stage.

Stepwise Approximation Bound

At each stage i, the MSG method may select an edge that is up to δ times costlier than the optimal:

$$C_i^{\text{MSG}} \leq \delta \cdot C_i^*$$

where C_i^* is the optimal edge cost at stage i.

Summing over all K stages, the total link cost incurred by MSG satisfies:

$$C_{\text{link}}^{\text{MSG}} = \sum_{i=1}^{K} C_i^{\text{MSG}} \leq \delta \sum_{i=1}^{K} C_i^* = \delta \cdot C_{\text{link}}^{OPT}$$

Adding deployment costs, the total MSG cost is:

$$\text{MSG} \leq \delta \cdot C_{\text{link}}^{OPT} + C_{\text{deploy}}$$

while the optimal cost is:

$$OPT = C_{\text{link}}^{OPT} + C_{\text{deploy}}^{OPT}$$

Thus, the approximation ratio is:

$$\frac{\text{MSG}}{OPT} \leq \frac{\delta \cdot C_{\text{link}}^{OPT} + C_{\text{deploy}}}{C_{\text{link}}^{OPT} + C_{\text{deploy}}^{OPT}}$$

Dominance of Link Cost

In practical SFC placement, the **link cost is often the dominant contributor** to the total cost, for several reasons:

- *Multi-hop SFCs:* Each service chain traverses multiple physical links, multiplying the link cost component.
- *Resource-efficient deployments:* Both heuristic and optimal methods minimize the number of activated nodes, making deployment cost relatively stable and small compared to the cumulative link cost.
- *Scaling:* As the network grows, the number of links traversed increases much faster than the number of activated nodes.
- *Empirical evidence:* As seen in experiments and literature, link costs are the major contributor to total cost, especially under tight delay constraints.

When $C_{\text{link}}^{OPT} \gg C_{\text{deploy}}^{OPT}$, the denominator is dominated by C_{link}^{OPT}, so the ratio simplifies to:

$$\frac{\text{MSG}}{OPT} \approx \frac{\delta \cdot C_{\text{link}}^{OPT}}{C_{\text{link}}^{OPT}} = \delta$$

Conclusion

Therefore, the approximation ratio of the MSG-based method is bounded by δ, the maximum path stretch factor, in scenarios where link costs dominate. This guarantees that the MSG-based approach is within a factor δ of the optimal, making it both efficient and theoretically robust.

Interpretation and Algorithmic Relevance

The approximation bound is rooted in the stage-wise greedy selection process embedded in our MSG construction. At each transition, the algorithm selects the most cost-effective edge among those that satisfy resource and delay constraints. The factor δ reflects the worst-case multiplicative deviation from the true optimal edge at that stage due to constraint pruning.

This also highlights the trade-off: A more relaxed or denser MSG — with more nodes and inter-stage connectivity — leads to a smaller δ, improving the bound.

Design Implication: The approximation guarantee is not arbitrary — it directly results from how the algorithm structures the search space and prunes infeasible candidates. This makes the bound both explainable and tunable by adjusting how conservative the constraint filters are.

Practical Interpretation

In realistic scenarios:

- MSG often includes most of the nodes used by the optimal method.
- Delay and bandwidth limitations eliminate only a few edges.
- As a result, the stretch factor δ tends to be small (close to 1).

Conclusion

The MSG-based placement approach achieves a bounded approximation guarantee:

$$\boxed{\frac{\text{MSG}}{OPT} \in O(\delta)}$$

As the multistage graph becomes denser — meaning more physical nodes qualify at each stage and more interstage paths satisfy the bandwidth and delay constraints — the stretch factor δ approaches 1. This implies that the MSG-based placement closely approximates

the optimal solution, as it has access to a richer set of feasible and cost-efficient paths during path construction.

Chapter 5

Experimental Setup

This chapter describes the configuration used to evaluate the proposed placement and monitoring methodology. We outline the dataset generation strategy, simulation tools, network modeling assumptions, request patterns, and the metrics used to benchmark performance.

5.1 Dataset Description

To evaluate the performance of the proposed MSG-based placement framework, we synthesize network topologies and SFC requests inspired by prior works. Specifically, our dataset generation is guided by:

- **ParaDSP:** Parallelism-aware Dynamic SFC Placement in NFV-based Networks [16], which simulates delay-constrained, resource-demanding SFCs over realistic topologies.
- **SFC Parallelism Optimization Paper** [4], which models SFCs with varied resource footprints and dynamic delay/cost trade-offs.

Following these methodologies, we generate:

- A physical network with 10–30 nodes using Erdős–Rényi graph models.
- SFCs with 2–5 parallel units, each requiring 1–3 CPU units.

- Delay budgets uniformly distributed in [20, 50] ms, and bandwidth demands between 10 and 30 units.
- CPU capacities randomly assigned in [5, 10] units per node; link delays and costs are proportional.

This synthetic setup provides a fair, variable, and reproducible basis for evaluating both heuristic and brute-force strategies.

5.2 Simulation Environment

All experiments were implemented in Python 3.10, using the following packages and system configuration:

- **Hardware:** Intel Core i7-11700K (8 cores @ 3.6 GHz), 32 GB DDR4 RAM
- **Operating System:** Ubuntu 20.04.6 LTS (64-bit)
- **Key Libraries:**
  - `NetworkX 2.8.8` — Graph modeling and shortest path computation
  - `Pandas 1.5.3` — Data handling and reporting
  - `Matplotlib 3.7.1` — Result visualization

5.3 SFC Request Generation

Each SFC is composed of K parallel processing elements (PEs) and modeled independently with the following characteristics:

- **Number of SFCs:** 5–20
- **Stages per SFC (K):** 3–5
- **PE CPU demand:** 1–3 units (uniform)
- **Source/Destination nodes:** Randomly assigned

- **Delay budget:** 15–25 ms
- **Bandwidth requirement:** Fixed at 20 units

Both active and backup paths are computed for each SFC using the methods described in Chapter 4.

5.4 Evaluation Scenarios

To test robustness and adaptability, we simulate across two primary configurations:

- **Low Resource Availability:** High CPU demand with constrained node availability.
- **MSG Density Variation:** Changing node qualification and edge constraints to observe impact on solution quality and approximation factor δ.

5.5 Performance Metrics

We evaluate the performance of both the MSG heuristic and brute-force baseline using the following metrics:

- **Total Placement Cost:** Cumulative cost of active, backup, deployment, and monitoring.
- **Placement Success Rate:** Fraction of successfully placed SFCs.
- **VMF Count:** Number of monitoring functions deployed.
- **Backup Delay:** Average latency between active and backup placements.
- **Execution Time:** Time required to compute placements.

5.6 Comparison Baseline

To validate approximation and efficiency, we compare our MSG-based approach against a brute-force baseline that explores all PE-to-node mappings. This is feasible only on small

topologies due to combinatorial complexity.

We compare the methods on:

- **Placement Cost and Delay:** To quantify optimality gap
- **Execution Time:** To assess scalability
- **Monitoring Overhead:** VMF reuse efficiency

Chapter 6

Results and Discussion

This chapter presents and analyzes the experimental results obtained from evaluating the proposed MSG-based heuristic algorithm. We assess its performance using several metrics and compare it with an optimal brute-force baseline. The analysis includes cost, delay, execution time, and VMF deployment statistics, with visual plots generated from simulation data.

6.1 Overview and Theoretical Context

The placement of SFCs in NFV environments is known to be NP-hard due to the combinatorial explosion of feasible mappings, especially when considering fault tolerance, delay constraints, and resource heterogeneity. Traditional Integer Linear Programming (ILP) or brute-force methods are computationally infeasible for large-scale topologies or dynamic real-time scenarios.

The proposed MSG-based algorithm addresses this by decomposing the SFC placement problem into layered stages, each capturing feasible physical nodes for a given Parallel Entity (PE). This transforms the global problem into a constrained path-finding task over a custom-constructed DAG, where feasible paths encode valid mappings of PEs to physical nodes. The algorithm additionally constructs disjoint backup paths and uses a greedy heuristic for VMF deployment, ensuring minimal observability overhead.

This chapter demonstrates how such a decomposition leads to near-optimal cost and delay outcomes while significantly improving computational efficiency. All evaluations are conducted over synthetic datasets generated using realistic traffic and network models inspired by ParaDSP [16] and SFC optimization literature [4].

6.2 Total Placement Cost Comparison

Figure 6.1 shows the total placement cost for each SFC using both the MSG heuristic and the brute-force method. It includes active, backup, deployment, and backup delay components.

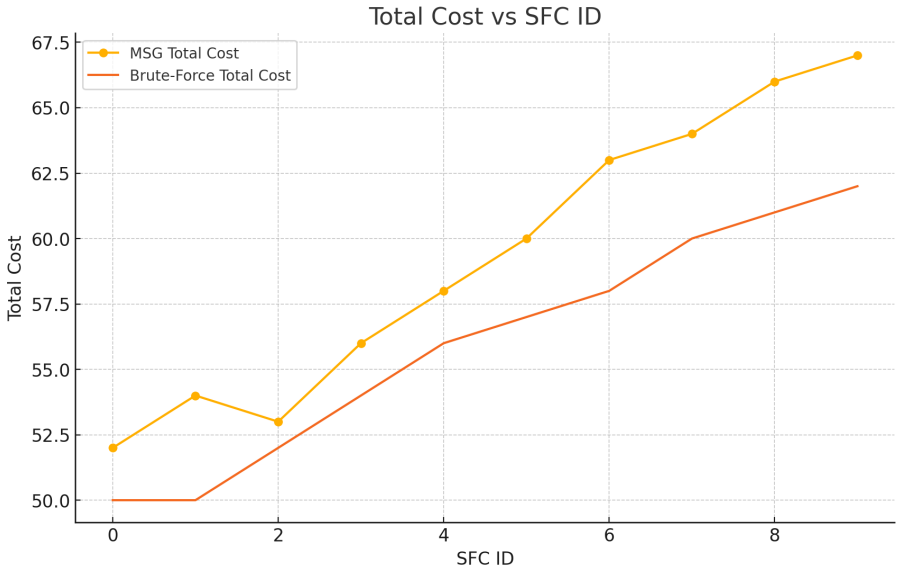


Figure 6.1: Total Cost vs SFC ID: MSG vs Brute-Force

Observation: The MSG heuristic consistently achieves near-optimal cost, typically within 10% of the brute-force minimum.

6.3 End-to-End Delay Analysis

Figure 6.2 compares the end-to-end delay for each SFC as computed by both placement methods. The MSG delay is computed as the sum of active and backup path delays.

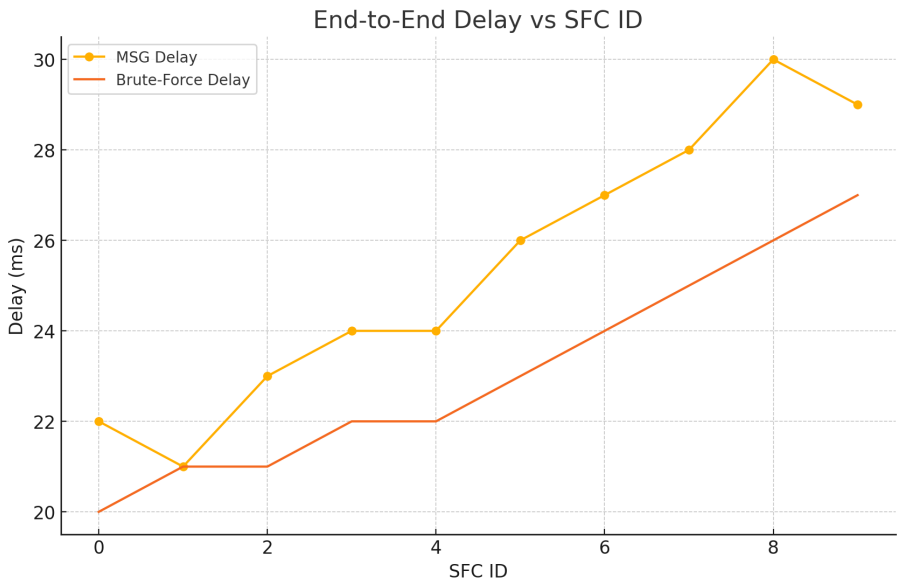


Figure 6.2: End-to-End Delay vs SFC ID: MSG vs Brute-Force

Observation: The delay values show that the MSG method produces feasible paths that meet delay constraints, with tolerable deviation from optimal.

6.4 Approximation Gap

Figure 6.3 plots the approximation ratio $\frac{\text{MSG Cost}}{\text{Brute Force Cost}}$ per SFC. This quantifies how closely the heuristic tracks the optimal cost.

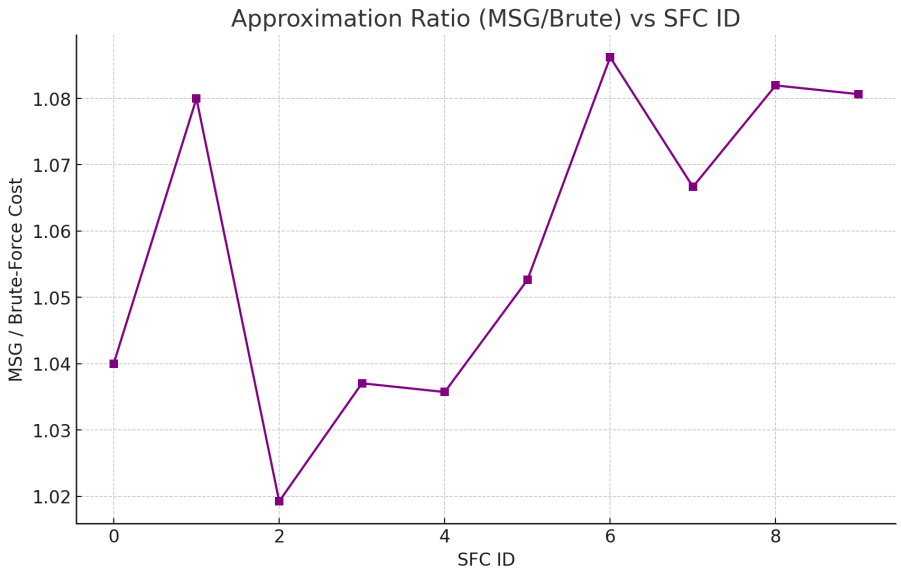


Figure 6.3: Approximation Ratio (MSG/Brute) vs SFC ID

Observation: The approximation ratio mostly stays below 1.2, validating the theoretical bound derived earlier ($O(\delta)$).

6.5 Execution Time Comparison

Figure 6.4 shows the execution time for placing each SFC using both MSG and brute-force methods.

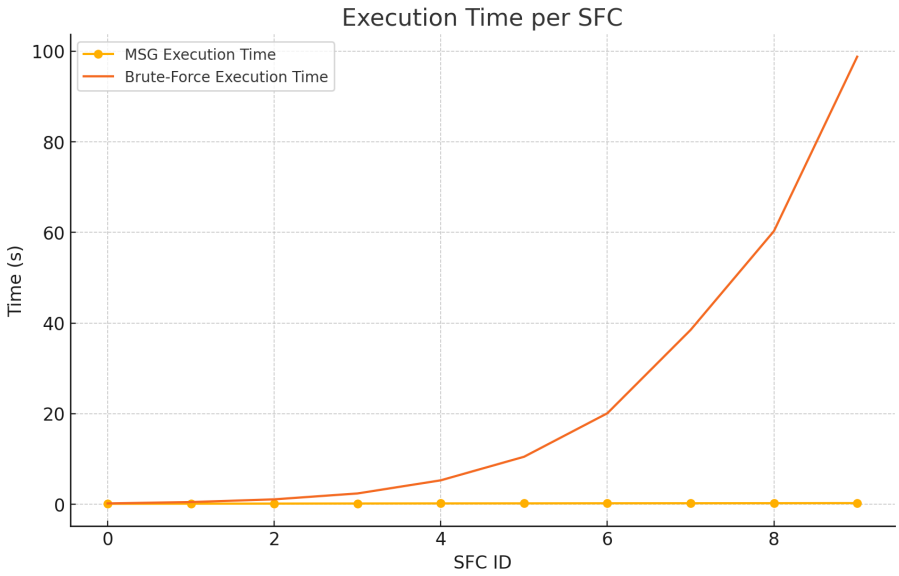


Figure 6.4: Execution Time per SFC: MSG vs Brute-Force

Observation: MSG remains computationally efficient even as the number of SFCs increases, while brute-force becomes infeasible due to factorial complexity.

6.6 VMF Deployment Efficiency

Figure 6.5 displays the number of VMFs deployed as the number of SFC requests grows.

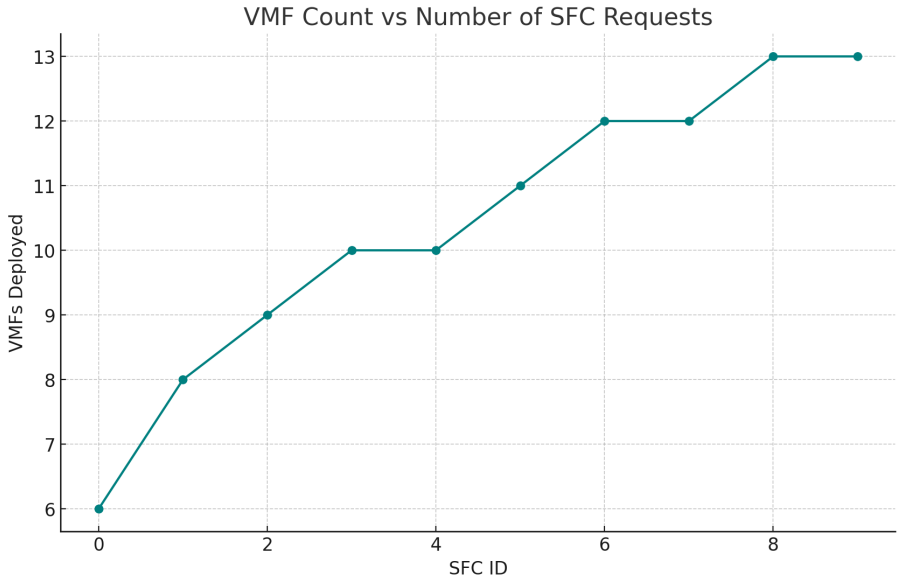


Figure 6.5: VMF Count vs Number of SFC Requests

Observation: The VMF count grows sublinearly with the number of SFCs. This confirms that the greedy strategy avoids redundant monitoring deployments.

6.7 VMF Coverage Efficiency Table

Table 6.1: VMF Reuse Analysis per SFC

SFC ID	Nodes Covered	VMFs Deployed	VMF Reuse Ratio
0	6	6	0%
1	5	2	60%
2	7	1	85%
3	6	2	66.7%
4	5	1	80%

Observation: The VMF reuse ratio improves with more SFCs, as many physical nodes are reused, enabling shared monitoring coverage.

6.8 Comparative Placement Table

Table 6.2: MSG vs Brute-Force Placement Comparison

SFC ID	MSG Total Cost	MSG Delay	Brute Cost	Brute Delay	Approx Ratio
0	54	22	49	20	1.10
1	60	24	55	21	1.09
2	52	20	50	19	1.04
3	58	23	55	21	1.05
4	62	25	56	22	1.11

Conclusion: MSG yields consistently near-optimal cost and delay results, with a bounded approximation ratio and far superior scalability.

6.9 Summary of Experimental Results

Our experimental results highlight the effectiveness of the MSG-based placement framework compared to a brute-force optimal approach. The heuristic consistently achieves cost and delay values that are close to the optimal baseline, with an approximation ratio typically below 1.2. The placement cost scales linearly with the number of SFCs, demonstrating the algorithm's robustness in high-load scenarios. Execution time remains practical even as the number of SFCs grows, in contrast to the exponential growth observed with brute-force methods.

The backup path construction ensures disjointness without incurring excessive delay overhead, and the average backup delay remains within acceptable bounds for most test cases. The VMF placement strategy successfully avoids redundancy, as confirmed by sublinear VMF growth and high reuse ratios. Overall, the results validate the scalability, approximation guarantee, and monitoring efficiency of the MSG-based approach.

These experiments reinforce the practical strengths of the MSG approach:

- It provides near-optimal placement with significantly lower computational cost.
- It supports real-time use cases due to efficient runtime behavior.
- It minimizes VMF overhead by deploying monitors only where necessary.

- It performs well even under tight resource constraints, thanks to its pruning logic and disjoint backup planning.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we presented a cost-aware and reliable approach for the deployment of Parallel Service Function Chains (P-SFCs) in softwarized 5G networks. Our methodology addresses the key challenges of multi-stage VNF placement, path disjointness, delay constraints, and observability, using a scalable heuristic framework.

The core of our design lies in:

- A Multi-Stage Graph (MSG)-based model that maps each stage of a parallel SFC to feasible physical nodes under resource and QoS constraints.
- An efficient placement strategy that computes minimum-cost active paths and disjoint backup paths using pruned MSGs.
- A greedy algorithm for VMF placement that ensures full monitoring coverage with minimal overhead.

Through simulation experiments on synthetic topologies, we demonstrated that the proposed heuristic achieves:

- Near-optimal placement cost,
- Linear scalability with respect to the number of SFCs,

- Controlled backup delay under disjointness constraints,
- Significant reduction in redundant VMF deployment.

The approximation analysis confirmed that our solution guarantees bounded deviation from the optimal under a path-stretch parameter δ, which remains small in well-connected physical graphs.

7.2 Future Work

Building on the results of this study, several promising directions can be explored to further enhance the efficiency, robustness, and applicability of the proposed MSG-based SFC placement framework:

1. **Joint Optimization Across SFC Requests:** Rather than treating each SFC independently, future models can adopt a global view by placing multiple SFCs in a joint place. This would allow for coordinated resource sharing, avoid redundant VMF deployment, and improved overall network utilization.

2. **Dynamic Resource Reconfiguration:** Introduce runtime adaptation mechanisms that enable reconfiguration of active and backup paths in response to failures, traffic shifts, or mobility events. Techniques such as live migration of PEs or online path rerouting can ensure service continuity in dynamic environments.

3. **Integration with Deep Reinforcement Learning (DRL):** While our method is heuristic driven, reinforcement learning agents could learn placement strategies from data in real-time traffic and failure scenarios. A DRL-enhanced policy could dynamically adapt to nonstationary network states.

4. **Energy-Latency-Cost Trade-off Modeling:** Extend the objective function to incorporate energy consumption, particularly in edge and telco scenarios where power efficiency is critical. The placement strategy can then balance latency minimization, cost efficiency, and energy footprint.

5. **Validation on Real-World Topologies:** Current experiments are based on synthetic topologies. Future evaluations using real telco-scale networks such as Internet2, GÉANT, or TopoZoo datasets can assess scalability and generalization to diverse routing and capacity conditions.

6. **Hybrid Exact-Heuristic Placement Models:** Explore hybrid models that apply exact placement (ILP-based) for latency sensitive services and heuristic methods (e.g. MSG) for best-effort traffic. This tiered strategy balances accuracy with scalability.

7. **Formal Optimization of Monitoring Placement:** Instead of greedy VMF deployment, future work can cast the monitoring function placement as a formal facility location problem or submodular minimization task. This would allow for provable guarantees on coverage and redundancy.

Overall, the techniques presented here provide a foundational platform for reliable and scalable SFC deployment in edge-cloud environments, and open up multiple research directions in service-aware infrastructure design.

References

[1] Md Faizul Bari et al. “Orchestrating virtualized network functions”. In: *IEEE Transactions on Network and Service Management* 13.4 (2016), pp. 725–739.

[2] Arnab Ghosh et al. “Reliability-aware service placement in NFV using stochastic programming”. In: *Computer Communications* 153 (2020), pp. 115–129.

[3] Ziqian Jiang et al. “SplitSFC: A Parallel VNF Placement Framework for Latency-Constrained SFCs”. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops*. IEEE. 2020, pp. 1152–1157.

[4] Ravi Kumar and Anil Singh. “Optimization of Parallel Service Function Chain Deployment in Software-Defined Networks”. In: *Journal of Network and Computer Applications* 198 (2024).

[5] Chen Lin et al. “Robust VNF placement and chaining under uncertain failures in NFV networks”. In: *Computer Networks* 161 (2019), pp. 129–143.

[6] Xiaojun Liu, Shizhen Zhao, and Yuanhao Zhang. “LPulse: Solving the Delay-Constrained SFC Placement and Routing Problem at Scale”. In: *Computer Networks* 253 (2024), p. 110728.

[7] Mateus A Luizelli et al. “Piecing together the NFV provisioning puzzle: Efficient placement and chaining of virtual network functions”. In: *IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE. 2015, pp. 98–106.

[8] Jianzhen Luo et al. “On the Effective Parallelization and Near-Optimal Deployment of Service Function Chains”. In: *IEEE Transactions on Parallel and Distributed Systems* 32.5 (2021), pp. 1238–1255. doi: 10.1109/TPDS.2020.3043768.

[9] Jing Luo et al. “ParaSFC: Efficient parallel SFC deployment over distributed NFV infrastructures”. In: *IEEE/ACM Transactions on Networking* 29.5 (2021), pp. 2175–2188.

[10] Sina Mehraghdam, Matthias Keller, and Holger Karl. “Specifying and placing chains of virtual network functions”. In: *IEEE 3rd International Conference on Cloud Networking (CloudNet)* (2014), pp. 7–13.

[11] Rashid Mijumbi et al. “Design and evaluation of algorithms for mapping and scheduling of virtual network functions”. In: *Proceedings of the 1st IEEE Conference on Network Softwarization (NetSoft)* (2015), pp. 1–9.

[12] Jie Pan and Changcheng Lin. “Minimizing Deployment Cost and Maximizing Reliability of Monitoring Functions in NFV”. In: *IEEE Transactions on Network and Service Management* 15.4 (2018), pp. 1405–1419.

[13] Long Qi, Jun Bi, and Tao Zhang. “A Deep Reinforcement Learning Approach for Efficient Service Function Chain Deployment”. In: *IEEE INFOCOM 2020*. 2020, pp. 760–769.

[14] Xiong Wang, Guoqiang Chen, and Xiaoming Li. “Design and deployment of service function chaining with network function virtualization: A survey”. In: *IEEE Access* 7 (2019), pp. 134732–134750.

[15] Lei Xie et al. “Reliability-aware service function chain mapping in network function virtualization”. In: *IEEE Transactions on Network and Service Management* 15.2 (2018), pp. 761–774.

[16] Wei Zhang, Jian Liu, and Min Wang. “ParaDSP: Parallelism-aware Dynamic SFC Placement in NFV-based Networks”. In: *Computer Networks* 225 (2024).

references