People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

University of Mila | **جامعة ميلة**

Faculty: Mathematics and Computer Sciences
Order N° : …….
Registration number: …….

Department: Computer Science
Field: Computer Science
Specialty: Networks and Distributed Systems

Doctoral Thesis

For Obtaining the 3rd cycle LMD Doctorate Degree in

Computer Science

SDN-based Solutions for Improving Network Performance

A thesis presented by:

Raid BOUDI

Thesis defended on January 24, 2026, before the jury composed of:

Mounira	BOUZAHZAH	**MCA.** Abdelhafid BOUSSOUF University of Mila, Algeria	President
Nardjes	BOUCHEMAL	**MCA.** Abdelhafid BOUSSOUF University of Mila, Algeria	Supervisor
Mohammed	LALOU	**MCA.** University BURGUNDY Europe - France	Co-supervisor
Chirihane	GHERBI	**MCA.** Ferhat Abbas University of Setif 1, Algeria	Examiner
Madjed	BENCHEIKH LEHOCINE	**MCA.** Abdelhafid BOUSSOUF University of Mila, Algeria	Examiner
Aissa	BOULMERKA	**Prof.** National School of Artificial Intelligence, Algeria	Examiner

2025-2026

Abstract

The integration of Software-Defined Networking (SDN) into the Internet of Things (IoT) offers a promising solution to address the growing challenges of scalability, complexity, and flexibility in modern networks. However, this convergence introduces significant management challenges, particularly regarding performance, latency reduction, and resilience. A critical issue is the Controller Placement Problem (CPP), which directly impacts the efficiency of Software-Defined Internet of Things (SD-IoT) networks and becomes more complex in heterogeneous environments with dynamic traffic and load variations, especially in mission-critical applications like healthcare and autonomous transport.

This thesis proposes a set of SDN-based approaches to improve Software-Defined IoT (SD-IoT) network performance by addressing the Controller Placement Problem (CPP). First, the Heterogeneous Traffic Flow–based Controller Placement (HTF-CP) method optimizes controller locations by accounting for traffic diversity under flexible deployment assumptions. Second, a population-aware SD-IoT architecture for the Algerian network is introduced, combining K-means clustering with enhanced K-center and K-median algorithms to incorporate population-weighted traffic demands. Third, a Weighted Betweenness Centrality–based approach (WBC-CPP) identifies influential nodes to enable efficient and load-aware controller placement with reduced search space. This approach is further enhanced through the integration of Grey Wolf Optimization (GWO), resulting in the GWO-WBC-CPP method, which considers the dynamic nature of IoT traffic over time. Finally, the Particle Swarm Optimization with Heterogeneous Switch Load (PSO-HSL) method applies a meta-heuristic strategy to jointly optimize latency and reliability while avoiding exhaustive search.

The effectiveness of these approaches is validated through extensive simulations on realistic network topologies (Internet Topology Zoo and OS3E), a newly designed population-aware Algerian network, and synthetic topologies for scalability analysis. The results show notable performance gains in terms of latency, load balancing, reliability, and overall network efficiency when compared to conventional CPP and baseline SDN placement methods.

Keywords: Software-Defined Internet of Things, Controller Placement Problem, Heterogeneous Switch Load, Dynamic traffic, Network performance.

Résumé

L'intégration du réseau défini par logiciel (SDN) dans l'Internet des Objets (IoT) offre une solution prometteuse pour relever les défis croissants de l'évolutivité, de la complexité et de la flexibilité des réseaux modernes. Cependant, cette convergence introduit des défis de gestion significatifs, notamment en ce qui concerne les performances, la réduction de la latence et la résilience. Un problème critique est le problème de placement des contrôleurs (CPP), qui impacte directement l'efficacité des réseaux IoT définis par logiciel (SD-IoT) et devient plus complexe dans des environnements hétérogènes avec un trafic dynamique et des variations de charge, en particulier dans les applications critiques comme la santé et le transport autonome.

Cette thèse propose un ensemble d'approches basées sur le SDN visant à améliorer les performances des réseaux SD-IoT en traitant le problème de placement des contrôleurs. Dans un premier temps, la méthode de Placement de Contrôleurs basé sur les Flux de Trafic Hétérogènes (HTF-CP) optimise la position des contrôleurs en tenant compte de l'hétérogénéité des flux de trafic et d'une installation flexible des contrôleurs dans le réseau. Ensuite, une architecture SD-IoT sensible à la population est proposée pour le réseau algérien, en combinant le clustering K-means avec des versions améliorées des algorithmes K-center et K-median, afin d'intégrer des demandes de trafic pondérées par la population. Par ailleurs, une approche basée sur la centralité d'intermédiarité pondérée (WBC-CPP) est introduite pour identifier les nœuds les plus influents et permettre un placement efficace et équilibré des contrôleurs tout en réduisant l'espace de recherche. Cette approche est ensuite renforcée par l'intégration de l'algorithme Grey Wolf Optimization (GWO), donnant naissance à la méthode GWO-WBC-CPP, qui prend également en compte le caractère dynamique du trafic IoT au niveau des commutateurs. Enfin, la méthode d'Optimisation par Essaims de Particules avec Charge Hétérogène des Commutateurs (PSO-HSL) introduit une approche méta-heuristique qui optimise le placement des contrôleurs à la fois pour la latence et la fiabilité, tout en évitant la surcharge de la recherche exhaustive.

Les performances des approches proposées sont évaluées à l'aide de simulations sur des topologies réalistes (Internet Topology Zoo et OS3E), un nouveau réseau algérien sensible à la population, ainsi que des topologies synthétiques pour analyser la scalabilité. Les résultats obtenus montrent des améliorations notables en termes de latence, d'équilibrage de charge, de fiabilité et de performance globale du réseau, par rapport aux méthodes classiques de placement des contrôleurs et aux approches SDN de référence.

Mots-clés : Internet des Objets Défini par Logiciel, Problème de Placement des Contrôleurs, Charge Hétérogène des Commutateurs, Trafic dynamique, Performance du réseau.

ملخص

يُعَدّ دمج الشبكات المعرفة برمجيًا (SDN) في إنترنت الأشياء (IoT) حلًا واعدًا لمواجهة التحديات المتزايدة المتعلقة بقابلية التوسع والتعقيد والمرونة في الشبكات الحديثة. ومع ذلك، فإن هذا التقارب يُدخل تحديات كبيرة في إدارة الشبكات، خصوصًا فيما يتعلق بالأداء وتقليل زمن الاستجابة (الكمون) وضمان المرونة. وتُعتبر مشكلة توزيع وحدات التحكّم (CPP) من القضايا الحرجة التي تؤثر مباشرة على كفاءة شبكات إنترنت الأشياء المعرفة برمجيًا(SD-IoT) ، وتزداد تعقيدًا في البيئات غير المتجانسة ذات حركة مرور ديناميكية وتغيرات في الحمل، خاصة في التطبيقات الحساسة مثل الرعاية الصحية والنقل الذاتي.

تُقدِّم هذه الأطروحة مجموعة من المقاربات المعتمدة على الشبكات المعرَّفة برمجيًا (SDN) بهدف تحسين أداء شبكات إنترنت الأشياء المعرَّفة برمجيًا(SD-IoT) ، من خلال معالجة مشكلة تموضع وحدات التحكّم(CPP) Controller Placement Problem. أولًا، يتم اقتراح طريقة تموضع وحدات التحكّم المعتمدة على تدفقات المرور غير المتجانسة(HTF-CP) ، والتي تعمل على تحسين مواقع وحدات التحكّم من خلال أخذ تنوّع تدفقات المرور بعين الاعتبار، مع افتراض إمكانية النشر المرن لوحدات التحكّم عبر فضاء الشبكة. ثانيًا، تُقدَّم بنية SD-IoT مراعية للكثافة السكانية في الشبكة الجزائرية، حيث يتم الدمج بين خوارزمية التجميع K-means وخوارزميتي K-center و K-median المُحسَّنتين، مع تضمين متطلبات المرور الموزونة حسب عدد السكان لتحقيق توزيع أمثل لوحدات التحكّم. ثالثًا، يتم اقتراح مقاربة قائمة على المركزية البينية الموزونة(WBC-CPP) ، تهدف إلى تحديد العُقد الأكثر تأثيرًا في الشبكة من أجل تمكين تموضع فعّال لوحدات التحكّم يراعي الحمل، مع تقليص فضاء البحث. ويتم تعزيز هذه المقاربة لاحقًا من خلال دمجها مع خوارزمية تحسين قطيع الذئاب الرمادية(GWO) ، مما ينتج عنه الأسلوبGWO-WBC-CPP ، والذي يأخذ في الحسبان الطبيعة الديناميكية لتدفّقات مرور إنترنت الأشياء عبر الزمن. وأخيرًا، يتم تقديم طريقة تحسين سرب الجسيمات مع الحمل غير المتجانس للمحوّلات (PSO-HSL) كنهج ميتا-استكشافي يهدف إلى التحسين المشترك لكل من الكمون والموثوقية، مع تجنّب التكلفة العالية للبحث الشامل.

تم التحقق من فعالية هذه المقاربات من خلال محاكاة مكثفة على طوبولوجيات شبكية واقعية مثل Internet Topology Zoo و OS3E، إضافة إلى شبكة جزائرية جديدة مراعية للكثافة السكانية، إلى جانب طوبولوجيات اصطناعية لتقييم قابلية التوسع. وتُظهر النتائج المحصَّلة تحسينات ملحوظة من حيث تقليل الكمون، وتحقيق توازن الحمل، وزيادة الموثوقية، وتحسين الكفاءة العامة للشبكة، مقارنةً بطرائق CPP التقليدية وأساليب تموضع وحدات التحكّم المعتمدة على SDN.

الكلمات المفتاحية: إنترنت الأشياء المعرف بالبرمجيات(SD-IoT) ، مشكلة تموضع المتحكمات(CPP) ، الأحمال غير المتجانسة للمفاتيح، حركة المرور الديناميكية، أداء الشبكة.

Acknowledgements

In the name of **Allah**, the Most Gracious, the Most Merciful. All praise is due to **Allah**, Lord of the worlds. I thank Allah for granting me the strength, patience, and clarity needed to complete this work. Without His guidance and blessings, none of this would have been possible.

I would like to express my deepest gratitude to my supervisor, **Dr. Bouchemal Nardjes**, and my co-supervisor, **Dr. Lalou Mohammed**, for their continuous guidance, scientific insight, and unwavering support throughout the preparation of this thesis. Their encouragement, availability, and patience have greatly contributed to the advancement and completion of this research.

My sincere thanks go to the honorable members of the jury, **Dr. Gherbi Chirihane**, my former teacher whose impact on my learning journey remains profound, as well as **Dr. Madjed Bencheikh Lehocine**, **Dr. Aissa Boulmerka**, and **Dr. Mounira Bouzahzah**, President of the jury, for accepting to evaluate my work and for the time and expertise they dedicated to improving this manuscript.

I extend my heartfelt gratitude to my family, the source of my strength and resilience. To my **mother**, whose sacrifices, unconditional love, and constant dedication gave us life and hope you gave everything without expecting anything in return except our happiness and success. To my **father**, who offered his support, his patience, and his infinite generosity thank you for giving everything and asking for nothing but our wellbeing.

My deepest appreciation goes to my beloved **older sister**, my second mother, who gave me everything I needed support, love, encouragement, and presence. Her strength has always guided me, and her sons, **Nino** and **Didi**, brought joy and motivation during the most challenging moments.

I am also grateful to my **brother** for his precious help in situations where knowledge went beyond what is taught in school, and for always being there with understanding and support. My thanks also go to his child, **Anes**, whose smile and innocence brought warmth and energy to my journey.

To my little sister, thank you for every cup of coffee you prepared for me, for your presence, and for the small but meaningful gestures that accompanied me throughout this adventure.

Finally, I would like to thank my friends, and every person who believed in me, supported me, and encouraged me. Your trust, kindness, and positive words carried me through difficult moments and made this achievement possible.

To all of you, from the bottom of my heart thank you.

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols

AI Artificial Intelligence

ANP Analytic Network Process

ANDP Analytical Network Decision Making Process

AVOA African Vultures Optimization Algorithm

CCPP Capacitated Controller Placement Problem

CD Community Detection

CNPA Clustering-Based Network Partition Algorithm

CPP Controller Placement Problem

CP Controller Placement

DEWO Differential Evolution and Whale Optimization

ESFO Enhanced SunFlower Optimization

FFA Firefly Algorithm

GA Genetic Algorithm

GWO Grey Wolf Optimizer

HC Hierarchical Clustering

HSL-CP Heterogeneous Switch Load–based Controller Placement

ILP Integer Linear Programming

IQP Integer Quadratic Programming

IoT Internet of Things

ITZ Internet Topology Zoo

MCDM Multi-Criteria Decision-Making

MILP Mixed Integer Linear Programming

MIP Mixed Integer Programming

NBI Northbound Interface

NDN Named Data Networking

PAM Partitioning Around Medoids

POCO Pareto Optimal Controller Placement

PSO Particle Swarm Optimization

QoS Quality of Service

RALO Reliability-Aware and Latency-Oriented controller placement algorithm

RLMD Reliable and Load balance-aware Multi-controller Deployment

SA Simulated Annealing

SBI Southbound Interface

SD-IoT Software-Defined Internet of Things

SDN Software-Defined Networking

SD-SatNet Software-Defined Satellite Network

UCPP Uncapacitated Controller Placement Problem

US-ML Unsupervised Machine Learning

VANET Vehicular Ad Hoc Network

WBC Weighted Betweenness Centrality

WMN Wireless Mesh Network

WSN Wireless Sensor Network

General Introduction

Software-Defined Networking (SDN) has emerged as a transformative paradigm in network management by separating the control plane from the data plane, enabling centralized and programmable network control. This separation provides flexibility, scalability, and ease of management compared to traditional architectures, fostering innovative solutions for performance optimization and supporting a wide range of applications.

Several SDN-based solutions have been developed to enhance network performance through traffic engineering, dynamic resource allocation, and automated management. These solutions not only improve quality of service but also ensure efficient use of resources in complex and large-scale networks. By leveraging SDN, networks can be adapted dynamically to varying traffic demands, providing better support for latency-sensitive, high-throughput, and mission-critical services.

Problem Statement

With the advent of the Internet of Things (IoT), the number of connected devices has grown exponentially, giving rise to the Software-Defined Internet of Things (SD-IoT). SD-IoT combines the programmability of SDN with the heterogeneity and massive scale of IoT, offering more efficient management and coordination of devices. However, the integration of SDN into IoT networks presents several challenges, including scalability, dynamic traffic patterns, and maintaining low latency across highly distributed devices. One of the most critical issues in SD-IoT is the Controller Placement Problem (CPP). Since controllers act as the "brain" of the SDN, their location significantly impacts network performance, particularly in IoT environments where delays and reliability directly affect service quality. Poor placement may lead to increased latency, bottlenecks, or controller overload, undermining the effectiveness of SD-IoT. Therefore, addressing CPP is essential for ensuring robust, scalable, and high-performing IoT networks.

Goals and Contributions

While many studies on CPP have been conducted in the context of wide-area networks (WANs), relatively fewer works have addressed this problem in IoT environments, which represent one of the most promising yet challenging domains. To bridge this gap, this thesis presents innovative SDN-based solutions adapted to the distinctive features of IoT environments. The main contributions are centered on two fundamental dimensions:

Heterogeneous traffic flows. We first examine the impact of varying flow rates generated by IoT devices. Initially, we evaluate this with a single deployed controller, and later we generalize to multiple controllers.

Dynamic traffic variation over time. We integrate temporal variations in flow to capture real IoT behavior and improve metrics such as latency, flow rate, and execution time.

To this end, four complementary solutions are proposed. First, the Controller Placement (CP) problem is addressed through traffic-aware optimization based on flow weights, initially allowing flexible controller deployment and subsequently restricting placement to switch nodes with support for multiple controllers. Second, a Weighted Betweenness Centrality (WBC)-based approach is introduced to improve placement efficiency by identifying influential nodes, although scalability limitations remain. Third, this approach is further extended by integrating meta-heuristic optimization techniques and incorporating dynamic flow models to better capture the temporal variability of IoT traffic. Finally, network reliability is explicitly considered as an optimization objective by accounting for controller failures, thereby enhancing network resilience and overall performance.

All chapters in this dissertation are based on manuscripts of our articles published in scientific journals. The thesis is organized into seven chapters, divided into two main parts: Background and Contributions. The Background part introduces the context of SDN and SD-IoT, presents the CPP, reviews existing approaches along with their limitations and challenges, and details the system modeling and problem formulation. The Contributions part presents our proposed solutions to address CPP in SD-IoT networks, focusing on heterogeneous switch traffic awareness, centrality-based controller placement, dynamic traffic adaptation over time, meta-heuristic approaches for reducing computational complexity, and reliability optimization. The overall structure of the dissertation is as follows:

- **Chapter 1:** Introduces the fundamentals of SDN and IoT, their convergence into SD-IoT, its architecture, key characteristics, applications, and open research challenges.

- **Chapter 2:** Reviews the Controller Placement Problem, analyzes existing surveys, classifies placement methods, and highlights limitations and challenges in current works.

- **Chapter 3:** Describes the system model and outlines the problem formulation, defining CPP complexity, optimization aspects, assumptions, performance metrics, and network topologies for evaluation.

- **Chapter 4:** Explores exact controller placement strategies, including free-space optimization and topology-aware designs, evaluated on uniform and weighted topologies using methods like K-center and K-median.
- **Chapter 5:** Proposes a WBC-based placement approach, focusing on candidate switch selection to minimize latency and improve flow distribution, with detailed performance analysis.
- **Chapter 6:** Integrates Grey Wolf Optimization (GWO) with the WBC approach to address dynamic IoT traffic, evaluating performance under multiple traffic models and resilience scenarios.
- **Chapter 7:** Introduces a Particle Swarm Optimization with Heterogeneous Switch Load (PSO-HSL) strategy, analyzing latency, flow rate, reliability, scalability, and execution time in high-density SD-IoT networks.

Part I

Background

1

Software-Defined Internet of Things: an overview

1.1 Introduction

The explosive growth of the Internet of Things (IoT) has introduced unprecedented complexity in modern network design [1]. Traditional networking models characterized by rigid, vertically integrated architectures struggle to accommodate the scale, heterogeneity, and real-time demands of IoT environments [2–6]. As IoT continues to expand across domains such as smart cities, healthcare, industry, and transportation, the need for intelligent, scalable, and programmable network management becomes increasingly urgent [7].

In this context, Software-Defined Networking (SDN) has emerged as a transformative paradigm, offering centralized control, network programmability, and dynamic flow management. By decoupling the control and data planes, SDN enables global network visibility and flexible policy enforcement capabilities that align closely with the requirements of large-scale IoT deployments [8].

This chapter provides a comprehensive overview of the convergence between SDN and IoT, forming what is known as the Software-Defined Internet of Things (SD-IoT). It begins with the foundational principles and architecture of SDN, followed by an examination of IoT systems and their communication models. The chapter then explores the evolution toward SD-IoT, its architecture, distinctive features, and real-world applications. Finally, it identifies key research challenges and open problems that motivate the work presented in the subsequent chapters.

1.2 Fundamentals of Software-Defined Networking

Traditional computer networks, often referred to as legacy networks, are built upon tightly coupled architectures where the control plane (responsible for decision-making) and the data plane (responsible for forwarding packets) coexist within the same network devices such as routers and switches. This coupling leads to distributed control, making network management complex, particularly in large-scale or highly dynamic environments [9].

On the other hand, as networks have grown in scale and complexity especially with the emergence of cloud computing, big data, and IoT, traditional architectures have revealed several limitations [10]. Notably, these networks suffer from:

- ***Limited scalability.*** due to static configurations.
- ***Complex and error-prone management.*** caused by manual device-level configuration.
- ***Vendor lock-in.*** due to proprietary control protocols.
- ***Limited adaptability.*** to real-time network conditions or application-specific requirements.

These challenges necessitated the exploration of complementary technologies capable of abstracting control logic and improving programmability. Among these, software-defined networking emerged as a revolutionary paradigm in the early 2010's [11]. The concept of SDN was originally developed at Stanford University through the Clean Slate Program [12], where the OpenFlow protocol was introduced to decouple the control plane from the data plane.

Multiple definitions have since been proposed to describe SDN. According to the Open Networking Foundation (ONF) [13], SDN is the physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices.

In another widely cited definition, Kreutz et al. [14] describe SDN as, a new approach to networking that allows network administrators to programmatically initialize, control, change, and

manage network behavior dynamically via open interfaces.

The core idea of SDN is to centralize the control logic into a software-based controller, providing a global view of the network. This controller communicates with forwarding devices via standard interfaces like OpenFlow, enabling dynamic and automated configuration of the network based on real-time demands and policies [15].

Table 1.1 compares Traditional Wide Area Network (WAN) and Software-Defined WAN in terms of cost, management, flexibility, automation, connectivity, performance, and security [16, 17]. SD-WAN reduces costs, automates management, and improves flexibility and cloud application performance. It also strengthens security and simplifies WAN scaling compared to traditional networks.

1.3 Principles and architectural layers of SDN

The architectural distinction between traditional and SDN network lies primarily in the coupling of control and data planes. In legacy architectures, network intelligence is distributed across individual devices, such as routers and switches, which independently make forwarding decisions. In contrast, SDN separates these concerns, allowing a centralized controller to manage all forwarding devices across the network [18]. This shift introduces flexibility, simplifies management, and allows dynamic policy enforcement, essential for modern and large-scale network environments [19].

1.3.1 Principles of SDN

SDN is founded on three fundamental principles:

- ***Separating of control and data planes.*** In SDN, the control plane is separated from the data plane. This means forwarding devices, e.g., switches and routers, are relieved of control responsibilities and simply execute rules sent by a centralized controller [9].

- ***Centralized control.*** The logically centralized controller has a complete view of the network, enabling intelligent decision-making and efficient traffic engineering. It acts as the network brain, managing all devices through programmable interfaces [20].

- ***Network programmability.*** SDN introduces a layer of abstraction and automation. Through well-defined APIs (Application Programming Interfaces), operators can dynamically configure, manage, and optimize network behavior without manual intervention, enabling innovations such as application-aware routing, QoS enforcement, and security policy enforcement [21].

Table 1.1: Comparison of Traditional WAN and Software-Defined WAN (SD-WAN)

Feature	**Traditional WAN**	**Software-Defined WAN**
Cost	High cost for deployment and maintenance	Consolidated services significantly reduce total cost of ownership
Management approach	Conventional method for managing Wide Area Networks	Software-defined approach for centralized and automated WAN management
Flexibility	Limited flexibility in WAN management	Provides high flexibility for dynamic WAN management
Configuration time	New configurations and scaling require significant time and manual intervention	New configurations and scaling are fast and automated
Automation	Requires skilled personnel for configuration	Network configuration is automated, minimizing human intervention
Connectivity	High cost, lower-speed connectivity	Provides low-cost, high-speed connectivity
Application performance	Low performance for cloud applications due to indirect access via hubs	High performance by enabling direct cloud application access
Data center limitations	Data center scalability is limited by physical hardware constraints	Data centers are not constrained by the underlying hardware
Complexity	High complexity in managing, configuring, and scaling WANs	Simplifies WAN management, configuration, and infrastructure deployment
Security features	Provides VPN security but lacks integrated features like firewalls or WAN optimization	Delivers secure VPN with integrated features such as firewalls, WAN optimization, and secure web gateways
Data security	Secure over private MPLS connections with minimized packet loss	Ensures secure data traffic with end-to-end encryption over VPN and additional security features

1.3.2 SDN layered architecture

Building on the previous principles, the SDN architecture is generally structured into three functional layers:

1. ***Application layer.*** This top layer includes business and network applications that define policies, requirements, and services. Examples include firewalls, load balancers, routing protocols, and QoS management tools. These applications communicate their intent to the controller using

Northbound APIs [22].

2. ***Control layer.*** At the core of SDN lies the SDN Controller or Network Operating System, which interprets application policies and converts them into actionable rules for the infrastructure layer. It performs tasks such as topology discovery, route calculation, network monitoring, and security enforcement [14]. Popular controllers include POX, ONOS, Ryu, and Floodlight.

3. ***Infrastructure (Data) layer.*** This layer consists of physical or virtual switches responsible for forwarding data packets based on instructions received from the control plane. These switches do not make autonomous decisions and operate solely as simple packet forwarders [23].

To enable seamless interaction among these layers, interfaces play a critical role. The Southbound Interface (SBI) acts as a bridge that manages the exchange of information between the controller and network devices, with OpenFlow being the most widely adopted protocol [24]. It allows the controller to install flow rules on forwarding devices, monitor packet statistics, and dynamically adjust traffic flows. On the other hand, the Northbound Interface (NBI) connects the controller with network applications. Typically implemented as RESTful APIs, NBIs enable applications to query network status or submit policy rules. Although less standardized than SBIs, NBIs provide the flexibility for application-driven control of the network [25].

One of SDN's most powerful features is its ability to present a logical abstraction of the entire network to applications. Instead of dealing with individual devices and low-level configurations, applications interact with a virtualized and simplified network model, which enables a rapid service deployment, dynamic resource allocation, and simplified troubleshooting and analytics.

This abstraction not only simplifies network management but also enables the realization of advanced capabilities such as intent-based networking, dynamic policy enforcement, and support for multi-tenant environments. By decoupling control from forwarding and introducing programmable interfaces, SDN significantly transforms how networks are designed, operated, and optimized. Figure 1.1 illustrates the architectural contrast between traditional networks and SDN, highlighting the centralized control and logical separation of layers that define the SDN paradigm.

1.4 Internet of Things networks

In recent decades, the accelerated convergence of embedded systems, wireless communications, and data analytics has given rise to one of the most transformative paradigms in modern computing: the

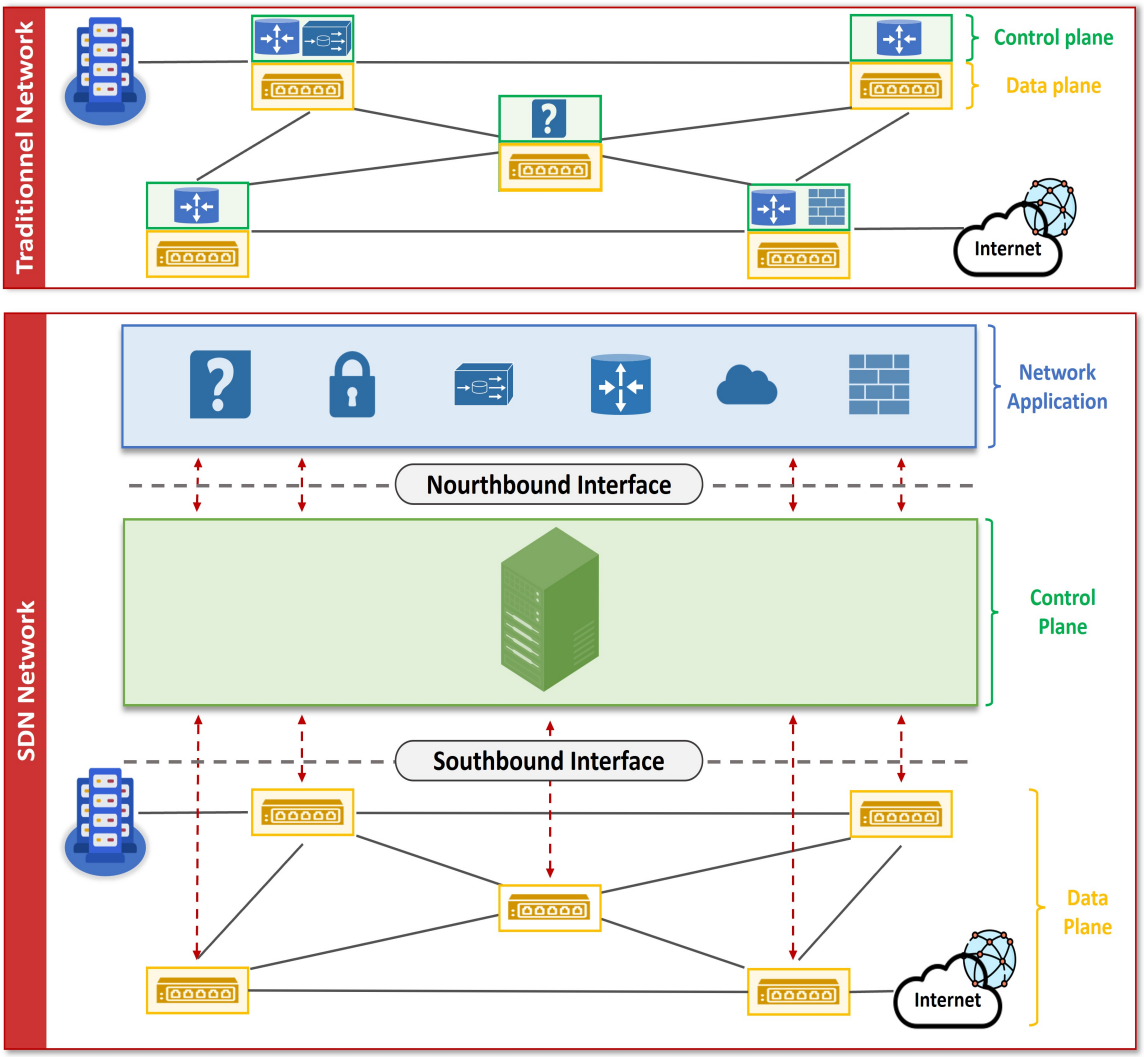


Figure 1.1: Traditional vs. SDN Architecture: illustrating the separation of control and data planes in SDN.

Internet of Things. Far from being a mere technological trend, IoT represents a profound shift in how we perceive and interact with our environment. By enabling everyday objects to sense, process, and exchange data through the internet, IoT redefines the boundaries between the physical and digital worlds [26].

Although the conceptual roots of connecting devices to networks can be traced back to earlier research in ubiquitous and pervasive computing, the term "Internet of Things" was formally introduced in 1999 by Kevin Ashton [27]. He envisioned a system in which physical objects equipped with sensors and identifiers could seamlessly communicate via the Internet offering a new dimension to automation, tracking, and data-driven decision-making [26, 27].

Since then, IoT has evolved into a key driver of digital transformation across multiple sectors, including healthcare, transportation, agriculture, manufacturing, and smart cities. According to recent

forecasts, an enormous number of devices are expected to be connected in the coming years, generating unprecedented volumes of data and requiring scalable and intelligent management strategies [3]

Despite its widespread use, the concept of IoT often suffers from ambiguity due to its interdisciplinary nature and broad range of applications. This lack of a universally accepted definition has led to varied interpretations across academia, industry, and policy-making, sometimes hindering cohesive development and standardization efforts. Nevertheless, several well-recognized definitions have been proposed in the literature, each highlighting key facets of IoT.

According to Oracle, the Internet of Things encompasses a vast array of connected devices. The Internet of Things describes the network of physical objects (things) that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the internet. These devices range from ordinary household objects to sophisticated industrial tools [28].

Adding to this, International Business Machines (IBM) defines IoT as a system of interconnected physical devices, vehicles, appliances, and other objects embedded with sensors, software, and network connectivity. This framework allows them to gather and share data efficiently, demonstrating the transformative impact of IoT across various sectors [29].

While the wording differs, these definitions converge on a common principle: the seamless integration of the physical world with the digital domain via Internet-enabled objects. This interconnectivity allows for the automated collection, exchange, and analysis of data, ultimately enabling smarter environments and more responsive systems without direct human intervention.

1.4.1 Fundamental concepts and architecture

A typical IoT architecture is commonly structured into three fundamental layers: the perception layer, the network layer, and the application layer, as illustrated in Figure 1.2. Each of these layers plays a distinct yet interdependent role in ensuring the efficient flow and processing of information from data acquisition to service delivery in IoT applications [30–33].

1. ***Perception Layer.*** The perception layer serves as the foundational component of the IoT ecosystem. It is responsible for direct interaction with the physical environment through a variety of devices such as sensors, actuators, and RFID (Radio Frequency Identification) readers. These devices detect physical parameters (e.g., temperature, motion, humidity) and convert them into digital signals for further processing. As the initial point of data collection, this layer is critical

to the overall value chain of IoT, providing the raw data upon which all subsequent decisions and actions are based.

2. ***Network Layer.*** Once data is acquired, the network layer handles its reliable transmission to data processing entities. This layer incorporates a wide range of communication technologies, from low-power wireless protocols like LoRa, Zigbee, and NB-IoT, to more conventional networks such as Wi-Fi, Bluetooth, and 5G cellular networks. In addition to data transmission, the network layer may also perform preliminary data filtering or aggregation to reduce the load on downstream systems. It acts as a bridge between the edge devices and higher-level processing units, which may reside in cloud platforms, edge gateways, or centralized servers.

3. ***Application Layer.*** The application layer represents the final stage of the IoT architecture, where processed data is transformed into actionable insights and user-oriented services. It includes a diverse range of IoT applications across sectors such as healthcare, smart cities, agriculture, and industrial automation. This layer not only interprets and visualizes data but also enables interaction with users through intuitive interfaces. The services provided are typically tailored to the specific requirements of end users, delivering added value through automation, monitoring, and decision support.

In the literature, several alternative IoT architectures have been proposed to meet evolving system requirements. Some studies extend the traditional model to a four-layer architecture by incorporating a data processing or service layer [34]. Others propose a five-layer architecture by introducing middleware and business layers to enhance interoperability, scalability, and service management [10, 35, 36]. Furthermore, to address more complex IoT challenges such as security, privacy, and quality of service (QoS), six- and seven-layer architectures have also been suggested [37, 38].

1.4.2 Key characteristics of IoT environments

The IoT constitutes a major shift in networked system paradigms, where ubiquitous connectivity merges the physical and digital worlds. Unlike traditional computing environments, IoT ecosystems are uniquely defined by their ability to integrate diverse, constrained devices into cohesive and intelligent networks. These environments exhibit four fundamental traits that shape their design and operational challenges. Specifically, IoT systems are characterized by:

- ***Massive scale.*** By 2030, billions of devices are anticipated to be interconnected. [39].

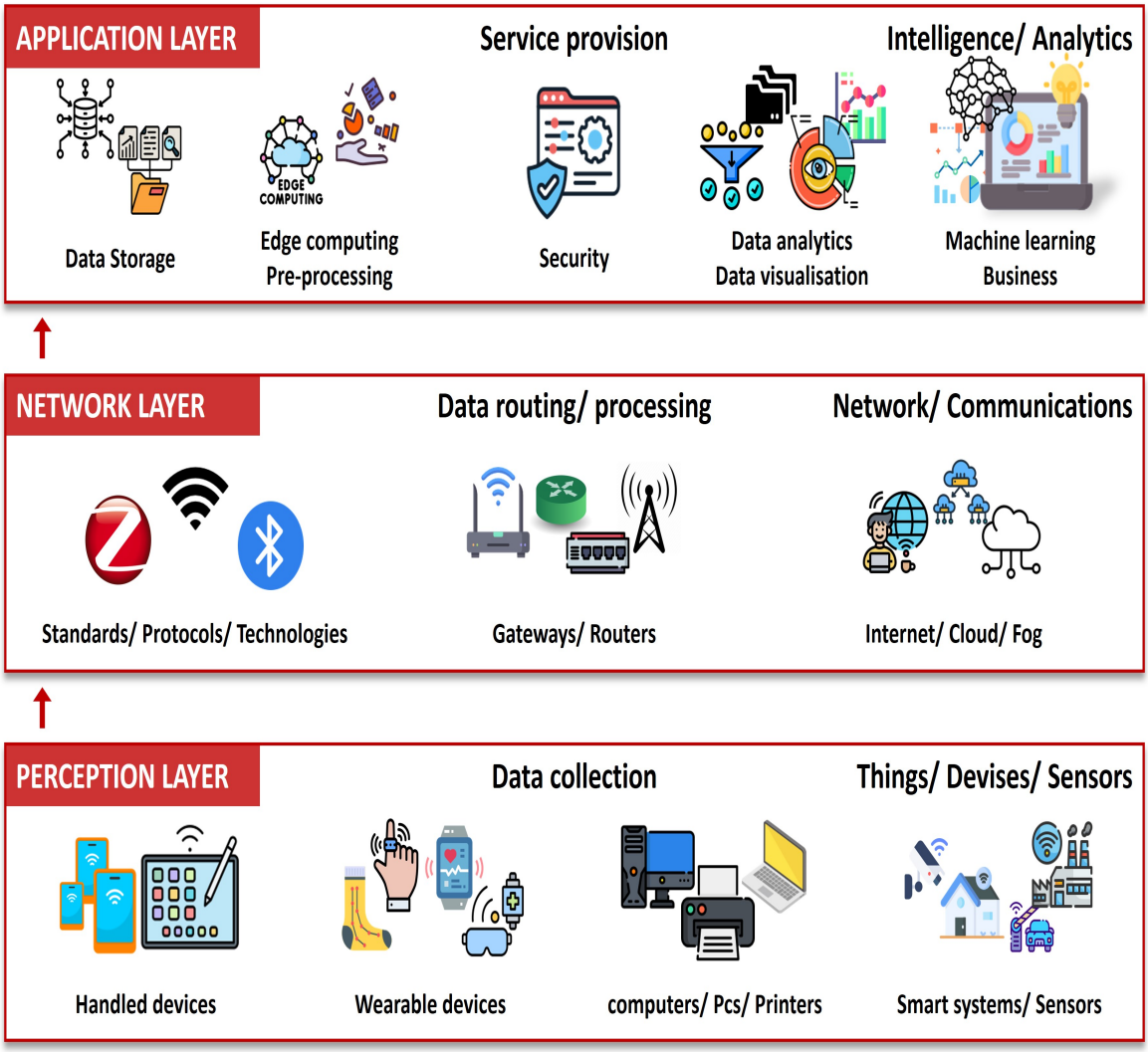


Figure 1.2: Three-layered IoT architecture.

- ***Heterogeneity.*** Devices vary in capabilities, protocols, and energy profiles [40].
- ***Resource constraints.*** Many IoT nodes are battery-powered with low CPU and memory [41].
- ***Real-time responsiveness.*** Applications such as autonomous vehicles and health monitoring require minimal latency [42].

These distinctive characteristics pose significant challenges to IoT network management, rendering traditional or static networking approaches insufficient. Dynamic and adaptive solutions are required to address scalability, heterogeneity, and stringent latency demands.

1.4.3 The need for advanced management technologies

The complexity and dynamic nature of IoT demand more intelligent, programmable, and scalable networking solutions. Traditional IP-based networks struggle with device mobility, interoperability,

and efficient resource allocation. This has led to the exploration of emerging paradigms such as SDN, which enables flexible and centralized control over heterogeneous IoT infrastructures. While SDN is not yet a universal solution, its architectural principles offer a promising foundation for future IoT network management [14].

1.5 From SDN and SD-WAN to SD-IoT: A paradigm shift

The evolution of networking paradigms has been marked by the transition from traditional architectures to more dynamic and programmable frameworks. SDN emerged as a response to the limitations of conventional networks, introducing a decoupled control and data plane architecture that allows for centralized management and enhanced flexibility [43]. This shift has been instrumental in addressing the growing demands of modern applications and services, particularly in the context of larger networks.

Building upon the principles of SDN, Software-Defined Wide Area Networking (SD-WAN) was developed to optimize the performance of wide-area networks, especially for enterprises with geographically dispersed branches. SD-WAN leverages SDN concepts to intelligently route traffic across multiple connections, ensuring improved application performance and reduced operational costs [44]. While SD-WAN provides centralized management and efficient control in traditional WANs, it faces distinct challenges in IoT networks, where the dynamic nature and large number of connected devices strain scalability and adaptability in ways not seen in conventional WANs.

The proliferation of IoT devices has introduced a new set of requirements for network infrastructures, including the need for real-time data processing, low latency, and robust security mechanisms. Traditional networking approaches often prove inadequate for such demands as a result of their static configurations and limited scalability. In this context, the integration of SDN principles into IoT networks has led to the emergence of Software-Defined IoT (SD-IoT), a paradigm that combines the programmability of SDN with the diverse and dynamic nature of IoT systems [45].

SD-IoT architectures facilitate centralized control over heterogeneous IoT devices, enabling dynamic resource allocation, efficient traffic management, and enhanced security through programmable policies [7]. By abstracting the underlying network infrastructure, SD-IoT allows for greater flexibility in deploying and managing IoT services, catering to the specific requirements of various applications. Moreover, integrating emerging technologies further enhances the capabilities of SD-IoT, enabling predictive analytics and autonomous decision-making within the network [46].

The integration of SDN and SD-WAN paradigms into IoT networks has led to the SD-IoT paradigm, representing a significant shift in network design and management, addressing the limitations of previous architectures and paving the way for more resilient, scalable, and intelligent networks. As the landscape of connected devices continues to expand, the adoption of SD-IoT frameworks will be crucial in meeting the evolving demands of modern digital ecosystems.

Table 1.2 presents a comparative overview of the main features distinguishing SDN, SD-WAN, and SD-IoT. In the following sections, we explore the architecture, components, and unique characteristics that define SD-IoT systems and highlight their role in enabling efficient, secure, and scalable IoT deployments.

Table 1.2: A comparative Overview of SDN, SD-WAN, and SD-IoT

Feature	**SDN**	**SD-WAN**	**SD-IoT**
Primary Use Case	Data centers, campus networks	Enterprise WAN connectivity	IoT environments (sensors, actuators, gateways)
Control Level	Centralized (network-level)	Centralized (branch/site-level)	Centralized & distributed (edge-level)
Device Awareness	Low	Medium	High (application & context-aware)
Protocol Support	IP-based	IP/MPLS/Broadband	Heterogeneous (LoRa, Zigbee, BLE, etc.)
Mobility and Scalability	Limited	Moderate	High
Security Model	Flow-based policies	Traffic encryption and segmentation	Lightweight, programmable, dynamic security

1.6 SD-IoT architecture and key components

The SD-IoT represents a convergence of network programmability and IoT scalability, built upon the core principles of SDN. Its architecture is designed to overcome traditional IoT limitations such as static routing, vendor lock-in, and limited scalability by introducing a centralized and programmable control model tailored to heterogeneous and resource constrained devices [45, 47].

1.6.1 Layered architecture and service delivery process of SD-IoT

The SD-IoT architecture typically adopts a layered and modular design, aligning with the principles of SDN while addressing the unique characteristics of IoT. As illustrated in Figure 1.3, it is commonly

divided into several layers [47, 48].

1. ***Perception/Device layer.*** This layer comprises physical IoT devices sensors, actuators, RFID tags, and embedded systems that collect data from the environment. These devices are often constrained in terms of power, memory, and processing, making lightweight and low-overhead communication essential.

2. ***Network layer.*** The role of this layer is to handle data transmission across the infrastructure. It integrates wireless and wired technologies, routing data toward processing centers. SDN switches or OpenFlow enabled devices may be deployed to support dynamic and programmable routing decisions.

3. ***Control layer.*** This is the intelligence core of the architecture, composed of SDN controllers adapted to IoT constraints. These controllers maintain a global view of the network, manage flow tables in switches, enforce security policies, and orchestrate resource allocation based on real-time information.

4. ***Application layer.*** This top layer hosts IoT applications and services such as smart city control, industrial automation, or health monitoring systems. Through northbound APIs, applications communicate intent and high-level policies to the control layer, allowing SD-IoT to dynamically adapt the network configuration to application-specific needs.

The service delivery process in a SD-IoT environment follows a multi-phase workflow designed to ensure efficient data handling, network control, and service provisioning [49]. It begins with the generation of large volumes of data by IoT devices such as sensors, actuators, and smart objects operating at the network edge. This raw data is then collected by gateways, which act as intermediaries between devices and the core network, performing task assignment and, when necessary, local processing for real-time applications. The programmable data layer, composed of switches controlled by the SDN controller, enables dynamic routing, flexible traffic management, and policy enforcement. At the control layer, SDN controllers function as the network's intelligence, managing traffic allocation, coordinating resources, enforcing security, and adapting to changing conditions based on their global view of the network. Finally, the processed and organized data is offloaded to cloud or fog computing platforms: cloud servers offer scalable and powerful resources for intensive data processing, while fog servers provide localized, low-latency computation to support time-sensitive services.

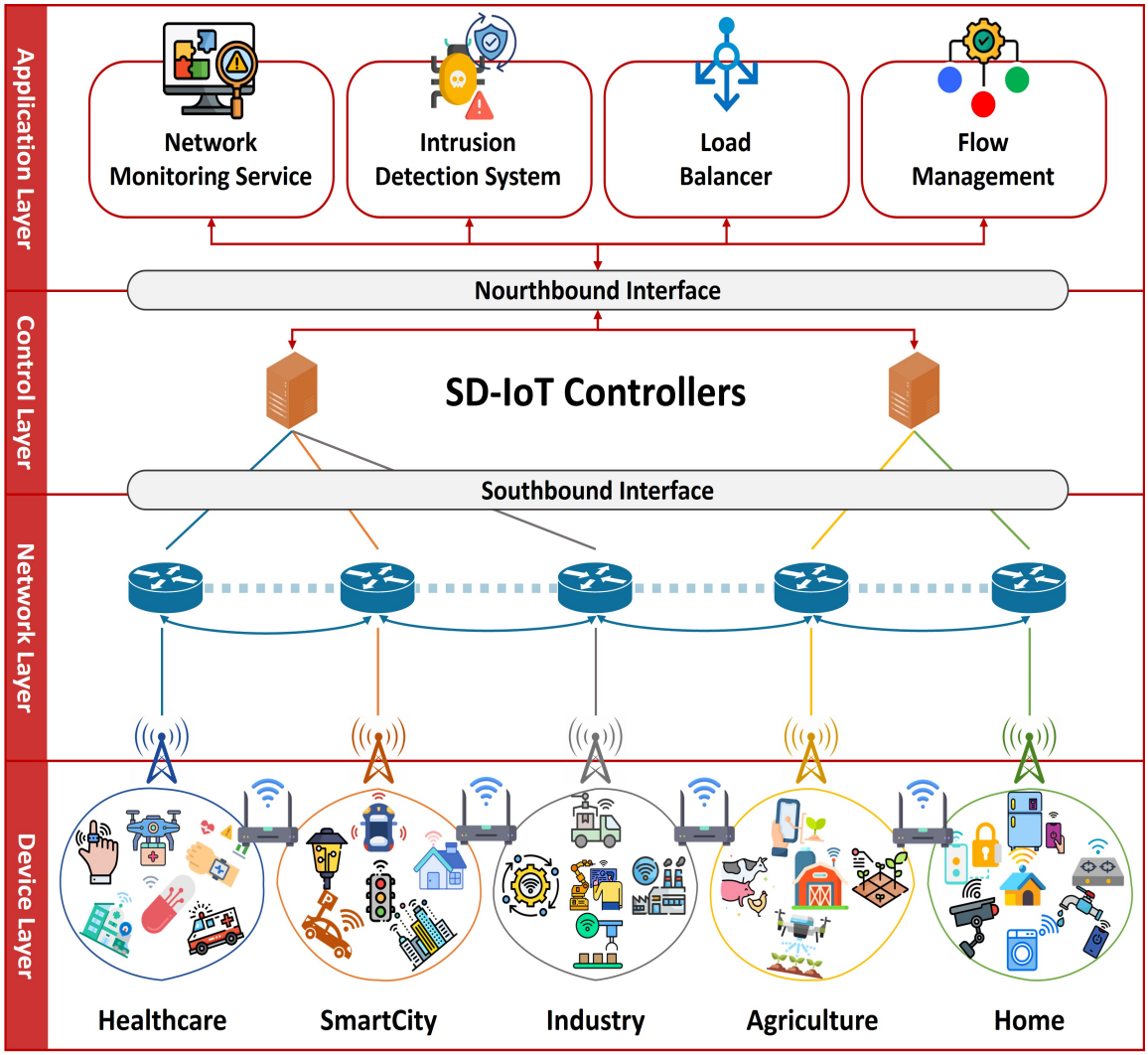


Figure 1.3: SD-IoT Architecture.

1.6.2 Control models in SD-IoT architecture design

The design of SD-IoT architectures incorporates different control models to address the diverse requirements of IoT networks in terms of scalability, latency, and fault tolerance. These networks can adopt centralized, flat distributed, or hierarchical control models, each with its own strengths and limitations. The choice of model depends on the specific requirements of a given deployment scenario [50].

A centralized control model, illustrated in Figure. 1.4 (a), relies on a single controller to manage the entire network, providing unified policy enforcement and simplified orchestration. However, it introduces a single point of failure and may suffer from high latency in large-scale or geographically dispersed IoT deployments, making it less ideal for dynamic, real-time applications. In contrast, as shown in Figure. 1.4 (b), a flat distributed control model employs multiple independent controllers

operating in a peer-to-peer or federated manner, eliminating the reliance on a central entity. This enhances fault tolerance and reduces decision-making latency by allowing localized control. However, the lack of a global view can lead to inconsistencies in policy enforcement, and coordination overhead may increase as the network scales.

The hierarchical control model, depicted in Figure. 1.4 (c), strikes a balance by structuring control across multiple layers. A global controller manages high-level policies while local edge controllers handle real-time, latency-sensitive decisions. This model is particularly effective in large-scale IoT environments such as smart cities and industrial IoT, where both centralized oversight and distributed processing are crucial. Although hierarchical architectures improve scalability and fault tolerance, they introduce complexity in maintaining synchronization between control layers [49].

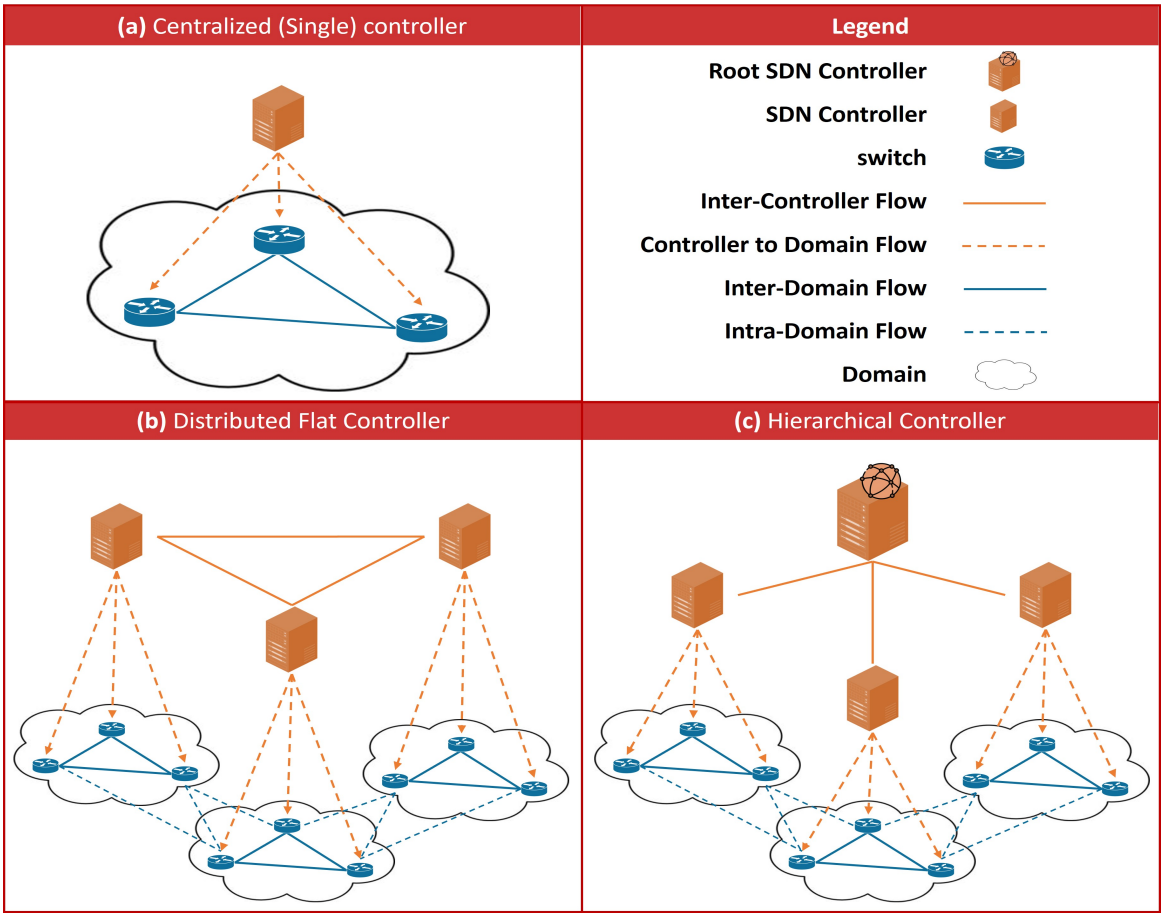


Figure 1.4: Type SDN control model.

1.7 Distinctive characteristics of SD-IoT

The Software-Defined Internet of Things introduces several distinctive characteristics that set it apart from traditional IoT systems. These characteristics stem from its SDN-inspired design principles, such as programmability, logical centralization, and dynamic policy enforcement, which are intended to meet the demands of large-scale, heterogeneous, and resource-constrained IoT environments. However, while these features are common in SDN-based systems, they become increasingly challenging to maintain in SD-IoT networks due to the highly dynamic, large-scale, and heterogeneous nature of IoT environments [8, 51–53].

- ***Programmability.*** One of the core features of SD-IoT is its high degree of programmability, allowing network operators and applications to define, update, and enforce custom network policies dynamically. This eliminates the need for manual device-level configuration and supports agile responses to changing network conditions.

- ***Centralized and Global Network View.*** Unlike traditional IoT networks, which rely on localized and often fragmented control, SD-IoT enables a logically centralized control plane. This centralized controller maintains a global view of the entire IoT infrastructure, enabling optimized decisions related to routing, load balancing, QoS management, and security policy enforcement.

- ***Scalability and Flexibility.*** SD-IoT is inherently scalable due to its layered and modular architecture, allowing the integration of new devices and services without reconfiguring the entire network. It supports horizontal and vertical scaling through distributed controllers or edge agents, as well as dynamic resource provisioning across edge and cloud nodes.

- ***Heterogeneity Support.*** IoT environments often consist of highly diverse devices with varying capabilities, communication protocols, and data formats. SD-IoT supports heterogeneous interoperability by abstracting device-specific complexities at the control plane level. Gateways and translation mechanisms (e.g., protocol converters) further enhance this support.

- ***Enhanced Security and Policy Enforcement.*** SD-IoT allows fine-grained and flow-level policy enforcement using the global visibility of the network. Security services such as access control, anomaly detection, and isolation of malicious nodes can be programmed directly into the controller logic. This enables lightweight and dynamic security mechanisms, crucial for protecting resource-constrained IoT devices.

- ***Real-Time Network Adaptation.*** Due to its control architecture, SD-IoT can respond in real time to topology changes, device failures, and varying QoS demands. This adaptability is critical in domains like smart healthcare or industrial IoT, where timely data processing and response are essential.

- ***Edge and Cloud Integration.*** SD-IoT architectures often support multi-layer orchestration between cloud and edge layers. This allows data to be processed close to the source for latency-sensitive applications, while still benefiting from cloud-based intelligence and long-term storage. Edge controllers or micro-SDN agents are commonly deployed in smart city and industrial settings.

1.8 Real world application scenarios of SD-IoT

SD-IoT is a paradigm that introduces dynamic control, centralized intelligence, and service-aware orchestration into IoT ecosystems. These capabilities enable SD-IoT to overcome many limitations of traditional IoT networks, such as scalability, heterogeneity, and the lack of end-to-end QoS enforcement. In smart cities, which rely on a dense and heterogeneous network of sensors, actuators, and communication nodes for services like traffic management, smart lighting, pollution monitoring, and waste management, SD-IoT provides centralized programmability and control, allowing dynamic reconfiguration of traffic lights based on real-time congestion or rerouting of vehicles during emergencies, while simplifying the management of thousands of heterogeneous devices [54]. In healthcare, especially post-COVID, wearable devices and home-based sensors enable real-time patient monitoring, elderly care, and emergency response. SD-IoT supports intelligent routing, secure communication, and policy enforcement for sensitive health data, prioritizing critical flows such as heartbeat anomalies over routine traffic using QoS policies [55]. Industrial IoT and smart manufacturing environments require low latency, real-time control, and resilient communication between sensors, robotic arms, and control systems. Here, SD-IoT provides centralized coordination, dynamic flow control, and network slicing tailored to time-critical industrial applications [56]. In intelligent transportation systems, modern vehicles and infrastructure integrate GPS, cameras, radar sensors, and Vehicle-to-Everything communication modules. SD-IoT facilitates adaptive routing, vehicular edge computing, and traffic prediction models through centralized policy enforcement and rapid response to environmental changes [57]. Similarly, in smart agriculture, precision farming leverages soil sensors, drones, irrigation actuators, and weather forecasting systems. SD-IoT enables centralized and

intelligent data aggregation, automating decisions such as when and how much to irrigate or fertilize [58]. Finally, during natural disasters, when communication networks are often disrupted, SD-IoT can deploy temporary, programmable emergency networks using mobile nodes like UAVs and ground sensors, managed by a centralized controller that prioritizes emergency communication and resource deployment. These diverse real-world applications of SD-IoT are illustrated in Figure. 1.5.

Figure 1.5: Application SD-IoT.

1.9 Open challenges and research directions in SD-IoT

Several key challenges must be addressed to fully realize the potential of SD-IoT. The following discussion highlights the main open issues and research directions.

- ***Controller Placement and Management.*** A central element of SD-IoT is the controller, responsible for network-wide logic and decision-making. The placement of controllers in distributed IoT environments is challenging due to trade-offs among latency minimization, load balancing, fault tolerance, and mobility support [59].

- ***Scalability and Dynamic Topologies.*** SD-IoT, built on IoT networks that may scale to billions of devices with dynamic states and intermittent connectivity, must address several key challenges,

including massive numbers of flow requests, varying traffic patterns, and dynamic device mobility [8]. Ensuring controller scalability, network slice isolation, and distributed flow management remains challenging, with hierarchical control, edge delegation, and multi-controller federation being active research solutions.

- ***Security and Privacy.*** The centralized nature of SD-IoT and the sensitivity of IoT data make the architecture vulnerable to denial-of-service attacks, man-in-the-middle intrusions, unauthorized policy access, and privacy violations. Lightweight encryption, blockchain-based authentication, and policy-aware secure routing have been proposed, though their deployment remains limited due to resource constraints on IoT devices [60].

- ***Energy Efficiency and AI-Edge Integration.*** Most IoT devices are battery-powered and resource-constrained, making energy efficiency a critical concern. SD-IoT introduces overhead through control polling, rule updates, and security handshakes. Energy-aware flow scheduling, duty-cycled control interaction, and offloading to edge nodes or microcontrollers are being explored to reduce consumption while maintaining responsiveness [60]. In parallel, AI-driven SD-IoT and edge-assisted control are gaining traction to enable decentralized decision-making, adaptive optimization, anomaly detection, and predictive maintenance. However, challenges remain in terms of training data quality, model interpretability, and preventing controller overload [51].

- ***Standardization, Interoperability, and Big Data Management.*** The lack of standardized southbound protocols for constrained IoT devices limits interoperability. While OpenFlow suits data centers, lightweight IoT often relies on CoAP, MQTT, or NETCONF, which require further integration for unified controller semantics [61]. Moreover, heterogeneous devices and variable data rates complicate network abstraction and policy enforcement. The resulting data volume challenges flow tables, bandwidth, and edge/cloud analytics [62], motivating AI-driven flow classification, adaptive buffering, and real-time stream processing.

- ***Real-Time Decision-Making and QoS.*** Applications such as smart grids, autonomous vehicles, and emergency systems demand strict QoS guarantees. SD-IoT must support latency-aware, priority-based, and context-driven flow control across multi-hop and wireless links [63]. Real-time monitoring, AI-driven traffic classification, and context-aware controller logic are emerging solutions to achieve dependable QoS.

1.10 Conclusion

This chapter presented the foundations of SD-IoT, tracing the evolution from traditional networks to SDN and highlighting the benefits of separating the control and data planes along with centralized programmability. We discussed the structural and functional characteristics of IoT networks, emphasizing their large scale, heterogeneity, and communication constraints. By applying SDN principles to IoT, SD-IoT provides a flexible and layered framework capable of efficiently managing diverse and large-scale IoT infrastructures.

Despite these advantages, SD-IoT still faces key challenges, including optimal controller placement, efficient big data flow management, energy optimization, and secure, real-time operation in highly dynamic environments. Addressing these issues motivates ongoing research and underscores the need for advanced SD-IoT models and solutions, which will be examined in the following chapters.

2

Controller Placement Problem

2.1 Introduction

The Controller Placement Problem (CPP) is a fundamental issue in Software-Defined Networking, as it significantly influences control-plane latency, network scalability, and fault tolerance. Since its emergence in 2012, numerous research efforts have proposed a variety of approaches to optimize controller deployment based on different objectives and constraints. This chapter presents a structured overview of the CPP, analyzing existing surveys, categorizing placement methods, and highlighting the network types and performance metrics considered. It also provides a comparative analysis to identify patterns, innovations, and limitations in the current literature. By synthesizing these studies, we aim to outline the evolution and current state of CPP research. This sets the stage for the upcoming chapter, which introduces the system model and formal problem formulation adopted in our work.

2.2 Controller Placement Problem

In SDN and SD-IoT, identifying the optimal number and locations of controllers within a network represents a fundamental challenge aimed at meeting specific performance objectives. It is typically defined over a network modeled as a graph G, with a subset V representing the switches or IoT gateways and a set of links connecting them. The CPP consists of selecting a subset of nodes from V to host controllers in such a way that a chosen performance metric such as average switch to controller latency, maximum latency, controller to controller delay, or load imbalance is optimized. This problem is particularly challenging in SD-IoT environments due to the large scale, dynamic behavior, and heterogeneity of devices, as well as the need to consider constraints related to resource usage, resilience, and scalability.

In the literature, CPP is commonly studied in two main variants. The k-controller CPP assumes that the number of controllers is fixed (e.g., due to cost or energy constraints), and seeks the best placement to optimize a performance criterion. In contrast, the β-metric CPP focuses on minimizing the number of controllers required to ensure that a given performance threshold is not exceeded for instance, ensuring that all switches are connected to a controller with a latency not exceeding a defined bound. These two formulations address complementary needs: the former is relevant when deployment resources are limited, while the latter is suitable when stringent performance requirements must be enforced. The CPP remains an active area of research due to its combinatorial nature and critical impact on the efficiency and responsiveness of SDN-based systems.

2.3 Summary and comparison of CPP studies in SDN

Over the past decade, the CPP in SDN has attracted considerable research attention due to its direct impact on network performance. In response, numerous survey papers have been published to synthesize and classify the evolving landscape of CPP solutions. Figure 2.1 illustrates the number of survey papers published per year, revealing the increasing attention the CPP topic has received over time. Furthermore, Figure 2.2 provides a two-line comparative view: the blue line plots the number of citations, calculated from the publication date up to the end of 2025 and used as an indicator of research impact, while the red line shows the number of research papers reviewed in each survey, reflecting the scope of literature coverage.

The diversity of these surveys in terms of scope, depth, network types considered, and proposed future directions makes it essential to conduct a comparative analysis. To support this evaluation,

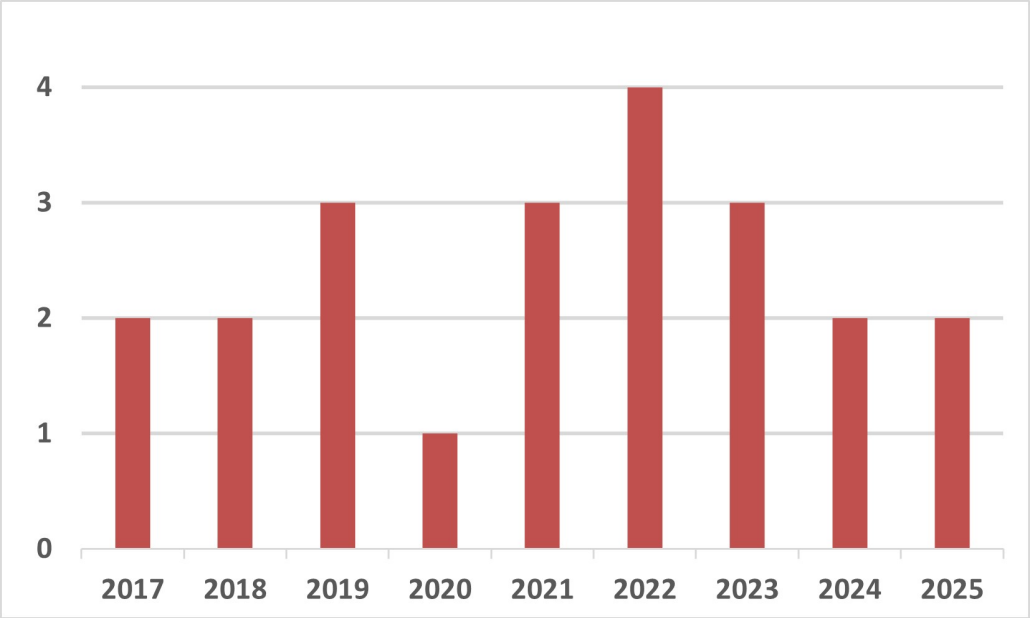


Figure 2.1: Number of survey papers published per year.

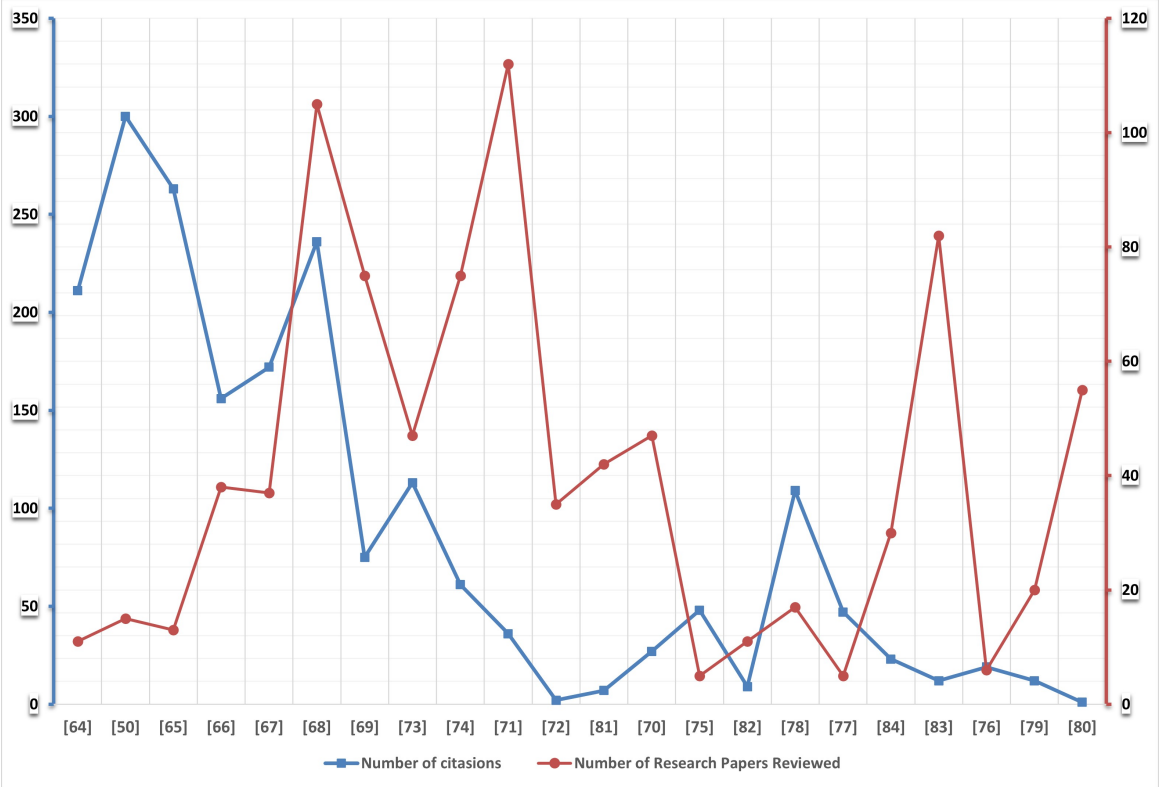


Figure 2.2: Comparison of the number of citations and reviewed papers across CPP survey studies.

Table 2.1 presents a detailed summary of each survey based on five key attributes: publication year, network type, level of CPP coverage, number of reviewed papers, and a brief description of the study. Such a comparison helps identify gaps in the existing literature and highlights emerging trends and research priorities, providing researchers with a meta-level overview of the field. In the following

section, we review and summarize the key points studied in each survey paper, providing insights into their unique contributions and focus areas.

The first comprehensive survey on CPP [64] defines the problem and classifies objectives into three main categories: minimizing network latency, maximizing reliability and resilience, and reducing infrastructure cost and improving energy efficiency. The study introduces the Clustering-Based Network Partition Algorithm (CNPA) to partition networks and place controllers strategically, evaluated using the OS3E network. Similarly, [50] provides a broad review of multiple SDN controllers, highlighting architectures, key challenges such as consistency, placement, and scheduling, and metrics for multi-controller systems. In the same period, [65] traces the evolution from single to multi-controller SDN, focusing on scalability, consistency, reliability, and load balancing, and introduces flat and hierarchical architectures. In addition, [66] categorizes CPP into Uncapacitated (UCPP) and Capacitated (CCPP) versions, further subdivided by traffic awareness, fault tolerance, and network partitioning.

Several subsequent works classify CPP solutions according to optimization goals: latency, reliability, cost, energy efficiency, or multi-objective criteria. For example, [67] categorizes solutions into latency, reliability, cost, and multi-objective goals. Along the same line, [68] classifies CPP research based on objectives, modeling choices, evaluation metrics, and solution methodologies, extending applications to mobile, 5G, named data, wireless mesh, and vehicular networks. Another contribution [69] emphasizes six aspects: target network environment, traffic patterns, controller features, solution approaches, reliability aspects, and optimization goals. More recently, [70] discusses optimization objectives including latency, reliability, controller capacity, load balancing, energy consumption, and cost, while also highlighting architectural considerations in the control and application planes.

The surveyed solutions also differ in terms of methodologies. Some reviews group CPP approaches into exact, heuristic, meta-heuristic, clustering-based, and game-theoretic categories. Exact methods, such as ILP and MILP, provide optimal solutions but are computationally intensive, whereas heuristics reduce complexity at the expense of optimality. Meta-heuristic techniques, including Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), explore the solution space effectively but require parameter tuning [71]. Clustering strategies partition networks into domains to simplify controller allocation, while game-theoretic models analyze switch-controller interactions. Other surveys categorize solutions into static versus adaptive approaches, with adaptive methods further subdivided into wired and wireless scenarios [72]. Optimization strategies also include clustering, ILP, evolutionary algorithms, and machine learning [73], while [74] presents a taxonomy based on objective functions and network traits, highlighting trade-offs among latency, load balancing, and reliability.

In addition to classical SDN, CPP has been studied in emerging network environments. Surveys dedicated to SD-WSNs and SD-IoT emphasize challenges such as limited energy, hardware constraints, and dynamic traffic, requiring tailored placement strategies [75–77]. Artificial intelligence and machine learning methods are being progressively utilized for adaptive, real-time, and energy-efficient controller placement within these environments. Software-defined satellite networks (SD-SatNets) [78] and SD-WANs [79] introduce additional constraints, including inter-satellite links, virtual ground stations, and cross-domain adaptation. These surveys emphasize AI-driven optimization, security-aware strategies, and multi-objective trade-offs to enhance scalability, reliability, and cost efficiency [80].

Across all these works, recurring open issues include scalability to large networks and dynamic conditions [81, 82], balancing latency, reliability, energy efficiency, and deployment cost [67, 70], and the need for adaptive and real-time CPP strategies capable of responding to dynamic network conditions [76, 79, 83]. Security, fault tolerance, and heterogeneity in wireless and emerging networks also remain critical concerns [75, 80]. These works collectively highlight that CPP is NP-hard, with complex trade-offs among multiple objectives. Future research is expected to focus on dynamic, energy-aware, scalable, and intelligent placement strategies suitable for heterogeneous SDN deployments, including IoT, WSNs, SatNets, and SD-WANs.

Table 2.1: Comparative Overview of Survey Papers on Controller Placement Problem (CPP) in SDN

Ref	Year	Network Type	CPP Review	#Papers Covered	Brief description
[64]	2017	Networks in general	Dedicated	11	Surveyed and classified CPP solutions in SDN by objectives like latency, reliability, cost, and energy, highlighting open challenges.
[50]	2017	Networks in general	Sectional	15	Surveyed and classified CPP solutions in SDN by objectives like latency, reliability, cost, and energy, highlighting open challenges.
[65]	2018	Networks in general	Dedicated	13	Reviewed and categorized multi-controller SDN research according to key aspects including scalability, consistency, reliability, and load balancing.
[66]	2018	Networks in general	Dedicated	38	Survey and classify CPP solutions in SDN
[67]	2019	Wired networks: Data-centers and WANs	Dedicated	37	Reviewed SDN CPP by objectives: atency, reliability, cost, and multi-objective highlighting key algorithms and future challenges.
[68]	2019	Data-centers, WANs, mobile networks, 5G, NDN, WMNs, and VANETs.	Dedicated	105	Provided an overview of the CPP in SDN, emphasizing its significance and recent applications across emerging domains.
[69]	2019	Wired and Wireless Networks	Dedicated	75	Analyzed existing CPP literature across six key aspects target network environment, traffic characteristics, controller characteristics, solution approach, network reliability, and optimization objectives such as latency, cost, load, energy, and QoS.

Table 2.1 : continued

Ref	Year	Network Type	CPP Review	#Papers Covered	Brief description
[73]	2020	Networks in general	Dedicated	47	Discussed SDN controller placement challenges, highlighting optimization trade-offs and the necessity for energy-, cost-, and attack-aware CPP strategies in SDN–WSN environments.
[74]	2021	Networks in general	Dedicated	75	Discussed SDN CPP optimization strategies and proposed a taxonomy of solutions categorized by objective functions, metrics, and research challenges.
[71]	2021	Networks in general	Dedicated	112	Explored CPP in SDN through formal definitions, performance metrics, and solution classifications, emphasizing static and dynamic controller placement methods and open research gaps.
[72]	2021	Networks in general	Dedicated	35	Explored SDN CPP challenges and presented the Garter Snake Optimization–based Capacitated Controller Placement Problem to minimize delay and improve network performance.
[81]	2022	Data-centers, WANs	Dedicated	42	Explored CPP in SDN through optimization-based algorithms, highlighting scalability, reliability, and architectural considerations within multi-controller environments.
[70]	2022	Networks in general	Dedicated	47	Reviewed SDN CPP studies, emphasizing optimization objectives, mathematical formulations, and solution strategies impacting network performance.
[75]	2022	WSN	Sectional	5	Reviewed SD-WSN with emphasis on Machine Learning-based solutions for intelligent, adaptive, and resource-aware network management.

Table 2.1 : continued

Ref	Year	Network Type	CPP Review	#Papers Covered	Brief description
[82]	2022	Networks in general	Dedicated	11	Reviewed SDN CPP studies with emphasis on the optimization metrics used.
[78]	2023	SatNet	Sectional	17	Discussed SD-SN architectures, key technologies, existing solutions, challenges, and simulation tools.
[77]	2023	WSN	Sectional	5	Focused on three SDWSN domains: controller-related, network-related, and energy-related issues.
[84]	2023	ISP/Telconetworks	Dedicated	30	Explores the benefits of network partitioning and clustering-based CPP approaches for efficient SDN deployment.
[83]	2024	Networks in general	Dedicated	82	Reviewed and categorized SDN CPP studies into four objectives: latency, resilience, energy/cost, and multi-objective, and introduced a new method to calculate the network search space for optimal controller placement.
[76]	2024	IoT	Dedicated	6	Reviewed and classified SDN-IoT CPP approaches into mono-objective and multi-objective categories.
[79]	2025	WAN	Dedicated	20	Critical review of SD-WAN CPP approaches highlighting strengths, limitations, and future research needs in multi-objective optimization.
[80]	2025	SatNet / IoT / VANET	Dedicated	55	Concise review of CPP in emerging networks, identifying factors that affect placement decisions and the essential requirements for optimized controller allocation.

Controller Placement Problem strategies differ widely based on the characteristics of the SDN environments in which they are applied. To capture this diversity, we organize the existing literature into a five-dimensional classification: network scale, mobility and dynamics, latency and criticality, deployment environment, and evolution stage. Each dimension emphasizes a different aspect that influences controller placement decisions and design priorities.

- **Network scale**. differentiates between large-scale networks, which span broad geographical areas and often involve long-distance communication with higher latency concerns, and small-scale networks, which operate in localized and often constrained environments that prioritize fast responsiveness and energy efficiency.
- **Mobility and dynamics**. addresses the degree of topological stability in the network. Static or fixed networks typically have predictable communication paths, making controller placement more stable. In contrast, mobile or dynamic networks experience frequent changes in topology, requiring adaptive and resilient placement strategies. Hybrid networks may include both mobile and static components, necessitating hybrid controller solutions.
- **Latency and criticality**. distinguishes between networks that demand ultra-low latency, where real-time responses are essential, and those that are tolerant to latency, where moderate delays can be accommodated without significant performance degradation.
- **Deployment environment**. classifies networks based on whether they are deployed in terrestrial zones, relying on traditional ground infrastructure, or in aerial or space-based environments, where high mobility or propagation delays present additional placement challenges.
- **Evolution stage**. separates networks into traditional and backbone deployments, which are well-established and focused on scalability and performance, and emerging vertical networks, which are specialized, dynamic, and often domain-specific, requiring more customized CPP strategies.

To further clarify how these classifications apply in practice, the following provides a brief description of each SDN network type and its relevance to controller placement studies:

- **SD-WAN.** Software-defined Wide Area Networks are used in enterprise and service provider environments to manage multi-site connectivity.
- **Legacy ISP/Telco.** These represent traditional internet service providers and telecommunications networks transitioning to SDN, aiming to improve resilience, integrate network functions virtually, and handle large-scale traffic.

- **Optical networks.** High-speed fiber-optic backbones used in transport networks, addressing issues like delay minimization and energy-efficient operation to support ultra-high throughput.
- **Satellite networks.** Deployed in space for global coverage, often in low or geostationary Earth orbits, with challenges including intermittent connectivity, long delays, and the integration of ground-space communication.
- **IoT (Internet of Things).** Networks of smart sensors and actuators with limited power and computing capacity, focusing on edge or fog-based controller deployment to reduce overhead and latency.
- **IoV (Internet of Vehicles).** Intelligent transportation systems involving connected cars and road infrastructure, dealing with high-speed mobility, dynamic topology, and safety-critical low-latency requirements.
- **IoF (Internet of Flying Things).** Includes UAVs and drones used for surveillance, delivery, or emergency response, considering aerial mobility, energy constraints, and temporary ad-hoc networking.
- **WSN (Wireless Sensor Networks).** Comprised of battery-powered sensors deployed for environmental monitoring or surveillance, emphasizing energy-efficient communication and sleep scheduling.
- **Data centers.** Centralized computing facilities hosting cloud and enterprise applications, focusing on high scalability, load balancing, and fast fault recovery.

This multi-dimensional classification provides a comprehensive perspective on the heterogeneity of SDN environments involved in CPP research. Understanding these distinctions is crucial for developing placement strategies that are adaptable and tailored to specific use cases. Figure 2.3 illustrates this classification, summarizing the five dimensions along with the associated network types and their defining characteristics.

2.4 Overview and classification of controller placement methods

The CPP has been extensively studied in the literature since 2012 and continues to attract significant research attention due to its crucial role in optimizing the performance of SDN. In our study, we review a comprehensive set of works [85–123], selected to cover a wide range of techniques based on

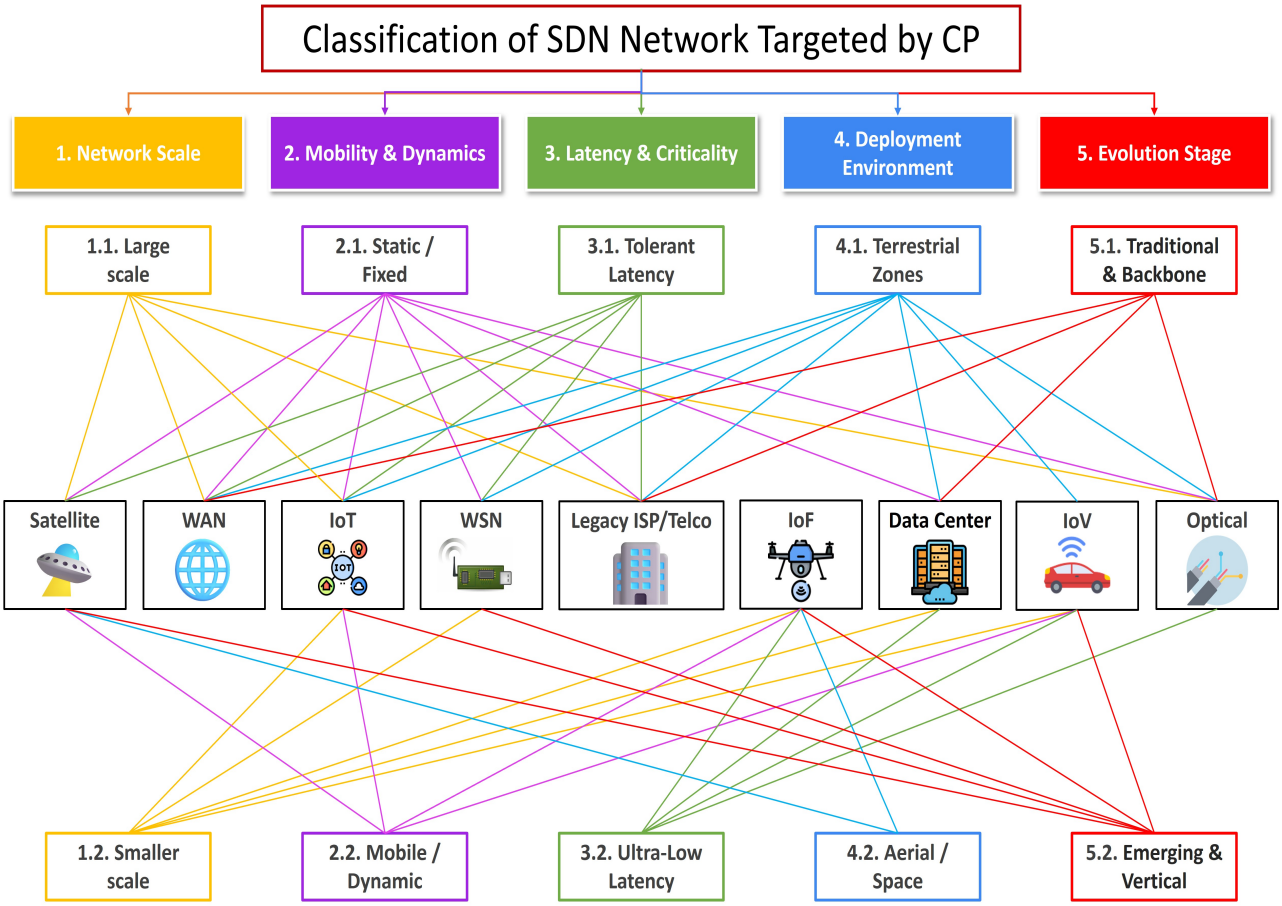


Figure 2.3: Classification of SDN Network Types Addressed in Controller Placement Problem Studies

the strategies employed and the performance metrics addressed. These studies reflect the evolution and diversity of CPP approaches across different network scenarios. Table 2.2 below summarizes these works in terms of publication year, network type, controller location (on-switch or independent), centrality-based approach (yes/no), objectives (LA: network latency, LB: load balancing, SC: system scalability, RE: network reliability, HL: heterogeneous load handling, ET: execution time, FL: average flow rate), methods used, limitations, and key observations. This classification offers a structured and insightful overview of the CPP research landscape.

The CPP was first formalized by Heller et al. [85] as a facility location problem, introducing three foundational metrics: worst-case latency (k-center), average latency (k-median), and switch coverage (set cover). While their work laid the groundwork for CPP research, it overlooked inter-controller latency a critical gap later addressed by subsequent studies. Over the years, CPP solutions have evolved to address diverse challenges, including latency optimization, fault tolerance, scalability, and load balancing, tailored to both SD-WAN and SD-IoT environments.

In SD-WAN networks, numerous approaches have been proposed to optimize controller placement.

Wang et al. [92] introduced a clustering-based algorithm to minimize end-to-end latency, while Chen et al. [96] employed the Louvain heuristic for community detection, achieving balanced controller loads with reduced latency. Kuang et al. [95] improved latency by partitioning the network into single-controller domains using hierarchical K-means clustering, ensuring that controller loads were evenly balanced. Resilience and capacity constraints were addressed by Killi and Rao [87, 90], who developed mixed-integer linear programming models to handle worst-case latency under controller failures. Similarly, Fan et al. [106] optimized latency under normal and failure conditions by balancing primary and backup path delays. Recent works by Dou et al. [115] and Santos et al. [107] incorporated critical programmability and geodiversity, respectively, to enhance placement robustness. Metaheuristic approaches have also been explored, with Sahoo et al. [93] comparing Particle Swarm Optimization (PSO) and Firefly algorithms, demonstrating the latter's superiority in balancing latency and computational efficiency. Kanodia et al. [105] introduced a link failure-aware algorithm to mitigate latency spikes, while Petale and Thangaraj [124] proposed a dynamic strategy for minimizing worst-case latency during multiple link failures. Advanced frameworks, such as those by Mojez et al. [114] and Singh et al. [119], have further improved latency-reliability trade-offs under varying network conditions.

Centrality-based approaches have gained prominence in CPP due to their ability to identify critical network nodes. Zobary et al. [121] leveraged degree centrality to minimize propagation delay, outperforming other methods. However, reliance solely on degree centrality may neglect load distribution, impacting latency under dynamic conditions. Petale and Thangaraj [124] and Qi et al. [100] incorporated betweenness and closeness centrality, respectively, to enhance failure resilience. Calle et al. [108] combined both metrics to optimize primary and backup controller placement, improving resilience against targeted attacks a challenge also addressed by Rueda et al. [91]. Adebayo et al. [117] integrated multiple centrality measures using Dempster-Shafer theory [125] to optimize placement in SD-WANs, though computational complexity remains a limitation. Hirayama et al. [99] demonstrated the correlation between salience and betweenness centrality, particularly in Barabási-Albert networks, while Adebayo et al. [118] proposed neighborhood-based algorithms to reduce computational overhead. Other hybrid approaches, such as Alhazmi et al.'s [97] integration of betweenness centrality with hierarchical clustering and Liao et al.'s [113] multi-objective differential evolution, have further advanced latency and load-balancing optimization.

In SD-IoT networks, CPP solutions must address dynamic and resource-constrained environments. Choumas et al. [126] formulated placement as an Integer Quadratic Programming problem, reducing

control traffic in low-power IoT. Tran et al. [102] employed submodular optimization to maximize utility under budget constraints, while Hans et al. [110] combined Enhanced Sunflower Optimization with Pareto Optimal placement to minimize latency and improve load balancing. Khera and Kurmi [116] applied unsupervised machine learning for latency reduction in wide-area IoT, whereas Keshari et al. [109] proposed a hybrid Differential Evolution-Whale Optimization algorithm for smart cities. Ali and Roh [111] used the Analytical Network Process for clustering, reducing end-to-end delays compared to k-means. Similarly, Abderrahmane et al. [123] combined the Louvain algorithm with betweenness centrality to enhance reliability in IoT networks, though computational intensity remains a challenge.

Recent advancements include Babbar et al.'s [120] Density-Based Splitting Controller Placement, which optimizes educational content distribution in SDNs, and Sahoo et al.'s [94] Firefly-based approach for wide-area SDNs. Liu et al. [89] and Hu et al. [98] addressed load balancing and reliability, respectively, while Yao et al. [86] emphasized capacitated controller placement. Chai et al. [103] minimized control plane delays using heuristic and Kuhn-Munkres algorithms, and Huque et al. [88] introduced Lidy+ for dynamic adaptation. Ali et al. [112] further extended these efforts in SD-IoT, optimizing controller selection via Analytical Network Decision Making. Collectively, these studies highlight the diversity of CPP solutions, spanning latency minimization, resilience, scalability, and dynamic adaptation across SD-WAN and SD-IoT environments.

Table 2.2: Comparative Overview of SDN Controller Placement Strategies

Ref.	Year	Net. Type	Cntrl Loc.	Cen.-Based	Objectives							Methods	Limitations and Observations
					LA	LB	SC	RE	HL	ET	FR		
[85]	2012	WAN	switch	Yes	✓	✗	✗	✗	✗	✗	✗	k-center,k-median	Controller load is not addressed
[86]	2014	SD-IoT	switch	No	✗	✓	✓	✗	✓	✗	✗	Capacitated K-center	Limited adaptability to dynamic traffic and real-time load changes
[87]	2016	WAN	switch	No	✓	✗	✗	✓	✓	✗	✗	Mathematical model	No support for multiple failures or cost, search space is too large
[88]	2017	WAN	independent	No	✓	✗	✓	✗	✗	✓	✓	Lidy+	Open search suits ideal cases only; controller positions are fixed; results rely on prediction accuracy
[89]	2017	WAN	switch	No	✓	✓	✓	✓	✗	✗	✗	CD, Controller Load-aware Placement	Simplified load and reliability metrics may miss network details
[90]	2017	WAN	switch	No	✓	✓	✗	✓	✗	✗	✗	MILP, SA	Limited to small networks with high execution time
[91]	2017	WAN	switch	Yes	✗	✗	✗	✗	✗	✗	✗	Attack-Aware CP	Ignores geographic placement and physical distances
[92]	2017	WAN	switch	No	✓	✗	✗	✗	✗	✗	✗	CNPA	Lack of resilience analysis

Table 2.2 : continued

Ref.	Year	Net. Type	Cntrl Loc.	Cen.-Based	Objectives							Methods	Limitations and observations
					LA	LB	SC	RE	HL	ET	FR		
[93]	2017	WAN	switch	No	✓	✗	✓	✗	✗	✓	✗	PSO and FFA	Analysis excludes load balancing, resilience, and cost
[94]	2018	WAN	switch	No	✓	✗	✓	✓	✗	✗	✗	PSO and FFA	No controller failure, load balancing, cost, energy, large search space
[95]	2018	WAN	switch	No	✓	✓	✗	✗	✗	✗	✗	Hierarchical k-means	One topology analyzed shows higher latency and no reliability
[96]	2018	WAN	switch	No	✓	✓	✓	✗	✗	✗	✗	CD	Resilience to controller failure is not considered
[97]	2018	WAN	switch	Yes	✓	✓	✗	✗	✗	✓	✗	HC & BC	Uses only BC, neglecting traffic dynamics and node load
[98]	2018	WAN	switch	No	✓	✓	✓	✓	✗	✗	✗	RLMD	Does not consider dynamic load balancing or switch migration
[99]	2019	WAN	switch	Yes	✓	✗	✓	✗	✗	✗	✗	Salience-Based CP	Only evaluates cases with equal link weights
[100]	2019	WAN	switch	Yes	✓	✗	✗	✗	✗	✗	✗	Clustering-based heuristic	Focuses only on latency; ignores load and fault tolerance
[101]	2019	WAN	switch	No	✓	✓	✗	✓	✓	✗	✗	Constraint-based solution	Ignores latency variations and load balance, search space is large

Table 2.2 : continued

Ref.	Year	Net. Type	Cntrl Loc.	Cen.-Based	Objectives							Methods	Limitations and observations
					LA	LB	SC	RE	HL	ET	FR		
[102]	2019	IoT	switch	No	✓	✗	✓	✗	✗	✓	✗	Submodularity Optimization	Cost and load balance metrics are not considered, search space is large
[103]	2019	WAN	switch	No	✓	✗	✗	✗	✓	✗	✗	Dijkstra-Based Heuristic, K-Means Clustering, Kuhn-Munkres Algorithm	Focuses solely on delay; overlooks controller resilience, scalability, and dynamic traffic changes
[104]	2020	IoT	switch	No	✗	✗	✗	✗	✗	✓	✓	IQP, Heuristic	Excessive computation time limits scalability in large networks
[105]	2020	WAN	switch	No	✓	✓	✗	✗	✗	✗	✗	GWO	Controller failure, load balancing, and cost are not addressed
[106]	2020	WAN	switch	No	✓	✓	✗	✓	✗	✗	✗	Meta-heuristic, RALO	Balancing controller loads is not considered
[107]	2021	WAN	switch	No	✓	✗	✗	✓	✗	✗	✗	ILP	Scalability is limited by large search space and high execution time
[108]	2021	WAN	switch	Yes	✓	✗	✗	✓	✗	✗	✗	MIP	Assumes static topology and predefined attacks; limited scalability

Table 2.2 : continued

Ref.	Year	Net. Type	Cntrl Loc.	Cen.-Based	Objectives							Methods	Limitations and observations
					LA	LB	SC	RE	HL	ET	FR		
[109]	2021	IoT	switch	No	✓	✓	✓	✗	✗	✗	✗	DEWO	Adding more controllers results in increased latency
[110]	2021	IoT	switch	No	✓	✓	✗	✗	✗	✗	✗	ESFO, POCO	Limits controller count, uses one network, and considers only latency
[111]	2022	IoT	switch	No	✓	✗	✗	✗	✗	✗	✓	ANP, MCDM	Neglecting the number of controllers
[112]	2022	WAN	switch	No	✓	✗	✓	✓	✗	✗	✓	ANDP-based MCDM	Limited to six predefined controllers and static feature sets; lacks dynamic adaptability
[113]	2022	WAN	switch	Yes	✓	✓	✗	✗	✗	✓	✗	heuristic	Does not account for dynamic traffic variations during CP
[114]	2022	WAN	switch	No	✓	✗	✓	✓	✗	✓	✗	MIP, SA	The load balance metric is not considered, Large search space
[115]	2023	WAN	switch	No	✓	✗	✗	✗	✗	✓	✗	Programmability Explorer	Analysis omits latency variations, resilience, and load balancing
[116]	2023	IoT	switch	No	✓	✗	✗	✗	✗	✗	✗	US-ML, PAM	Random Distance Measures, considers only latency, fixed controllers
[117]	2023	WAN	switch	Yes	✓	✗	✗	✗	✗	✗	✓	Centrality-based heuristic	Only latency and flow rate is considred

Table 2.2 : continued

Ref.	Year	Net. Type	Cntrl Loc.	Cen.-Based	Objectives							Methods	Limitations and observations
					LA	LB	SC	RE	HL	ET	FR		
[118]	2023	WAN	switch	Yes	✓	✗	✗	✗	✗	✗	✗	Centrality-based heuristic	No comparison with existing switch to controller allocation methods
[119]	2023	WAN	switch	No	✓	✗	✗	✓	✗	✗	✗	PSO	Single topology analysis without cost and load balance metrics
[120]	2024	WAN	switch	No	✓	✗	✓	✗	✗	✗	✗	Density based splitting CP	Cost, load, and reliability metrics are not considered
[121]	2024	WAN	switch	Yes	✓	✗	✗	✗	✗	✗	✗	Clustering-based heuristic	Scalability issues and limited adaptability to diverse topologies
[122]	2024	SATNet	switch	Yes	✓	✓	✗	✗	✓	✗	✗	AVOA	Ignores hierarchical control and real-time adaptability
[123]	2024	IoT	switch	Yes	✓	✗	✓	✓	✗	✓	✗	CD	Fixed controller count, load balancing not considered

2.5 Analysis and discussion

Based on the previously discussed CPP solutions, we can draw the following conclusions.

- **Latency minimization is central.** Most CPP solutions prioritize reducing the switches to controllers latency, as well as among controllers themselves. This objective is crucial for maintaining network responsiveness, especially in delay-sensitive IoT applications.
- **Hybrid and heuristic approaches show promise.** As CPP is an NP-hard problem, heuristic, meta-heuristic, and hybrid algorithms such as PSO, GA, and clustering techniques have proven effective in generating near-optimal solutions achievable within acceptable computation times, particularly in large networks
- **Static vs. dynamic placement trade-offs.** Static controller placement strategies are computationally simpler but fail to adapt to traffic fluctuations, mobility, or failures. In contrast, dynamic placement approaches are more adaptable but incur higher overhead due to frequent recomputation and migration costs.
- **Underutilization of real-world constraints.** Many CPP solutions rely on synthetic topologies or ignore realistic constraints such as heterogeneous switch loads, limited controller capacities, or network failures, which limits their deployment feasibility in real-world SD-IoT networks.
- **Lack of unified performance benchmarks.** Comparative evaluations are often hindered by the absence of standardized performance benchmarks and datasets. This inconsistency makes it difficult to objectively assess and compare the effectiveness of different CPP approaches.

2.6 Challenges in the existing literature

Although numerous studies have targeted CPP, many challenges persist in the existing literature. These issues span various architectural levels and optimization paradigms, reflecting the complexity of achieving efficient and scalable controller placement in dynamic and heterogeneous SD-IoT environments. The following points highlight key open issues that still require attention for the development of robust and performance-aware CPP solutions:

- **Scalability in ultra-dense environments.** Most proposed solutions are not easily scalable to massive-scale IoT deployments, where the number of switches and traffic demand vary rapidly across time and space.

- **Neglect of controller resource constraints.** Many CPP approaches assume ideal controller capacities and ignore limitations in processing power, memory, and energy, which can lead to controller overload or degraded service quality.
- **Insufficient focus on security and fault tolerance.** Few studies consider controller placement strategies that are resilient to malicious attacks, link failures, or controller crashes, which are critical in mission-critical or public IoT infrastructures.
- **Limited multi-objective optimization.** Current works often optimize for a single metric (e.g., latency or cost), while real-world deployments require balancing multiple objectives.
- **Inadequate consideration of dynamic network conditions.** The majority of existing solutions assume static network topologies and do not account for node mobility, dynamic link failures, or time-varying traffic patterns inherent in SD-IoT environments.
- **Sparse integration with edge and fog paradigms.** Although edge and fog computing are increasingly integrated with SD-IoT, controller placement solutions rarely incorporate these paradigms into their design, missing opportunities to optimize locality-aware decisions.
- **Lack of real-world deployment and validation.** Most CPP proposals remain at the simulation level. There is a lack of real-world experimentation, testbeds, or deployment validation to assess the practical viability and robustness of these methods under realistic workloads and failure conditions.

2.7 Conclusion

In this chapter, we conducted a comprehensive review of the CPP in SDN, focusing on various strategies, network types, and evaluation criteria addressed in the literature. The comparative analysis revealed significant diversity in problem formulations and solution techniques, while also exposing persistent challenges such as scalability, dynamic network adaptation, and load-awareness. We also classified placement methods and examined their respective strengths and weaknesses in different SDN environments. These insights underline the need for more adaptable and efficient solutions to meet the demands of heterogeneous and large-scale networks. The next chapter builds on this foundation by presenting a detailed system model and formally defining the CPP in the context of our proposed approach.

3

System Modeling and Problem Formulation

3.1 Introduction

This chapter establishes a formal foundation for tackling the controller placement challenge in SD-IoT systems by clearly defining the problem and highlighting its NP-hard nature. We break CPP into three key components controller quantity, placement, and switch assignment and present a system model that includes heterogeneous traffic, capacity limits, and latency constraints. The chapter also introduces a variety of evaluation topologies (real, synthetic, and a custom Algerian network) and defines six performance metrics centered on latency, throughput, reliability, and runtime. These elements collectively underpin the optimization strategies developed in the following chapters.

3.2 Problem statement

As stated earlier in this thesis, our focus lies on solving the CPP in SD-IoT networks.This problem involves determining the number and placement of SDN controllers within a given network to achieve

optimal performance. In this section, we provide a detailed definition of the CPP, divide it into two major categories, and explain its complexity and relevance to real-world SD-IoT infrastructures.

3.2.1 Nature and complexity of the controller placement problem

The placement issue of SDN controllers is classified as an NP-hard problem. This categorization stems from its close relationship to the Facility Location Problem [127], a well-known NP-hard optimization problem in operations research. In the CPP, selecting locations for controllers from among potential sites can significantly impact performance metrics such as delay, fault tolerance, load balancing, scalability, and energy efficiency.

Seeking an optimal solution to the CPP is computationally expensive, particularly as the network size increases. With N switches and K potential controller locations, the solution space grows exponentially, making brute-force approaches impractical for large-scale networks. To visualize this complexity, we calculated the number of combinations using Eq. 3.1. The resulting Figure 3.1 shows that with 100 switches and 5 controllers, there are over 10^6 possible placements, while for 200 switches and 10 controllers, this number exceeds 10^{16}. This combinatorial explosion clearly illustrates the intractability of the problem as both the network size and the number of controllers increase. The problem becomes even more complex when heterogeneous traffic loads, varying link latencies, and fault tolerance constraints are introduced, further reinforcing the need for efficient heuristics or meta-heuristic algorithms to obtain near-optimal solutions in a scalable manner. Improper controller placement, on the other hand, can lead to:

- Increased average and maximum controller to switch latency,
- Higher controller to controller communication delays,
- Load imbalance across controllers,
- Reduced overall network responsiveness and resilience.

$$C\binom{N}{K} = \frac{N!}{K!\,(N-K)!} \tag{3.1}$$

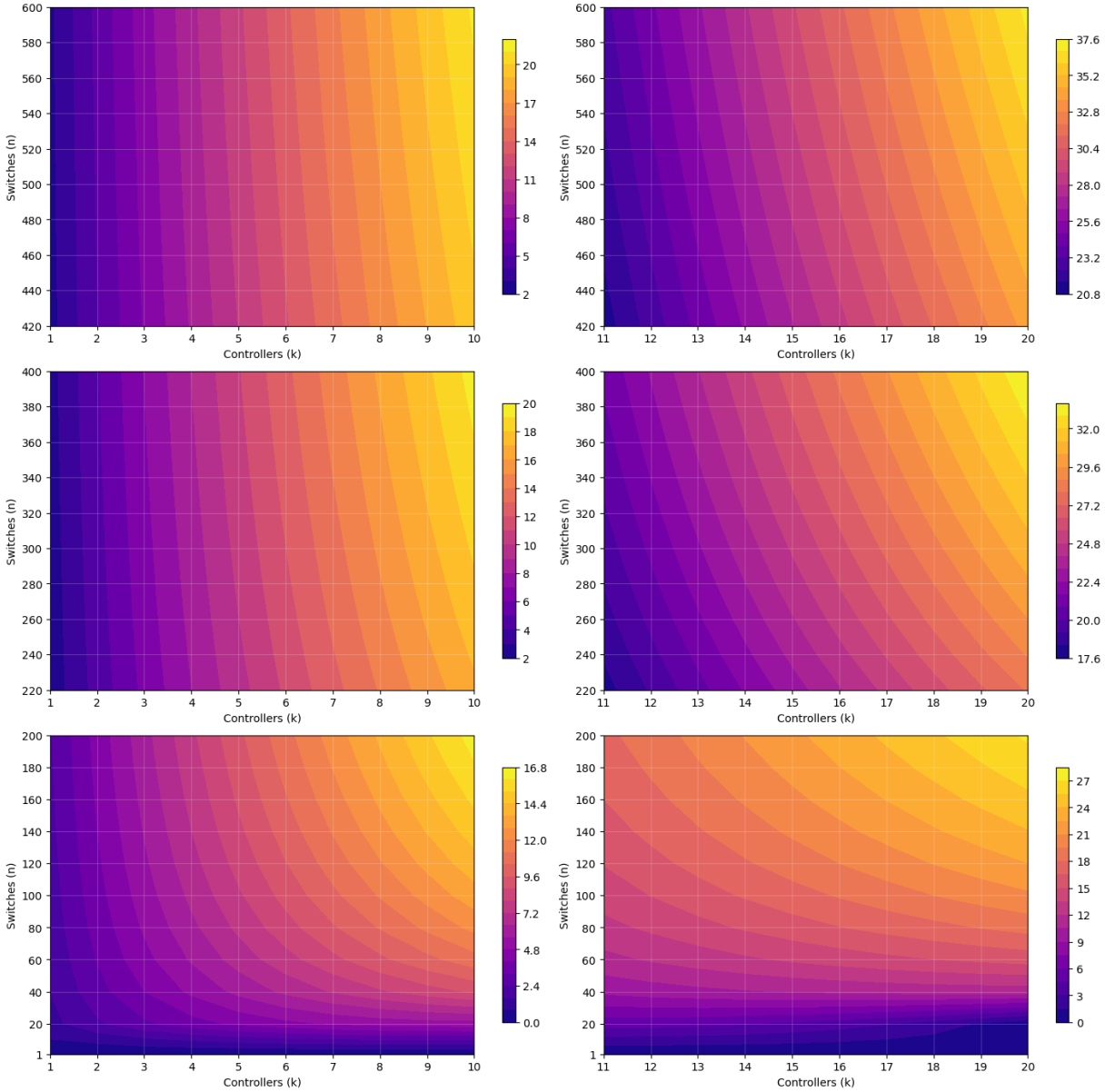


Figure 3.1: Number of combinations.

3.2.2 Controller placement problem: three optimization aspects

The control-plane placement issue is a critical design challenge in SDN-enabled IoT networks, as it directly influences the network's overall performance, responsiveness, and resource efficiency. At its core, this optimization challenge revolves around three interrelated and fundamental questions that must be addressed to ensure an effective and scalable control plane architecture. These questions guide both the structural layout of the network and the strategy for assigning switches to controllers. Specifically, it seeks to answer the following:

1. What is the optimal number of controllers required?

2. Where controllers can be deployed for best performance?

3. Which controller should each switch be assigned to?

Figure 3.2 illustrates this division and its relationship to single and multiple controller deployment models. In the following parts, we provide a detailed discussion of each CPP aspects.

1. β-Metric CPP: optimization of controller count

In cases where the number of controllers is not predetermined, the primary objective is to determine the minimal number of controllers required to satisfy specific performance constraints. This aspect of the problem is referred to as the β-Metric CPP. The optimization can follow a mono-objective approach targeting a single metric such as latency, reliability, energy efficiency, or deployment cost or a multi-objective formulation that considers multiple criteria simultaneously, allowing for trade-offs across different dimensions of network performance. The corresponding mathematical formulation is given below:

$$\min k \quad \text{subject to:} \begin{cases} \min\limits_{s \in S} \ \text{Metric}(s, C) \leq \beta, \\ \text{or} \\ \max\limits_{s \in S} \ \text{Metric}(s, C) \geq \beta \end{cases} \tag{3.2}$$

In this formulation, k denotes the number of controllers to be deployed, while β represents a predefined threshold for the selected performance metric, such as the maximum allowable latency or the minimum required reliability. The set S includes all switches in the network, and $\text{Metric}(s, C)$ denotes the value of the chosen performance metric for a given switch controller pair. The constraint ensures that the minimum or maximum value of this metric across all switches satisfies the threshold β, depending on whether the objective is to minimize the metric (as in latency) or maximize it (as in reliability).

2. k-Controller: optimizing placement with a fixed number of controllers

In this aspect of the CPP, the number of controllers k is fixed in advance, typically due to practical limitations. Given this constraint, the optimization goal is to determine the best locations for placing the k controllers within the network to enhance overall performance.

The optimization may be guided also by a mono-objective criterion such as minimizing average latency, maximizing reliability, reducing energy usage, or minimizing deployment cost or by a multi-objective formulation that balances trade-offs among these factors. The aim is to ensure efficient control plane coverage while respecting the fixed number of controllers. This aspect can be formally expressed as follows:

$$\min/\max \sum_{s \in S} \text{Metric}(s, C) \quad \text{subject to:} \quad |C| = k \tag{3.3}$$

In this formulation, C represents the set of selected controller locations, while the constraint $|C| = k$ ensures that exactly k controllers are deployed in the network. The term $\text{Metric}(s, C)$ denotes the value of the chosen performance metric for each switch s with respect to the set of controllers C.

3. Switch assignment optimization: mapping switches to controllers

The third interdependent aspect of the CPP addresses the optimal assignment of switches to the deployed controllers. Once the number and placement of controllers are determined either through the β-Metric or k-Controller formulations each switch must be associated with one controller in a way that preserves performance and balances system load. This aspect directly affects overall control plane efficiency, as poor assignments can result in increased latency, overload of certain controllers, or violation of reliability and QoS requirements. The optimization of this aspect is formally expressed by the following equation:

$$\min/\max \sum_{s \in S} \sum_{c \in C} x_{sc} \cdot \text{Metric}(s, c) \quad \text{subject to:} \quad \sum_{c \in C} x_{sc} = 1, \quad \forall s \in S \tag{3.4}$$

In this formulation, x_{sc} is a binary variable that equals 1 if switch s is assigned to controller c, and 0 otherwise. The term $\text{Metric}(s, c)$ denotes the cost, delay, or any other selected performance metric between switch s and controller c. The constraint $\sum_{c \in C} x_{sc} = 1$ ensures that each switch $s \in S$ is connected to exactly one controller $c \in C$.

3.2.3 Implications for single vs. multiple controller networks

The distinction between single and multiple controller configurations plays a pivotal role in shaping how the CPP is approached, especially across small, medium, and large-scale networks. In smaller networks, a single controller may suffice due to limited scale and simpler topology. However, as

networks grow in size and complexity, particularly in SD-IoT environments, distributing control functions becomes increasingly essential. This evolution directly ties into the three fundamental aspects of CPP: determining the appropriate number of controllers, selecting their optimal placement, and managing the switch to controller assignment strategy.

1. **Single controller.** The single controller configuration is most suitable for small-scale or low-demand networks, where centralized management is sufficient to meet latency and reliability constraints. In this model, all switches are connected to a single controller, making the assignment process deterministic and thereby eliminating the need for assignment optimization. Furthermore, since the number of controllers is fixed at one, the core challenge reduces to identifying the optimal physical or logical location for that controller. This scenario is formalized as the *1-Controller CPP*, where the primary objective is to minimize latency or maximize network performance through optimal placement. The simplicity of this formulation renders it computationally less intensive and well-suited for environments with modest scalability demands.

2. **Multiple controllers.** As the network scales up in size, complexity, or geographic distribution, a single controller becomes a bottleneck and a single point of failure. Larger and more dynamic networks require a distributed control architecture to ensure low-latency responses, fault tolerance, and efficient load balancing. In this more intricate setup, the CPP must address all three dimensions: determining how many controllers are needed (linked to performance and fault tolerance metrics), optimizing their geographic or topological distribution, and strategically allocating switches to controllers to minimize latency and balance control loads. This increases the computational and design complexity of the problem but allows for scalability and resilience. The necessity for multi-controller frameworks becomes especially apparent in mission-critical applications, where controller redundancy and proximity to switches can dramatically impact overall system performance and robustness.

Figure 3.2 illustrates the relationship between controller configurations and the three core aspects of CPP, highlighting the distinct implications of single and multiple controller scenarios.

3.2.4 Packet routing mechanism in SD-IoT

*In SD-IoT environments, efficient traffic handling depends on the dynamic reconfiguration of routing tables. Unlike traditional networks, which maintain relatively stable routes, SD-IoT enables flow

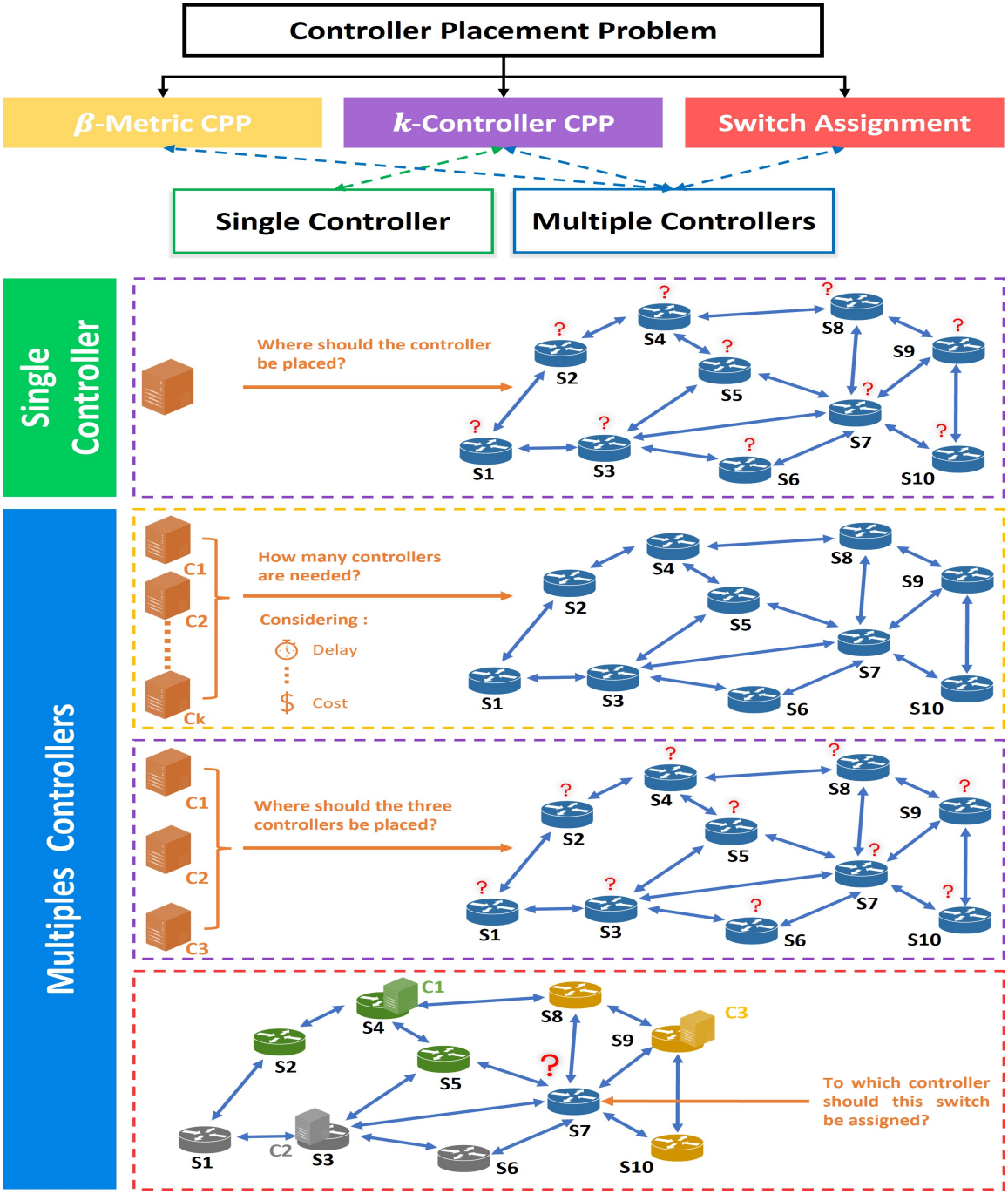


Figure 3.2: CPP aspects according to controller count, placement constraints, and switch allocation.

rules to be updated in real time in response to changing traffic patterns, security policies, or network topology. This adaptive behavior significantly enhances network responsiveness, scalability, and resilience. At the core of this process lies the interaction between the SDN controller and OpenFlow-enabled switches, where packet handling depends on whether the network is orchestrated by a single centralized controller or multiple distributed controllers.

In a single-controller architecture, all switches are directly connected to one central control node. When a data packet arrives at a switch, the forwarding decision is straightforward if a corresponding

rule exists in the switch's flow table the packet is immediately forwarded. However, in the case of a table miss, where no matching rule exists, the switch sends a Packet-In message to the controller. The controller, acting as the brain of the network, processes this request, determines the appropriate forwarding path, and responds with a Packet-Out message containing instructions or new flow rules. These are then installed on the switch to handle current and future packets. While this centralized approach simplifies control logic, it can become a bottleneck in larger deployments due to latency and controller overload.

By contrast, in a multi-controller architecture, typical of large-scale SD-IoT networks, switches are divided among several controllers based on predefined mappings or geographic domains. The local controller manages its assigned switches and handles table misses similarly to the single-controller case receiving Packet-In messages and responding with Packet-Out instructions. However, the complexity arises when packets need to traverse inter-domain paths, i.e., from one controller's domain to another. In such cases, the involved controllers must engage in controller to controller communication to compute end-to-end paths and ensure consistent policy enforcement. This distributed coordination enhances fault tolerance, reduces latency through localized control, and supports horizontal scalability, though at the cost of more complex synchronization and routing logic.

Figure 3.3 provides a comprehensive overview of the packet processing workflow under both single and multi-controller scenarios, emphasizing their operational distinctions and inter-controller communication patterns.

3.3 System model and assumptions

In this subsection, we present the system model used to represent the SD-IoT network, along with the key assumptions adopted for the controller placement analysis. The model defines the network structure, node roles, traffic behavior, and delay components that influence performance. These elements serve as the foundation for the problem formulation and simulation setup discussed in subsequent sections.

3.3.1 Network representation

The SD-IoT network is represented by an undirected graph $G = (V, E)$, where $V = S \cup C$. Here, $S = \{s_1, s_2, \ldots, s_n\}$ denotes the set of switch locations and $C = \{c_1, c_2, \ldots, c_k\}$ represents the set of potential controller sites, with $k \leq |V|$. The set E corresponds to the links (communication edges)

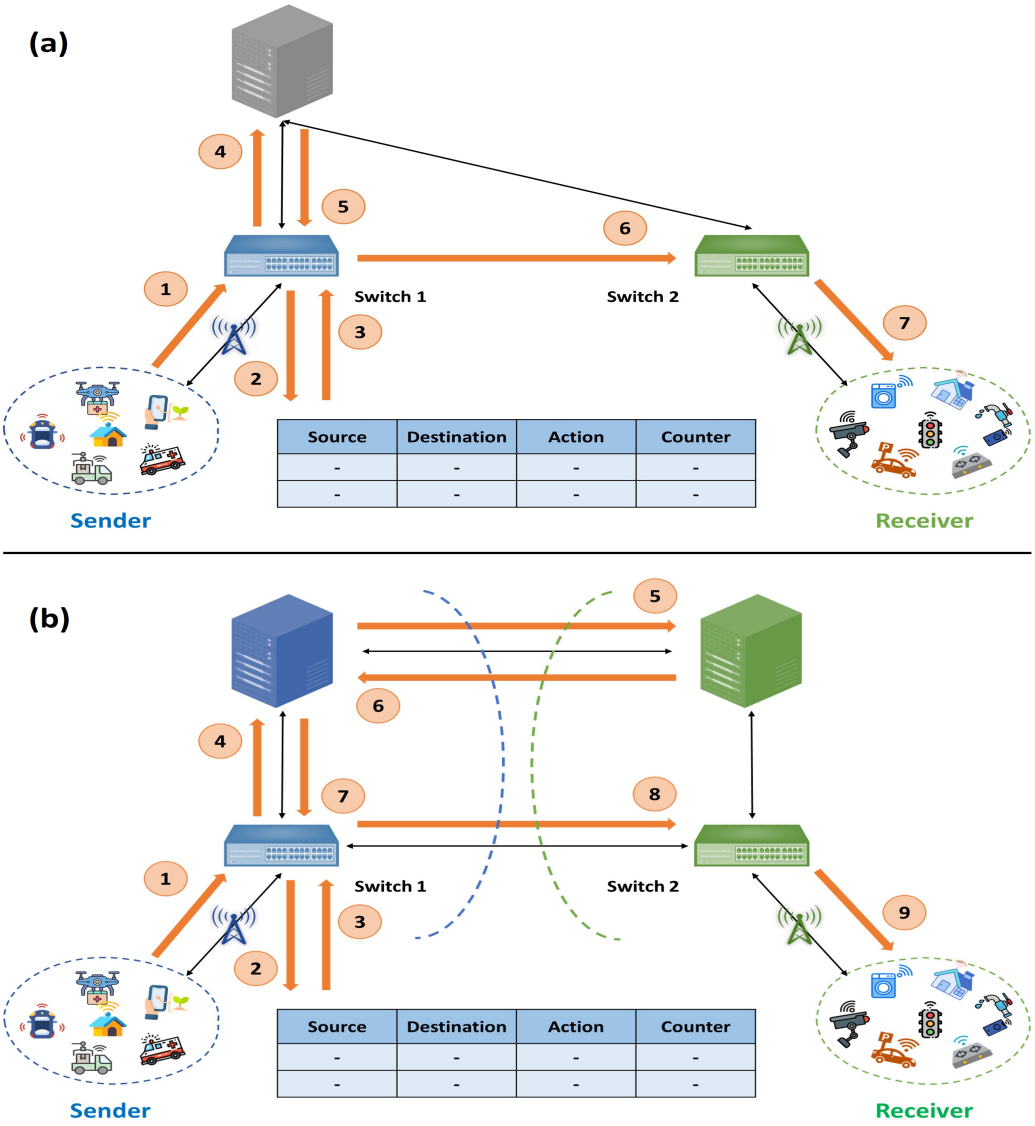


Figure 3.3: Packet processing workflow under single (a) and multi-controller (b) architectures.

connecting the nodes in the network. Each switch is uniquely assigned to a single controller and shares its location.

Let W_{s_i} represent the weight of node $s_i \in S$, corresponding to its traffic load, and let U_{c_j} denote the processing capacity of controller $c_j \in C$. The distance matrix $\texttt{sp\_dist} = [d_{i,j}]_{n \times n}$ contains the shortest path distances between every pair of switches in the network. Furthermore, Dsc denotes the minimum communication distance (or propagation latency) between a switch s_i and a controller placed at c_j.

Two latency metrics are commonly used to evaluate controller placements. The average switch to controller latency across the network is given by:

$$SC_{AvgLat}(G) = \frac{1}{K}\sum_{j=1}^{K}\frac{1}{\phi_{c_j}}\sum_{i=1}^{N} D_{sc} \times x_{i,j} \tag{3.5}$$

while the maximum switch to controller latency (i.e., the worst-case latency experienced by any switch) is defined as:

$$SC_{MaxLat}(G) = \frac{1}{K}\sum_{j=1}^{K}\max_{s_i \in S} D_{sc} \times x_{i,j} \tag{3.6}$$

In these expressions, $x_{i,j} \in \{0, 1\}$ is a binary variable indicating whether switch s_i is assigned to controller c_j, D_{sc} represents the propagation latency between switch s_i and controller c_j, and ϕ_{c_j} denotes the number of switches assigned to controller c_j.

This graph-based abstraction enables rigorous mathematical formulation and supports flexible modeling of network heterogeneity, link delays, and controller load, which are critical for realistic SD-IoT environments.

3.3.2 Node and link characteristics

In SD-IoT environments, the dynamic nature and density of connected IoT devices generate highly variable traffic flows across the network. This variability directly affects the workload distribution among switches. Some switches operate within regions of sparse IoT deployment and handle minimal traffic, while others are situated in dense zones where large volumes of sensor-generated data must be processed and forwarded. To accurately capture this heterogeneity, our model incorporates non-uniform switch loads, allowing each switch to reflect its local IoT traffic intensity.

Throughout our contributions, we adopt this heterogeneous switch load model to ensure a more realistic and performance sensitive evaluation. This approach allows us to better assess the impact of controller placement decisions on system behavior under uneven traffic conditions.

In addition, each link in the network is assigned a propagation delay that is assumed to be proportional to its physical length. This assumption reflects the fact that transmission latency grows with geographical distance, a critical factor in latency-aware controller placement strategies.

3.3.3 Controller features

Controllers in SD-IoT architectures are central entities responsible for managing data plane operations, computing forwarding decisions, and maintaining global network views. In our system model, each

controller is assumed to possess a finite processing capacity denoted by U_{c_j}, which limits the number of switches and flow requests it can handle simultaneously without performance degradation.

To reflect practical deployment scenarios, we consider a load balancing constraint such that no controller is overloaded beyond its capacity. This ensures fair distribution of switch to controller assignments and avoids bottlenecks. Furthermore, controllers are capable of communicating with each other over dedicated or shared links, which becomes essential in multi-controller deployments for tasks such as topology synchronization and inter-domain routing decisions.

All controllers are also co-located with switch nodes, implying that a controller is physically deployed at the same location as one of the network's switches. This co-location model reduces deployment cost and simplifies the controller switch connectivity assumptions.

To enforce load-aware operation, we ensure that the total load aggregated from the switches assigned to a given controller does not exceed its processing capacity. This condition is captured by the following constraint:

$$\sum_{i=1}^{N} W_{s_i} * x_{i,j} \leq U_{c_j} \quad \forall\, j \in C \tag{3.7}$$

Here, W_{s_i} represents the traffic load generated by switch s_i, $x_{i,j} \in 0, 1$ is a binary variable that specifies whether switch s_i is associated with controller c_j, and U_{c_j} denotes the processing capacity of controller c_j.

3.3.4 Communication delays

In all proposed approaches, the communication delay considered is the propagation latency, which is directly proportional to the physical distance between communicating nodes in the SD-IoT network. This metric provides a practical approximation of delay and is essential for evaluating controller placement strategies and the efficiency of control signaling.

To compute the distance between nodes, we adopt two different methods depending on the nature of the topology. For real-world topologies represented through geographic coordinates (longitude and latitude), we employ the Haversine formula to calculate the great-circle distance between any two nodes on the Earth's surface:

$$\text{Dist}(s_i, s_{i'}) = 2R \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_{i'} - \varphi_i}{2}\right) + \cos(\varphi_i) \cdot \cos(\varphi_{i'}) \cdot \sin^2\left(\frac{\lambda_{i'} - \lambda_i}{2}\right)}\right) \tag{3.8}$$

where $Dist$ is the geographic distance, R is the Earth's radius, $\varphi_i, \varphi_{i'}$ are the latitudes, and $\lambda_i, \lambda_{i'}$ are the longitudes of the two nodes.

For synthetic topologies, where nodes are generated on a 2D plane with (x, y) coordinates, we use the Euclidean distance to evaluate the physical proximity between any two connected switches:

$$\text{Dist}(s_i, s_{i'}) = \sqrt{(x_i - x_{i'})^2 + (y_i - y_{i'})^2} \tag{3.9}$$

Once the distances are obtained, we convert them into propagation latencies using the formula:

$$\text{PropagationDelay}(s_i, s_{i'}) = \frac{\text{Dist}(s_i, s_{i'}) \quad \text{(m)}}{2 \times 10^8 \quad \text{(m/s)}} \tag{3.10}$$

where the denominator approximates the propagation speed of signals in optical fiber.

In all our models and experiments, we account for two primary types of latency: the latency between a switch and its assigned controller, and the latency between controllers. For each case, we evaluate the average and worst-case (maximum) latencies, which are critical for determining the responsiveness, scalability, and synchronization capability of the SD-IoT control plane.

As the number of deployed controllers increases and their placement becomes more distributed across the network, the average distance between switches and their respective controllers generally decreases. This leads to a reduction in switch to controller communication latency and traffic volume. Conversely, with fewer and more centralized controllers, inter-controller communication tends to dominate due to the increased need for synchronization among distant controllers. Therefore, optimizing controller placement requires a careful balance between minimizing switch to controller and inter-controller communication overheads.

This trade-off is illustrated in Figure 3.4, which presents a toy example of an IoT network with six switches. The configuration on the left employs two controllers located at switches 1 and 2, whereas the configuration on the right utilizes three controllers placed at switches 1, 2, and 3. In the first scenario, the number of inter-controller (red) links is smaller, but the switch to controller (green) connections are longer and more numerous. Conversely, in the second scenario, adding an additional controller shortens the switch to controller paths while increasing the inter-controller communication overhead. The most suitable placement configuration ultimately depends on the relative communication cost between these two types of traffic, particularly in large-scale SD-IoT networks.

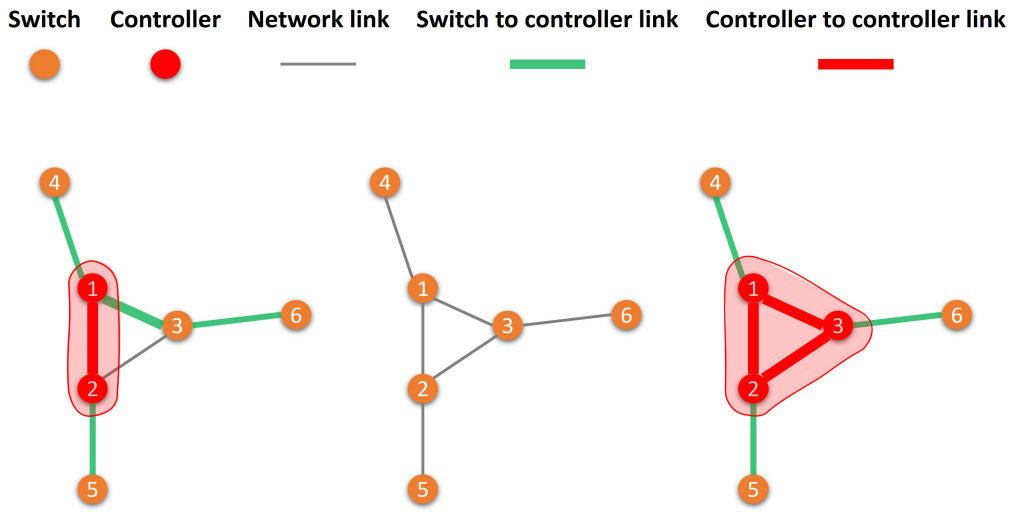


Figure 3.4: Illustrative example: Comparison between two different controller placement strategies. In the first case (left), controller positioning leads to higher C–S communication, while in the second case (right), C–C communication becomes more dominant. Green lines denote C–S connections, and red lines represent C–C connections, with line widths proportional to the corresponding communication load.

3.3.5 Traffic load model

In the literature, numerous approaches addressing the CPP assume a homogeneous traffic model, where all switches are treated as generating an equal amount of data traffic. While this simplification facilitates analytical modeling and reduces computational complexity, it does not reflect the traffic diversity typically observed in real-world SD-IoT environments. Some recent studies have begun to explore heterogeneous traffic models, assigning different load levels to switches based on arbitrary or domain-specific criteria. However, these approaches often fall short of incorporating the traffic load variability directly into the latency-aware optimization formulations of the CPP.

To address this gap, our contributions adopt a more realistic and adaptive modeling of switch-generated traffic loads by exploring two complementary strategies:

- **Random load assignment**. In the first approach, we assign traffic loads to switches randomly, following a bounded distribution to simulate generic heterogeneous environments without prior spatial knowledge.

- **Population-based load generation.** The second, more context-aware method, derives the traffic load at each switch from the population density of the area it serves. This assumption is grounded in the IoT paradigm, where each individual is likely to own or interact with multiple IoT devices (e.g., smartphones, wearables, smart appliances). Consequently, the expected traffic load is

considered proportional to the number of individuals within a switch's coverage zone. This model provides a more realistic representation of urban and densely populated deployments, particularly in smart city or healthcare monitoring scenarios.

By integrating these heterogeneous load profiles into our optimization framework, we aim to enhance the precision and scalability of CPP solutions, especially when balancing controller workloads and minimizing latency in high-density SD-IoT deployments.

3.3.6 Assignment constraints

The CPP in Software-Defined IoT networks must adhere to a set of essential constraints to guarantee the feasibility, efficiency, and stability of the control plane. These constraints ensure proper mapping between switches and controllers, enforce capacity limitations, and maintain logical consistency across the network. In this work, we consider the following assignment constraints:

First, each switch must be assigned to exactly one controller, ensuring that every switch has a designated control point and that no switch is left unmanaged. This is expressed as:

$$\sum_{j=1}^{K} x_{i,j} = 1 \quad \forall\, i \in S \tag{3.11}$$

Second, each controller must be deployed at exactly one switch location, preventing multiple controller instances from being assigned to the same position:

$$\sum_{i=1}^{N} y_{i,l} \leq 1 \quad \forall\, l \in C \tag{3.12}$$

Third, the cumulative traffic load assigned to a controller must not exceed its processing capacity, which safeguards against overloading and performance degradation. This capacity constraint is formalized in Eq. (3.7).

Lastly, from a logical feasibility perspective, the number of controllers K deployed in the network must not exceed the number of available switch locations $|S|$. This constraint ensures that controllers can only be co-located with valid network nodes:

$$K \leq |S| \tag{3.13}$$

Together, these constraints form the foundational rules for a viable controller placement strategy that is consistent with the physical and logical structure of SD-IoT networks. They are integrated into

all optimization models discussed throughout our work to ensure that solutions are both operationally sound and practically deployable.

3.4 Network topologies for evaluation

To comprehensively evaluate the proposed solutions for the CPP in SD-IoT environments, our contributions rely on a wide range of network topologies varying in size, type, and structural complexity. These topologies include both real-world and synthetically generated networks, ensuring that the analysis is both practically grounded and theoretically robust.

We consider three network scales: small, medium, and large topologies, in order to test the scalability and performance of our models under varying conditions. Two main categories of topologies are used:

- **Real-world topologies.** extracted from the widely-used Internet Topology Zoo (ITZ) dataset, which provides a curated collection of network topologies sourced from service providers, universities, government agencies, and multinational corporations. These topologies capture the realistic structure of operational networks across different geographic scopes, such as intra-city, inter-city, and even inter-country backbones. The diversity in topology sizes, connectivity, and geographic layouts makes them ideal for validating the adaptability and resilience of our proposed algorithms.

- **Synthetic topologies.** which are programmatically generated using custom Python scripts. Two node positioning strategies are adopted:

  - ***Uniform distribution.*** switches are evenly spaced across the region, ensuring balanced coverage.

  - ***Random distribution.*** switches are placed without a fixed pattern, producing areas with higher and lower densities across the network.

For these synthetic networks, the node coordinates are randomly assigned in a 2D space, and connectivity is established based on distance thresholds to ensure overall network connectivity. This allows for controlled experimentation across different network densities and link configurations.

In addition to leveraging existing real-world data, one of our key contributions is the design of a novel national-level topology representing Algeria. Inspired by the ITZ structures, this topology

models each Algerian state (wilaya) as a node, with links established based on shared borders and estimated traffic significance. The aim is to create a realistic testbed tailored to national-scale SD-IoT deployment scenarios. This model enables a contextual performance evaluation of controller placement strategies in the context of Algerian infrastructure and population distribution.

A complete description and visual representation of all topologies used both synthetic and real is provided in the following chapters, including node and link counts, spatial dimensions, and specific use cases for each topology.

3.5 Performance metrics

In order to evaluate the performance of the proposed controller placement strategies in SD-IoT networks, we use six complementary performance metrics, with a particular emphasis on latency. These metrics collectively capture responsiveness, traffic handling capacity, resilience, and computational feasibility vital factors for real-world SD-IoT deployments.

- **Switch to controller average latency.** The mean delay experienced by switches when communicating with their assigned controller. This metric reflects the overall responsiveness of the network and is critical for time-sensitive IoT applications.

- **Switch to controller maximum latency.** The highest observed delay among all switch to controller paths. It serves as a worst-case performance measure, ensuring that even the farthest or densest switches maintain acceptable latency bounds.

- **Controller to controller average latency.** The average delay between all pairs of deployed controllers. This indicates the efficiency of inter-controller communication, which is essential for network synchronization, global view sharing, and distributed decision-making.

- **Switch to controller average flow rate.** Calculated as the total traffic flow divided by the total latency across all switch controller links, this metric quantifies the network's throughput efficiency how well the control plane supports data traffic relative to the incurred latency.

- **Reliability under controller failure.** The proportion of switches that maintain connectivity to an operational controller after the failure of one controller. This measures the resilience of the placement strategy against single-controller failures and informs redundancy planning.

- **Execution time.** The runtime required by placement algorithms to compute controller deployment and assignment. This metric evaluates computational scalability and determines the practical suitability of each method for deployment in large-scale networks.

3.6 Conclusion

We have laid out a rigorous modeling framework and formulated the CPP with mathematical precision, capturing all relevant constraints and performance factors. By integrating realistic network attributes and diverse topologies, we set a solid stage for empirical assessments. The six selected metrics enable comprehensive performance evaluation of controller placement schemes. Building on this foundation, subsequent chapters will introduce our original optimization algorithms designed to deliver scalable, efficient, and resilient CPP solutions in SDN-enabled IoT environments.

Part II

Contributions

4

Exact Controller Placement in SD-IoT: From Free-Space to Realistic Topology Design

4.1 Introduction

The efficient placement of controllers is a fundamental challenge in SD-IoT networks, directly impacting performance, scalability, and resilience. While numerous studies have addressed this problem under simplified or idealized conditions, there remains a significant gap in integrating both realistic traffic heterogeneity and topology constraints in the controller placement process. This chapter presents a twofold contribution. First, we propose a free-space placement approach that incorporates heterogeneous traffic loads to optimize controller location beyond the physical positions of switches. Second, recognizing the limitations of abstract models, we extend the study to a realistic, topology-aware placement for SD-IoT using the Algerian national network as a representative case study. Through this methodology, we demonstrate how combining traffic awareness and topology constraints leads to more efficient and practical controller deployment strategies.

4.2 Single Controller Deployment in Free-Space SD-IoT under Heterogeneous Traffic Conditions

To initially demonstrate the effectiveness of considering heterogeneous switch loads in the controller deployment process, we begin by determining the optimal placement of a single controller within the network area. Unlike traditional approaches where the controller is restricted to existing switches, this preliminary study allows the controller to be positioned freely within the geographical space, independent of switch locations. This free-space scenario provides an ideal benchmark to assess performance under optimal conditions.

4.2.1 Problem statement and motivation

The CPP is a critical design consideration in Software-Defined IoT networks, directly impacting network performance in terms of latency, scalability, control overhead, and fault tolerance. Existing research on CPP, whether targeting traditional SDN environments [86, 89, 98, 103, 111, 112, 119, 128], predominantly adopts a restricted search space approach, where the controller's position is selected exclusively from a predefined set of locations corresponding to existing network switches. While this methodology simplifies the search space and aligns with practical deployment constraints, it imposes significant limitations on finding the true optimal placement, especially in large-scale or geographically distributed IoT networks.

Restricting controller placement solely to predefined node locations may lead to suboptimal network performance. This limitation neglects the underlying geographical layout and spatial distance distribution within the deployment area, potentially resulting in inefficient controller positioning. Furthermore, it prevents the exploitation of alternative infrastructure or free-space locations that could offer improved latency and enhanced traffic load distribution. In addition, such a restriction fails to account for dynamic network characteristics, particularly heterogeneous traffic demands originating from different regions, which can significantly influence overall system efficiency and responsiveness.

Recent works, such as [88], have recognized these limitations and proposed the open search approach, which allows the controller to be placed anywhere within the geographical area, independent of switch positions. This approach is increasingly feasible due to advancements in remote sensing technologies and semantic segmentation of satellite imagery [129], which enable precise identification and classification of sub-regions within the deployment area for potential controller installations. Such methods provide the network designer with unprecedented flexibility, opening opportunities to

optimize controller placement beyond conventional node-constrained models.

Nevertheless, even state-of-the-art open search methods often neglect a crucial aspect of real-world IoT networks: the heterogeneity of traffic loads across switches. IoT environments are inherently characterized by varying device densities, application requirements, and data generation patterns. Some switches serve as gateways for highly active clusters of devices, leading to significantly higher traffic loads, while others experience relatively light traffic. Ignoring this heterogeneity in controller placement decisions can result in suboptimal controller to switch latencies for heavily loaded nodes, imbalanced utilization of the control plane, and an increased number of controllers required to satisfy latency or fault-tolerance constraints.

To clearly illustrate this gap and motivate our contribution, we present a conceptual example, depicted in Figure 4.1, of a simple SD-IoT network consisting of four fully connected switches (S1, S2, S3, and S4) arranged at the corners of a square area with a side length of 200 meters. Table 4.1 summarizes the coordinates and traffic load (expressed as node weights) for each switch. The traffic loads are intentionally heterogeneous, with switch S2 handling a significantly higher volume of flows than the others. We analyze three distinct scenarios to evaluate the impact of controller placement strategies:

1. **Restricted search approach *(Scenario 1).*** The controller is restricted to one of the switch positions, as in conventional methods. The optimal among these locations is chosen based on minimizing switch to controller latency.

2. **Open search without load awareness *(Scenario 2).*** The controller can be freely placed within the area, following the method proposed in [88], but without considering traffic heterogeneity.

3. **Proposed open search with traffic load consideration *(Scenario 3).*** We introduce a novel approach that combines the flexibility of open search with an explicit consideration of heterogeneous switch traffic loads to optimize controller placement decisions.

Table 4.1: Information on the topology for Figure 4.1.

Labels/Placements	**S1**	**S2**	**S3**	**S4**	$C_{(B)}$	$C_{(C)}$
x-coordinate	0	200	0	200	100	150
y-coordinate	200	200	0	0	100	150
Load (switch)	100	500	100	100	/	/

The comparison is conducted based on three key performance indicators: latency, the number of controllers required, and traffic load distribution. The results, summarized in Table 4.2, clearly

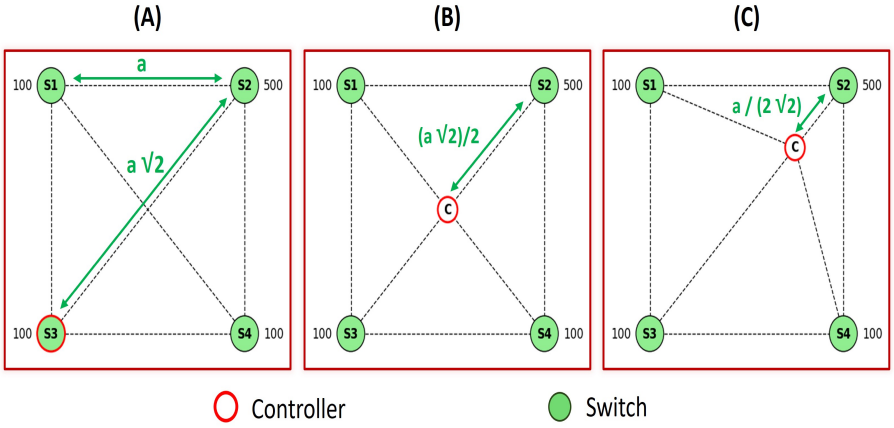


Figure 4.1: SDN switch traffic loads and controller placement effects.

demonstrate the impact of different controller placement strategies on overall network performance and management efficiency.

- **Latency reduction.** In Scenario 1, restricted to node locations, the worst-case (maximum) switch to controller latency is $5\times\mathbf{200}\times\sqrt{2}$ (ms). Scenario 2 achieves better latency by positioning the controller at the geometric center, reducing the worst-case latency to $5 \times \mathbf{100} \times \sqrt{2}$ (ms). Our approach in Scenario 3, by prioritizing proximity to the heavily loaded switch (S2), further reduces the worst-case (maximum) latency to $5 \times \mathbf{50} \times \sqrt{2}$ (ms), effectively halving the delay compared to Scenario 2.

- **Controller count optimization.** To satisfy a maximum latency constraint of $5 \times 50 \times \sqrt{2}$ (ms), the restricted search approach requires deploying at least two controllers at different switch locations. The open search method without traffic awareness still necessitates more than one controller. In contrast, our proposed approach achieves the same performance threshold with only a single controller, demonstrating improved deployment efficiency.

- **Traffic load distribution.** In Scenario 1, the maximum achievable flow rate is $\frac{500}{5\times\mathbf{200}\times\sqrt{2}}$. Scenario 2 increases this to $\frac{500}{5\times\mathbf{100}\times\sqrt{2}}$, while our load-aware placement in Scenario 3 nearly doubles it to $\frac{500}{5\times\mathbf{50}\times\sqrt{2}}$, highlighting improved controller utilization and enhanced traffic-handling capability.

These results underscore the limitations of both the restricted search and traffic-agnostic open search methods. Our proposed approach addresses these gaps by combining the flexibility of free-space placement with an explicit consideration of heterogeneous traffic loads, leading to tangible improvements in latency, controller efficiency, and overall network performance. This illustrative

Table 4.2: Analytical comparison of the three cases shown in Figure 4.1.

Metrics	**Case (A)**	**Case (B)**	**Case (C)**
Maximum Latency (ms)	1414.21	707.10	**353.55**
Average Latency (ms)	604.73	282.84	**110.23**
Maximum Load Rate (Flows/ms)	0.35	0.71	**1.42**
Average Load Rate (Flows/ms)	0.45	0.71	**0.78**

example highlights the necessity of incorporating switch traffic loads into the controller placement process, a consideration that is formally addressed and evaluated in the subsequent sections.

4.2.2 Proposed approach

The proposed method aims to jointly optimize both the latency between switches and the controller, as well as the efficient handling of heterogeneous traffic loads within an SD-IoT environment. The approach follows a two-phase process: first, it performs a location-independent identification of potential controller placements, where the search space is extended beyond switch locations to encompass the entire network area; second, it conducts a traffic-aware optimization of controller placement, integrating switch-specific traffic loads into the placement decision to enhance overall network performance. For clarity, the proposed method is illustrated and validated using a single-controller scenario within a given SD-IoT domain.

A) Location-independent identification of potential controller placements

The first phase eliminates the limitations of traditional switch-constrained placement by enabling the controller to be positioned anywhere within the deployment area. This follows the general principles of the location-independent controller placement model initially proposed in [88].

The procedure starts by geometrically characterizing the network. In this step, Welzl's algorithm [130] is employed to determine the Smallest Enclosing Circle (SEC) that encompasses all network switches. The SEC is defined by its center coordinates (x, y) and radius R, representing the maximum feasible area for potential controller placements.

To systematically generate candidate locations for the controller within this region, we follow a structured sampling approach based on radial distributions. We define a distance parameter D_{stop} determining the spacing between successive concentric circles within the SEC, computed as:

$$D_{stop} = \frac{R}{n} \tag{4.1}$$

Here, n denotes the desired number of subdivisions, controlling the granularity of potential placements.

Next, for each concentric circle generated within the SEC, candidate controller positions are uniformly distributed along its circumference. The angular separation between two consecutive positions on the same circle is given by:

$$\text{Angle} = \frac{D_{stop} \times 360^\circ}{2 \times \pi \times R} \tag{4.2}$$

This systematic placement strategy produces a comprehensive set of potential controller P_C locations, independent of switch positions, while ensuring spatial uniformity. Figure 4.2 visually illustrates an example of this process.

The entire procedure is formalized in the flowchart depicted in Figure 4.3(A). The method takes as input a graph $G(S, E)$, where S represents the set of switches and E the set of links, abstracting the network topology for the domain under consideration.

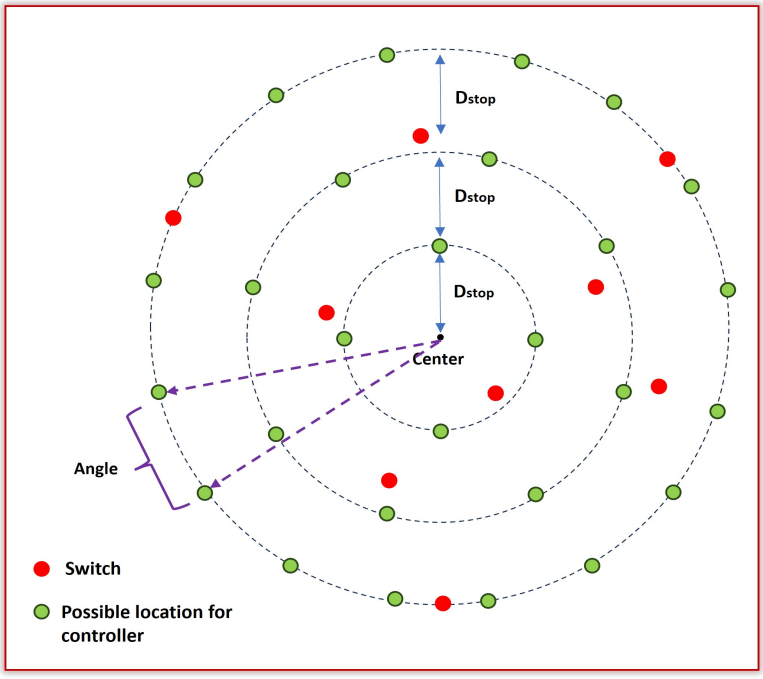


Figure 4.2: Example of a possible controller location generated according to the steps in Figure 4.3 (A).

B) Traffic-aware optimization of controller placement

Once the set of feasible controller locations is established, the second phase focuses on identifying the optimal placement by incorporating the heterogeneous traffic loads generated by switches.

The process begins by assigning a specific traffic load to each switch, reflecting its role and activity within the IoT network. Due to the variability of traffic loads across switches, normalization is required to ensure comparability. The normalized node weight W_{s_i} for each switch s_i is calculated using the Min-Max Normalization formula below:

$$W_{s_i} = y_{min} + \left(\frac{X_{s_i} - x_{min}}{x_{max} - x_{min}} \right) \times (y_{max} - y_{min}) \tag{4.3}$$

Where X_{s_i} denotes the raw traffic load of switch s_i, x_{min} and x_{max} represent the minimum and maximum traffic loads in the network, and y_{min} and y_{max} define the target normalization range that ensures all node weights remain within consistent bounds. This transformation standardizes traffic load values, facilitating their direct integration into the placement optimization process.

The objective is to determine the controller location that minimizes the maximum switch to controller latency, taking into account the normalized traffic load of each switch. The optimization problem is formulated as follows:

$$SC_{MaxLat}(G, P_C) = \min_{c_j \in P_C \cup S} \max_{s_i \in S} \left(D_{sc} \times W_{s_i} \right) \tag{4.4}$$

Where D_{sc} denotes the distance (or latency) between switch s_i and the candidate controller position c_i, and W_{s_i} represents the normalized weight corresponding to the traffic load of switch s_i. The set $P_C \cup S$ represents the union of all generated possible controller locations and the list of switches. The outer minimization seeks the controller location that results in the lowest maximum weighted latency across all switches.

The flowchart in Figure 4.3 (B) summarizes this selection process, ensuring that the final controller placement achieves a balanced compromise between minimizing latency and effectively handling heterogeneous switch traffic.

4.2.3 System model

To assess the effectiveness of the proposed approach, extensive simulations are performed using real-world network topologies obtained from the ITZ [131]. Specifically, five distinct network topologies

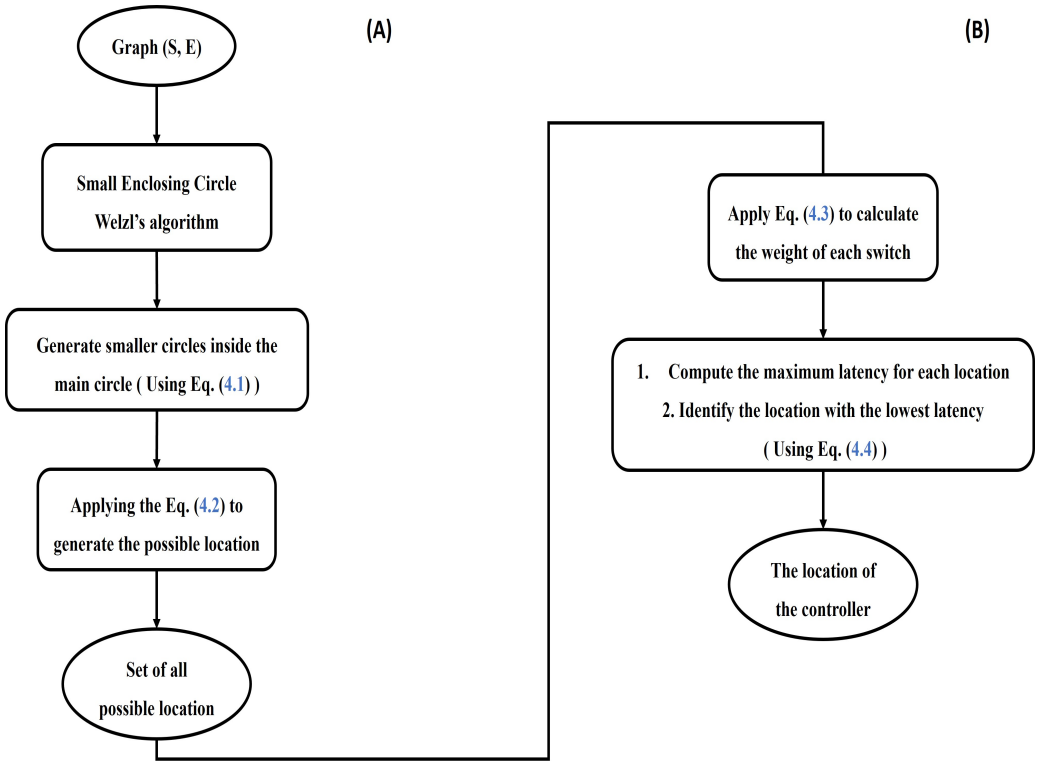


Figure 4.3: Left (A): Switch Location-Independent algorithm for controller placement; Right (B): Selecting the optimal controller placement considering the heterogeneous switch flows.

are considered: GridNet, Abilene, GoodNet, PsiNet, and OS3E. The structural characteristics of these topologies, including the number of nodes and edges, are summarized in Table 4.3, while their graphical representations are depicted in Figure 4.4.

Table 4.3: Characteristics of the network topologies used.

Topology	**Number of Nodes**	**Number of Edges**
GridNet	9	20
Abilene	11	14
GoodNet	17	31
PsiNet	24	25
OS3E	34	42

In alignment with realistic IoT deployment scenarios, we assume a heterogeneous traffic environment. The maximum traffic generation capacity of each switch is assigned randomly within a predefined range of 0.04 to 0.4 million flows per second (MFlows/s), consistent with traffic profiles used in [86]. The upper bound of 0.4 MFlows/s represents the maximum potential flow generation per switch within the network.

For the purpose of this study, the entire network is considered a single administrative domain, requiring the deployment of only one controller to manage the control plane. This setting allows for

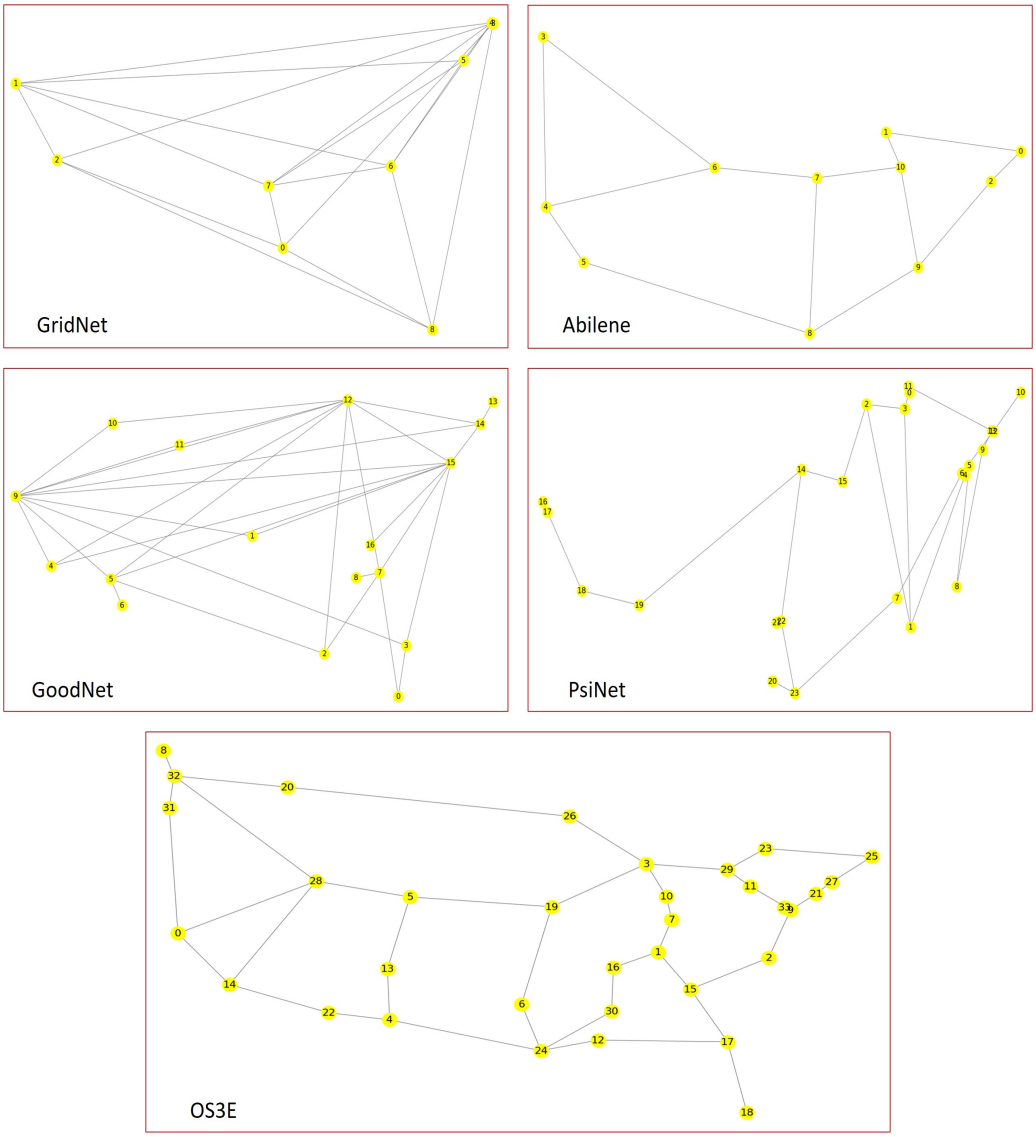


Figure 4.4: Visualization of the five network topologies considered in the simulations.

the evaluation of our proposed method in scenarios where centralized control is maintained. The proposed Heterogeneous Traffic Flow–based Controller Placement (HTF-CP) method is implemented and compared against two benchmark approaches:

- **Lidy+ approach** [88], representing an open-search controller placement technique without traffic awareness.
- **Random placement**, where the controller location is selected arbitrarily within the network area.

All simulations are executed using Python 3.12.0 within a computational environment running Windows 10 (64-bit) and equipped with an Intel Core i7 processor, 16 GB of RAM operating at 2667 MHz, and a 1.6 series graphics card. The simulation setup is based on several assumptions. The

distance between switches is known and is considered directly proportional to the communication latency. Each switch exhibits a heterogeneous maximum flow generation capacity, reflecting realistic IoT traffic patterns. The network coordinates of all switches are precisely defined and known. For the potential controller placements, the number of concentric subdivisions in (4.1) is fixed at $n = 10$. The performance of the proposed HTF-CP approach is evaluated using key network performance metrics, namely the maximum and average latency between switches and the controller, as well as the average traffic load rates managed by the controller.

After applying the method described in [88], the number of potential controller locations generated for each concentric circle, with n = 10 subdivisions, is presented in Table 4.4. This distribution ensures both fine-grained spatial coverage near the center and an expanded search space toward the network boundaries.

Table 4.4: Number of potential controller placements per circle.

Circle index	1	2	3	4	5	6	7	8	9	10	**Total**
Potential placements	6	12	18	25	31	37	43	50	59	62	**340**

Comparative analysis with the Lidy+ and random placement approaches provides insights into the advantages of integrating traffic heterogeneity awareness into the controller placement process. The results and discussion, presented in subsequent sections, demonstrate the capability of the proposed method to enhance overall network performance.

4.2.4 Results and discussion

This section presents a detailed performance analysis of the proposed controller placement method against existing solutions. Through simulations conducted on five realistic topologies, we analyze key performance indicators including maximum latency, average latency, and network load rate to demonstrate the effectiveness of our approach.

A) Switch to controller maximum latency

The switch to controller maximum latency (denoted as SC_{MaxLat}) reflects the worst-case communication delay between the deployed controller and the farthest switch in the network. This metric is computed using the maximum normalized distance between the controller and all switches, as defined in *Eq. (4.4)*.

Figure 4.5 presents a comparative analysis of SC_{MaxLat} across the five selected network topologies using our proposed approach, the Lidy+ method, and a Random placement baseline. The results clearly demonstrate that our method consistently achieves the lowest maximum latency across most topologies. This improvement is attributed to the traffic-aware placement strategy, which prioritizes positioning the controller closer to high-load switches.

Moreover, as the network size increases from GridNet to OS3E a natural growth in maximum latency is observed across all approaches, consistent with larger geographic coverage and higher node density. However, despite this increase, our approach maintains superior performance, keeping SC_{MaxLat} lower than both Lidy+ and Random placements. An exception is noted for the PsiNet topology, where the performance of Lidy+ approximates that of our approach, highlighting topology-specific behavior that could merit further investigation.

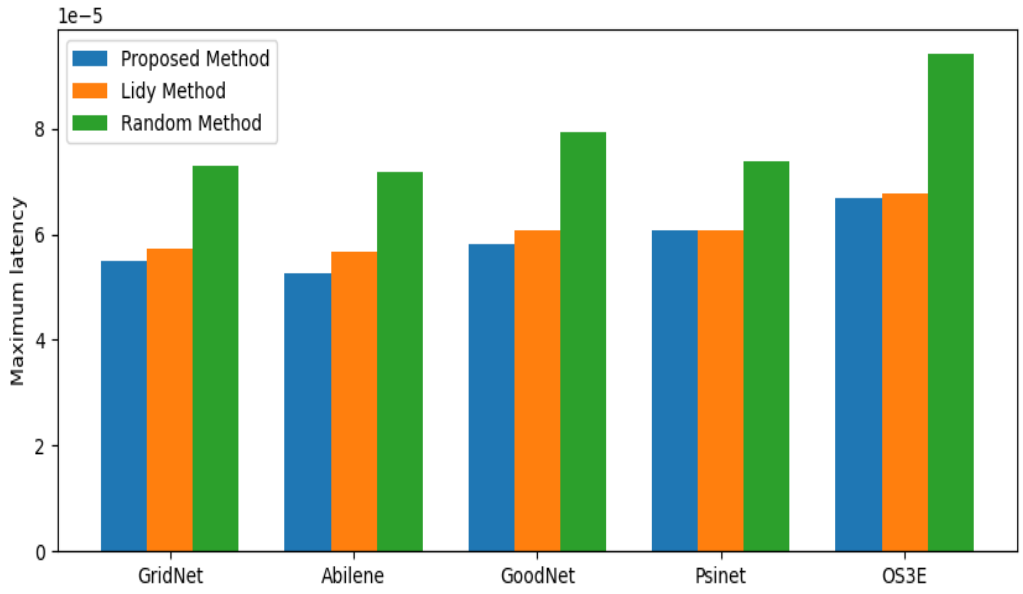


Figure 4.5: Comparison of maximum switch to controller latency across five topologies.

B. Switch to controller average latency

In addition to SC_{MaxLat}, evaluating the average latency provides insight into the overall communication efficiency between the controller and all switches. The average switch to controller latency (SC_{AvgLat}) is computed after obtaining the best controller placement B_c using *Eq. (4.4)*, as the weighted sum of latencies from each switch to the controller, normalized by the total number of switches, as expressed in *Eq. (4.5)*.

$$SC_{AvgLat}(G, B_c) = \frac{1}{|S|} \times \sum_{i=1}^{|S|} d(s_i, B_c) \times W_{s_i} \tag{4.5}$$

Figure 4.6 displays the results for SC_{AvgLat} across the five networks under the three placement strategies. As expected, the Random approach yields the highest average latency across all topologies, confirming the inefficiency of uncontrolled controller placement. The Lidy+ method achieves better performance by leveraging location-independent placement, yet it overlooks traffic heterogeneity.

In contrast, our proposed method systematically outperforms both baselines by incorporating switch traffic loads into the placement decision. This ensures that switches with high traffic demands experience reduced communication delays, leading to a lower overall average latency. The performance gap between our approach and Lidy+ becomes more pronounced as network complexity increases.

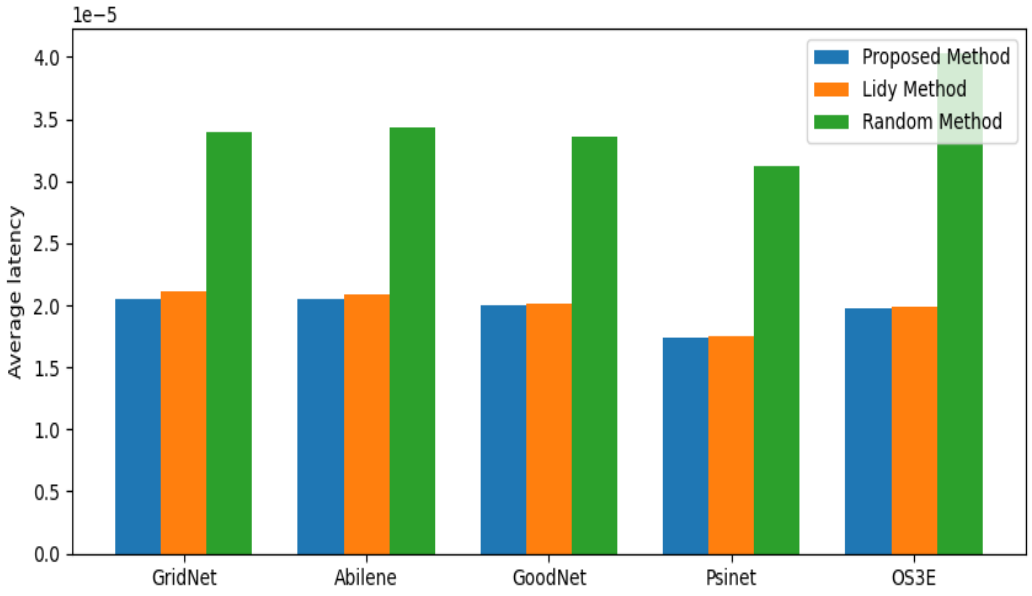


Figure 4.6: Comparison of average switch to controller latency across five topologies.

C) Average load rate

Given that traffic loads are heterogeneous across switches, an effective controller placement strategy should enhance the network's overall load handling capacity. To assess this, we compute the average load rate using *Eq. (4.6)*, reflecting the system's ability to process flows efficiently based on controller placement.

$$SC_{AvgFlowRate}(G, B_c) = \frac{Total\,flow}{SC_{AvgLat}(G, B_c)} \tag{4.6}$$

Figure 4.7 summarizes the comparative results for the average load rate using our proposed method, Lidy+, and the Random approach across the five networks. It is evident that our method consistently achieves the highest load rate in all topologies. This improvement stems from the strategic placement of the controller near switches with higher traffic demands, thereby optimizing resource utilization and reducing bottlenecks.

The Random approach exhibits the poorest performance due to arbitrary placements that often neglect high-demand regions of the network. The Lidy+ method shows moderate improvements by exploring location-independent placement but remains sub-optimal in scenarios where traffic heterogeneity significantly influences network performance.

In summary, the results confirm that integrating traffic-awareness into controller placement yields substantial benefits in terms of both latency reduction and traffic load handling, reinforcing the practical advantages of our approach.

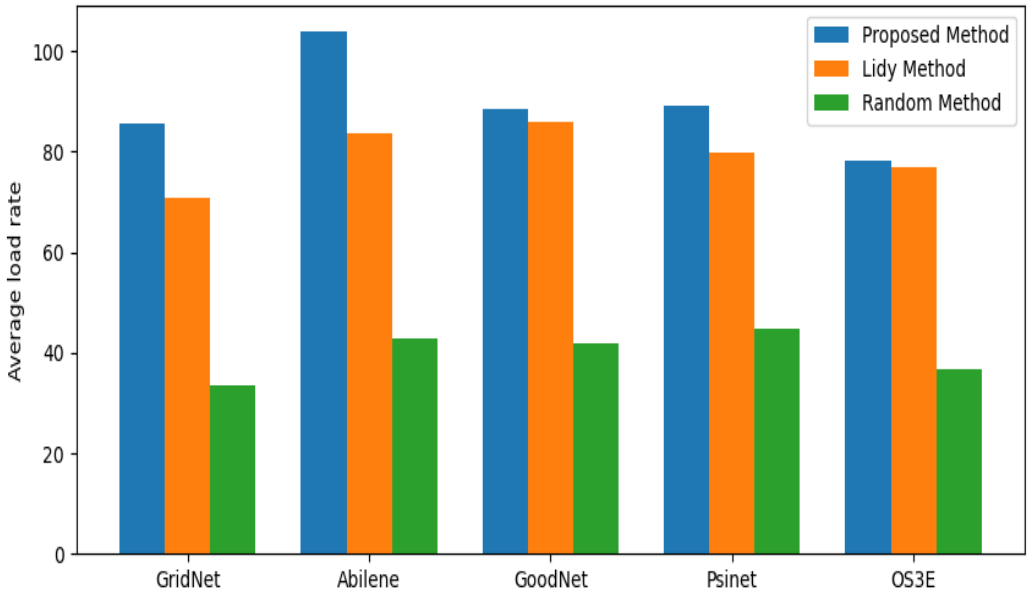


Figure 4.7: Comparison of Average Load rate in five topologies.

4.3 Topology-aware controller placement for SD-IoT: case study of the algerian network

In this section, we extend our study to a real-world inspired SD-IoT scenario by modeling the Algerian national network topology. This case study aims to demonstrate the applicability and effectiveness of

our controller placement strategy in a geographically distributed, population-aware network environment.

4.3.1 Problem statement and motivation

In this study, we propose the design of an SDN-based infrastructure that reflects Algeria's geographical characteristics, with a primary focus on minimizing latency one of the major performance challenges in SDN-enabled IoT environments. To date, no SD-IoT topology reflecting Algeria's geographic and demographic distribution has been proposed in the literature. In contrast, other regions, such as the United States, Germany, and China, benefit from publicly accessible network datasets, including Internet2 OS3E [132], DFN, and ChinaNet [131], which have facilitated extensive research on realistic, large-scale network performance evaluation.

Inspired by these existing resources, our objective is to design a novel SD-IoT topology for Algeria that considers both the spatial distribution of states and realistic traffic heterogeneity. Notably, previous studies assume either uniform node distributions or randomly generated switch traffic, which does not accurately capture the network dynamics of real-world environments. Moreover, it has been shown that a single controller is insufficient to cover a geographically large or highly populated region while maintaining acceptable latency levels, particularly in the presence of heterogeneous traffic demands.

To overcome these limitations, we propose a two-step modeling and optimization framework. First, we generate two distinct instances of the Algerian SD-IoT topology: one with homogeneous (unit-weight) nodes, and another weighted instance, where node weights reflect the actual population of each state. This approach introduces realistic traffic variability into the network model.

Subsequently, to ensure scalability and minimize latency, we apply a clustering-based controller placement strategy. Specifically, the k-means clustering algorithm is used to partition both the homogeneous and heterogeneous topologies into clusters. Within each cluster, the controller placement is optimized using both K-median and K-center algorithms, targeting average latency minimization and worst-case latency reduction, respectively. The main contributions of this study are summarized as follows:

- Development of the first SD-IoT network topology specifically modeled for Algeria based on the country's administrative division.
- Construction of a population-weighted version of the network to realistically represent heterogeneous traffic patterns across different states.

- Application of k-means clustering to both network instances to improve scalability and management.
- Within each cluster, deployment of K-median and K-center algorithms, along with their improved versions, to optimize controller placement considering both average and worst-case latency metrics.
- Comprehensive performance evaluation and comparison between the different placement approaches, demonstrating the benefits of integrating geographical and traffic-aware information into SDN controller deployment strategies.

This methodology not only contributes a new reference topology for Algeria but also highlights the significance of combining demographic data with network softwarization techniques to improve the performance and resilience of SD-IoT infrastructures.

4.3.2 Methodology

In this section, we present the methodology adopted to design a realistic SDN-based network topology for Algeria, with particular emphasis on latency-sensitive scenarios and heterogeneous traffic considerations. Our approach involves translating the geographical map of Algeria into a structured network graph, where each administrative state (wilaya) is modeled as a network node, and physical adjacencies between states determine the interconnections (edges). The resulting topology is further enhanced by introducing population-based weights for each node, capturing realistic traffic generation potential. This modeling is crucial for simulating practical SD-IoT deployments, as traffic generation is inherently linked to population distribution, particularly given that mobile devices represent a significant portion of IoT end devices.

The methodology consists of two main stages. First, a baseline topology is constructed with uniformly weighted nodes. Second, the topology is refined by incorporating population-driven heterogeneous node weights to better represent realistic traffic conditions. An overview of the entire process is illustrated in Figure. 4.8, which depicts both the physical map of Algeria and the corresponding generated network graph.

A) Baseline topology with uniform node weights

We begin by constructing a geographical network abstraction of Algeria based on the country's administrative divisions. The coordinates of each state's centroid (longitude and latitude) were

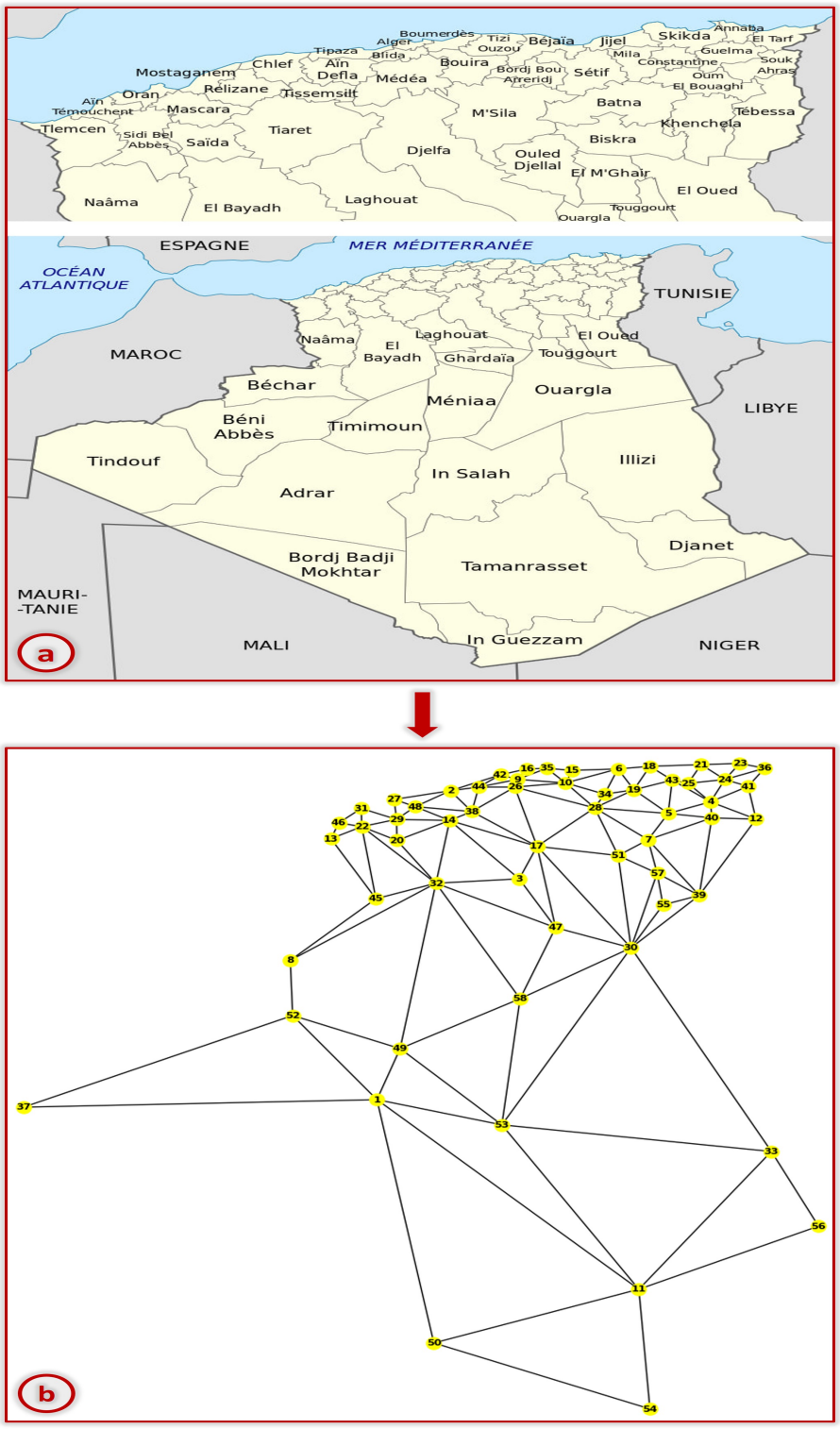


Figure 4.8: (a) Geographical map of Algeria. (b) Network topology where nodes represent Algerian states, and edges reflect physical borders between neighboring states.

collected from reliable sources such as Google Maps to ensure accurate spatial placement of nodes within the topology. Two nodes are connected by an edge if the corresponding states share a land-based geographical border. The generated graph structure, shown in Figure. 4.8(b), models Algeria as a

contiguous, undirected network suitable for SDN-based simulations.

In this initial representation, all nodes are considered to have equal, unitary weights. While this simplifies certain network evaluations, such an assumption fails to capture the significant heterogeneity in population density, urbanization, and traffic demand across different regions, motivating the next stage of our methodology.

B) Population-driven heterogeneous node weights

To develop a more realistic network model that reflects the diverse traffic generation capacities across Algeria, we incorporate node weights derived from each state's population. This enhancement acknowledges that states with higher populations, such as Algiers or Oran, contribute more substantially to network load than sparsely populated regions like Illizi or Djanet.

The weight W_{s_i} assigned to each node s_i (state i) is computed using a standard min-max normalization technique, defined as follows [133]:

$$W_{s_i} = y_{\min} + \left(\frac{P_{s_i} - P_{\min}}{P_{\max} - P_{\min}} \right) \times (y_{\max} - y_{\min}) \tag{4.7}$$

Where P_{s_i} denotes the population of state s_i, based on the latest available statistics published by Algerian government sources[1] . The terms $P_{\min}$ and $P_{\max}$ represent the minimum and maximum population values among all 58 Algerian states, respectively, while $y_{\min}$ and $y_{\max}$ define the normalization range for weights, fixed to $[1, 10]$ to reflect realistic traffic demand variability and maintain computational stability.

This normalization ensures that nodes representing highly populated regions are assigned higher weights, modeling both increased connectivity requirements and greater data generation potential in line with SD-IoT principles. Such a heterogeneous weight distribution is critical for accurately evaluating controller placement strategies, traffic engineering, and latency performance under realistic operating conditions.

The computed weights for all 58 Algerian states are summarized in Table 4.5. This realistic, population-weighted topology serves as the foundation for our SDN simulations, providing a more accurate evaluation environment for controller placement optimization, traffic engineering strategies, and latency management.

[1] https://interieur.gov.dz/Monographie/

Table 4.5: Algerian network topology: states and assigned weights.

ID	State (Wilaya)	Weight	ID	State (Wilaya)	Weight
1	Adrar	2.04	30	Ouargla	2.24
2	Chlef	4.70	31	Oran	6.80
3	Laghouat	2.86	32	El Bayadh	1.91
4	Oum El Bouaghi	3.12	33	Illizi	1.13
5	Batna	4.76	34	Bordj Bou Arreridj	2.98
6	Bejaia	3.68	35	Boumerdes	3.61
7	Biskra	3.11	36	El Tarf	2.29
8	Bechar	1.80	37	Tindouf	1.24
9	Blida	4.68	38	Tissemsilt	1.97
10	Bouira	3.29	39	El Oued	3.00
11	Tamanrasset	1.47	40	Khenchela	2.26
12	Tebessa	3.12	41	Souk Ahras	2.44
13	Tlemcen	4.00	42	Tipaza	2.91
14	Tiaret	3.89	43	Mila	3.70
15	Tizi Ouzou	4.27	44	Ain Defla	3.48
16	Algiers	10.00	45	Naama	1.76
17	Djelfa	5.12	46	Ain Temouchent	2.16
18	Jijel	3.14	47	Ghardaia	2.13
19	Setif	6.10	48	Relizane	3.61
20	Saida	2.09	49	Timimoun	1.30
21	Skikda	3.98	50	Bordj Badji Mokhtar	1.01
22	Sidi Bel Abbes	2.93	51	Ouled Djellal	1.45
23	Annaba	3.15	52	Beni Abbes	1.11
24	Guelma	2.46	53	In Salah	1.11
25	Constantine	4.52	54	In Guezzam	1.00
26	Medea	3.89	55	Touggourt	1.65
27	Mostaganem	3.42	56	Djanet	1.02
28	M'Sila	4.65	57	El M'Ghair	1.42
29	Mascara	3.60	58	El Meniaa	1.13

4.3.3 Results and discussion

Following the design of the SDN network topology, we employ the k-means clustering algorithm to divide the network into domains, each managed by a dedicated controller. We consider five clustering scenarios with 2, 3, 4, and 5 clusters. For each scenario, we evaluate the maximum and average latency using both the K-center and K-median approaches.

This section first provides an analysis of the infrastructure deployment cost for the proposed topology. Subsequently, we conduct a detailed performance evaluation of K-center and K-median algorithms under two settings: (i) unweighted topology where all nodes are considered homogeneous, and (ii) weighted topology where node heterogeneity, representing traffic load or population, is

integrated into the controller placement process. Both worst-case (maximum latency) and average-case latency metrics are analyzed.

The experimental framework is implemented using Python 3.12.0 to simulate k-means clustering, K-center, and K-median algorithms, along with their enhanced variants incorporating node weights.

A) Infrastructure cost assessment

The estimated cost for deploying the proposed SDN infrastructure considers factors such as the number of switches, total link length, number of controllers, and associated labor costs. The designed topology comprises 58 nodes representing switches, interconnected by 140 links, with a cumulative length of approximately 24,307 km, spanning a geographical area of 2,381,741 km^2. The highest node degree (defined as the number of neighboring nodes directly connected to a given node) is observed in El Bayadh and Ouargla (degree 9), while the lowest degree of 2 is recorded for Tindouf, Djanet, and In Guezzam. It is important to emphasize that the current assessment focuses on the cost of extending the existing infrastructure rather than constructing it entirely from scratch, leveraging already deployed resources to achieve the proposed topology.

B) Performance of K-center and K-median in unweighted node topology

In the first set of experiments, we consider an unweighted topology, where all nodes contribute equally to the network's traffic load. Figure 4.9 presents the latency performance for varying numbers of controllers (Nc) ranging from 2 to 5. Figure 4.9(a) illustrates the average latency obtained using the K-median algorithm, while Figure 4.9(b) shows the maximum latency resulting from the K-center approach. As expected, K-median optimizes average latency, whereas K-center focuses on minimizing the worst-case (maximum) latency.

The optimal controller placements in both cases are determined using Eqs 3.5 and 3.6, respectively. The results demonstrate a consistent decrease in both average and maximum latencies as the number of deployed controllers increases.

C) Weighted node topology: K-center and K-median with node heterogeneity

To incorporate node heterogeneity representing traffic volume, population, or switch load improved versions of the K-median and K-center algorithms are implemented. These variants account for node weights W_{s_i} in both average and maximum latency computations, as defined by Eqs 4.8 and 4.9:

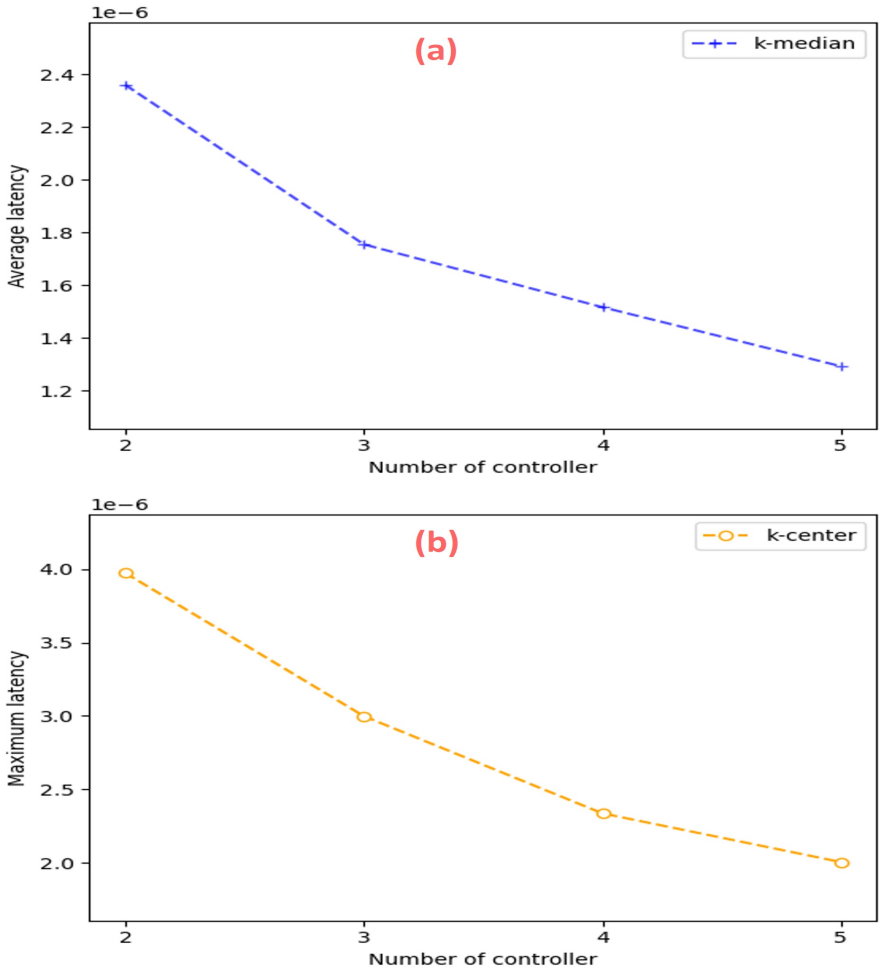


Figure 4.9: Impact of controller count on latency: K-center vs. K-median in an unweighted node topology.

$$SC_{AvgLat}(G) = \frac{1}{K}\sum_{j=1}^{K}\frac{1}{\phi_{c_j}}\sum_{i=1}^{N} D_{sc} \times x_{i,j} \times W_{s_i} \tag{4.8}$$

$$SC_{MaxLat}(G) = \frac{1}{K}\sum_{j=1}^{K}\max_{s_i \in S} D_{sc} \times x_{i,j} \times W_{s_i} \tag{4.9}$$

Figure 4.10 summarizes the comparative performance in the weighted topology scenario. Figure 4.10 (a) compares the average latency obtained with both the standard K-median and its improved, weight-aware version. Results consistently show that incorporating node weights significantly reduces average latency, with the performance gap narrowing as the number of controllers increases.

Similarly, Figure 4.10 (b) presents the maximum latency achieved by both the original and improved K-center algorithms. The improved version consistently outperforms the standard K-center across all controller configurations, highlighting the importance of accounting for node heterogeneity in

minimizing maximum latency.

In all cases, increasing the number of controllers results in a clear reduction of both average and maximum latencies.

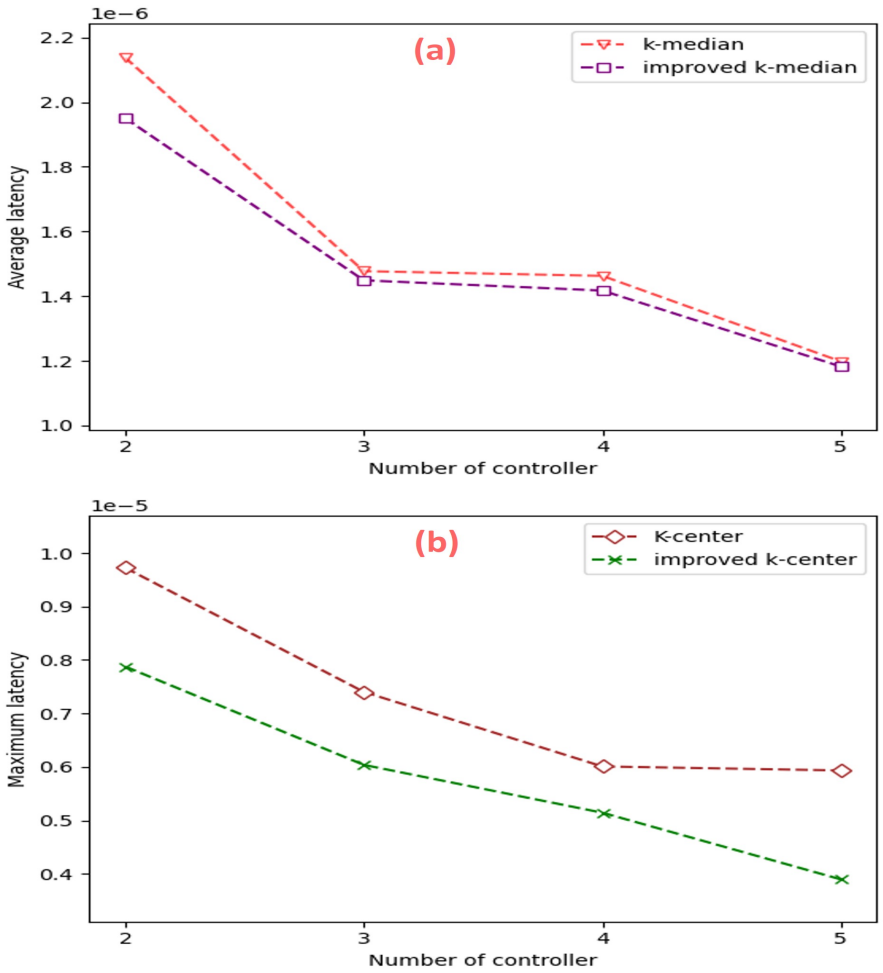


Figure 4.10: Impact of controller count on latency: K-center/K-median vs. improved versions in a weighted node topology.

D) Controller placement and distribution analysis

The impact of node weights on controller distribution is visually depicted in Figures 4.11 and 4.12, which illustrate the controller placements under different scenarios for both K-median and K-center approaches, along with their improved, weight-aware counterparts.

Figure 4.11 displays the controller placements after k-means clustering for scenarios with Nc=2 to 5. The left column (subfigures (a), (c), (e), (g)) corresponds to the standard K-median algorithm, while the right column (subfigures (b), (d), (f), (h)) shows placements derived from the improved K-median method. Controllers are visually distinguished by larger circles. The improved K-median

clearly adjusts placements based on node weights, ensuring optimal average latency performance.

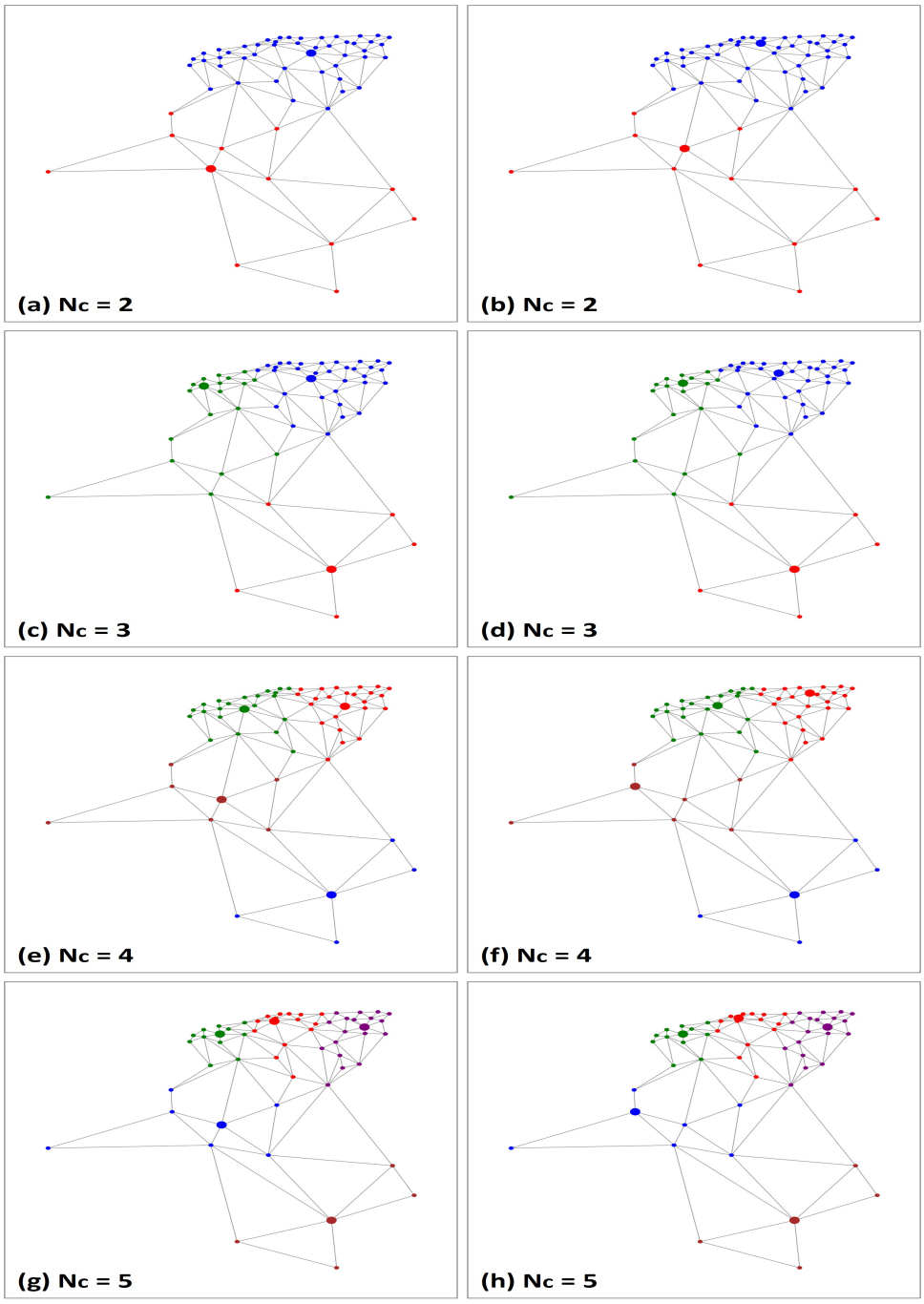


Figure 4.11: Controller placement (Nc= 2 to 5) in Algeria network using K-median and improved K-median algorithms.

Similarly, Figure 4.12 illustrates controller distributions for maximum latency minimization. Subfigures (a), (c), (e), (g) correspond to the original K-center algorithm, while (b), (d), (f), (h) depict results using the improved K-center. Again, accounting for node heterogeneity yields better-optimized controller locations, emphasizing the necessity of weight-based placement for worst-case latency optimization.

The experimental results demonstrate that both average and maximum latencies decrease with an increasing number of controllers. Moreover, incorporating node weights into K-center and K-median algorithms significantly enhances controller placement, leading to improved network performance in heterogeneous topologies. These findings highlight the importance of considering both node distribution and heterogeneity for effective controller placement in large-scale SDN deployments.

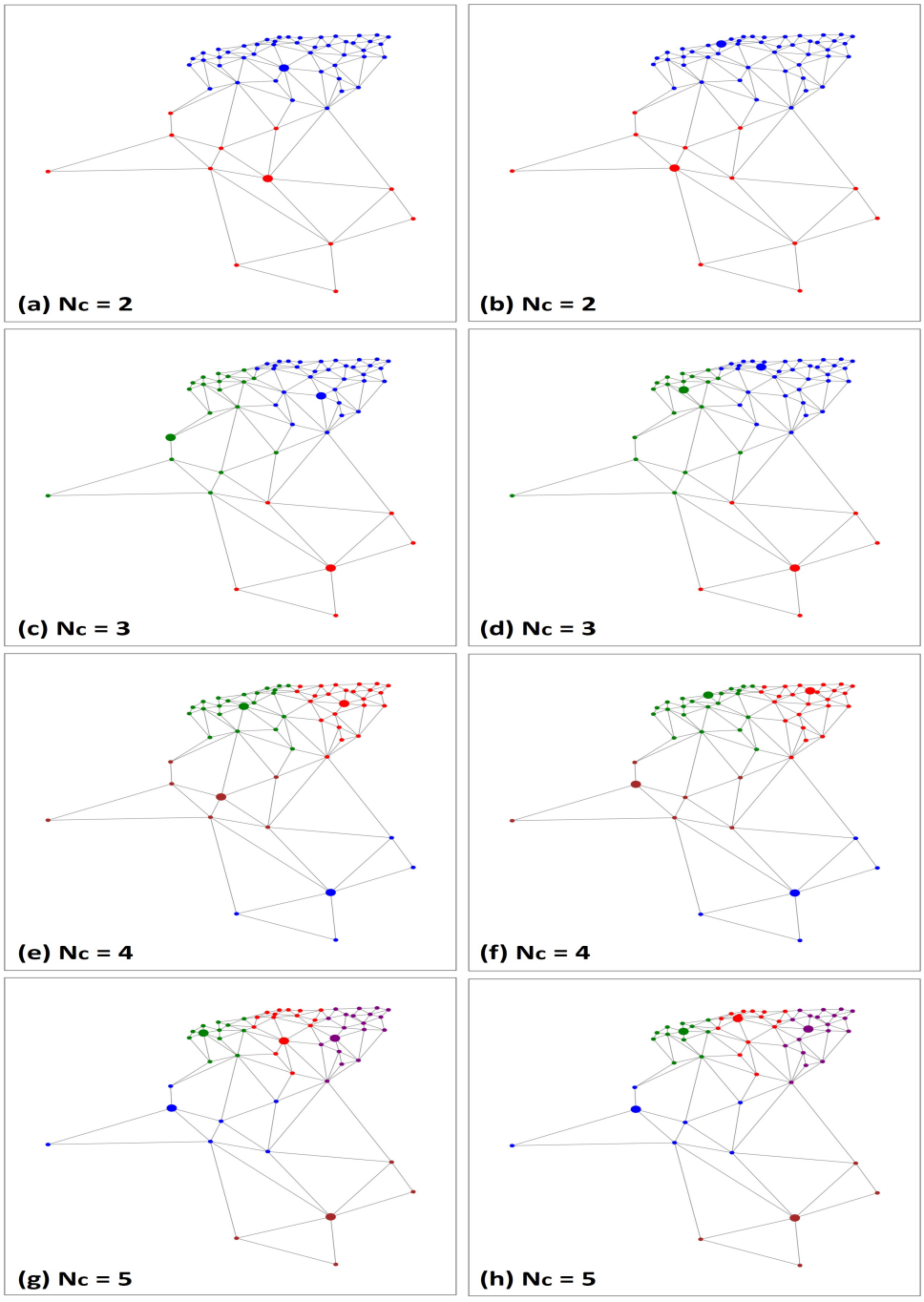


Figure 4.12: Controller placement (Nc= 2 to 5) in Algeria network using K-center and improved K-center algorithms.

4.4 Conclusion

This chapter presented a comprehensive investigation into exact controller placement in SD-IoT networks, spanning both free-space and realistic, topology-aware scenarios. Initially, we introduced a placement method that integrates traffic heterogeneity within an unconstrained search space to minimize latency and balance network load. Building upon these insights, we developed a geographically grounded SD-IoT topology for Algeria and evaluated controller placement using clustering and optimization techniques. The obtained results underline the importance of considering both network structure and node-specific traffic demands for effective controller deployment. Ultimately, this chapter bridges the gap between theoretical placement models and real-world network design, providing actionable strategies for enhancing SD-IoT performance in practical deployments.

5

Controller Placement using Weighted Betweenness Centrality

5.1 Introduction

This chapter provides a detailed performance evaluation of the proposed Weighted Betweenness Centrality-based Controller Placement (WBC-CPP) approach. Through extensive simulations, we assess the effectiveness of WBC-CPP under varying network conditions and candidate selection scenarios. The proposed method is assessed against state-of-the-art benchmarks to verify its performance in optimizing latency, flow rate, and computational efficiency.

5.2 Betweenness Centrality: concept and relevance for SD-IoT

Betweenness Centrality (BC) is a fundamental network science metric used to quantify the structural importance of nodes within a graph based on their participation in the shortest communication paths.

For any connected network, numerous shortest paths may exist between pairs of nodes, and BC evaluates how frequently a given node acts as an intermediary along these optimal paths. Formally, the BC_{s_i} of a switch s_i is expressed in SDN network as follows:

$$BC_{s_i} = \sum_{s_k, s_l \in (V \setminus s_i),\ s_k \neq s_l} \frac{\sigma_{s_k,s_l}(s_i)}{\sigma_{s_k,s_l}} \tag{5.1}$$

Where $\sigma_{s_k,s_l}(s_i)$ represents the number of shortest paths between switches s_k and s_l that pass through switch s_i, and σ_{s_k,s_l} denotes the total number of shortest paths between switches s_k and s_l.

The significance of BC lies in its ability to identify nodes that serve as critical intermediaries for information dissemination across the network. Nodes with high BC values inherently possess greater influence over the flow of data, making them essential for enhancing connectivity, improving robustness, and optimizing communication efficiency. Consequently, BC has been widely adopted in diverse domains such as transportation networks, social network analysis, biological systems, and communication infrastructures to reveal vulnerabilities, optimize resource placement, and improve system resilience.

In the context of SD-IoT networks, BC offers particular advantages for addressing the CPP, which involves determining the optimal locations for SDN controllers to ensure efficient network management. SD-IoT environments are characterized by large-scale, heterogeneous, and often dynamic topologies, where efficient controller placement directly influences key performance metrics, including latency, fault tolerance, and traffic load distribution.

Leveraging BC for controller placement in SD-IoT provides the following benefits:

- ***Reduced latency.*** Placing controllers at or near nodes with high BC minimizes the average and worst-case propagation delays, as these nodes lie along the most critical communication paths.

- ***Enhanced network resilience.*** High-BC nodes play a pivotal role in maintaining network connectivity. Their strategic utilization for controller placement increases fault tolerance and ensures efficient rerouting during failures or disruptions.

- ***Improved traffic management.*** By focusing on topologically significant nodes, BC-based placement supports better load balancing and prevents the formation of bottlenecks, which is vital in dense IoT environments.

- ***Scalability and adaptability.*** BC can dynamically reflect changes in network topology, allowing for adaptive controller reconfiguration to accommodate evolving IoT infrastructures.

Despite these advantages, classical BC formulations assume uniform traffic distribution across the network, a simplification that fails to capture the realities of heterogeneous IoT traffic patterns. To address this limitation, our work introduces a weighted BC approach that incorporates real-world traffic variability, providing a more effective controller placement strategy for complex SD-IoT deployments.

5.3 System assumptions and motivation

The Controller Deployment Problem is a central challenge in SD-IoT, with the objective of positioning controllers to optimize performance in terms of latency, reliability, and load distribution. Traditional solutions to CPP have largely relied on graph centrality metrics, particularly closeness centrality and eccentricity centrality, due to their ability to minimize average and worst-case communication distances respectively [134, 135].

However, these methods inherently assume homogeneous traffic distributions, where all switches generate equivalent data loads, an assumption that is fundamentally incompatible with real-world SD-IoT environments. IoT networks exhibit pronounced heterogeneity in traffic generation due to:

- ***Application-driven variability.*** IoT applications such as video surveillance, environmental monitoring, or industrial automation produce vastly different traffic profiles, ranging from high-volume continuous streams to sporadic, low-bandwidth transmissions.

- ***Device density fluctuations.*** Certain network segments may serve high-density clusters of devices, significantly increasing local traffic loads, while others remain sparsely populated.

- ***Temporal and event-driven dynamics.*** IoT networks often experience time-dependent traffic bursts or sudden surges due to event-triggered communications, as seen in smart cities or disaster response scenarios.

These traffic disparities compromise the effectiveness of uniform centrality-based CPP strategies, potentially leading to increased latency, congestion hotspots, and unbalanced controller workloads.

Betweenness Metric has emerged as a promising alternative for CPP by prioritizing nodes that function as critical transit points within the network [136]. However, classical BC does not account for the heterogeneous nature of IoT traffic, limiting its practical applicability.

To resolve this limitation, we develop a novel formulation that incorporates switch load variability into the controller placement framework. The approach introduces a weighted betweenness centrality

metric that extends the classical measure by integrating dynamic switch load information, thereby aligning controller placement decisions with both network topology and real-world traffic demands. Furthermore, a traffic-aware controller placement algorithm is developed, combining the proposed weighted centrality metric with a latency optimization process to enable adaptive controller positioning under heterogeneous and evolving network conditions. The proposed method is evaluated against three state-of-the-art approaches: the exact optimal solution, the Hierarchical Clustering with Betweenness Centrality (HC-BC) method [97], and the Louvain Community Detection with Betweenness Centrality (Louvain-BC) approach [137]. The performance comparison considers several key metrics, covering switch to controller and inter-controller latency, flow management efficiency under diverse traffic conditions, and computation time.

Our proposed WBC-CPP approach bridges the gap between theoretical CPP models and the operational complexities of SD-IoT networks. By unifying structural topology insights with traffic-aware considerations, our method enables more robust, scalable, and efficient controller placement, meeting the demands of modern IoT applications such as smart cities, industrial automation, and critical infrastructure monitoring.

5.4 Problem formulation

In this work, the SD-IoT network is modeled as an undirected graph $G = (V, E)$, where V represents the set of nodes and E is the set of links connecting them. The nodes are divided into two distinct categories: switches $S = \{s_1, s_2, \ldots, s_N\}$, which represent IoT gateways or forwarding devices, and controllers $C = \{c_1, c_2, \ldots, c_K\}$, which manage the control plane of the network. Each controller is colocated with a switch, ensuring that $K \leq |S|$.

To reflect realistic deployment conditions, the geographical positions of switches and controllers are taken into account, with the communication latency between them assumed to be proportional to their physical distance. This distance is computed using the Haversine formula as shown in Eq. (3.8).

The SD-IoT network exhibits heterogeneous traffic patterns, meaning switches handle varying amounts of data generated by connected IoT devices. Moreover, each controller has finite processing capabilities, imposing additional constraints on their placement to ensure load balancing.

The primary objective of the CPP is to minimize the following key performance metrics:

- ***Switch to controller average latency*** (SC_{AvgLat}). weighted by the traffic load of each switch, as defined in Eq. (4.8).

- ***Inter-controller average latency*** (CC_{AvgLat}). considering inter-controller communication delays, defined in Eq. (5.2).

$$CC_{AvgLat}(G) = \frac{1}{K(K-1)} \sum_{j=1}^{K} \sum_{l=1\ l\neq j}^{K} D_{cc} \times U_{c_l} \tag{5.2}$$

The CPP is subject to the following constraints to ensure feasibility and balanced operation:

$$\sum_{j=1}^{K} x_{i,j} = 1 \quad \forall\, i \in S \tag{5.3}$$

$$\sum_{i=1}^{N} y_{i,l} \leq 1 \quad \forall\, l \in C \tag{5.4}$$

$$\sum_{i=1}^{N} W_{s_i} \cdot x_{i,j} \leq U_{c_j} \quad \forall\, j \in C \tag{5.5}$$

$$K \leq |S| \tag{5.6}$$

The list of key notations is summarized in Table 5.1.

Table 5.1: Summary of notations used in the model.

Symbol	**Meaning**
$G = (V, E)$	Network graph with vertices V and edges E
S, C	Sets of switches and controllers
N, K	Numbers of switches and controllers
$Dist\_short$	Shortest path distance matrix
D_{sc}, D_{cc}	Distances between switch to controller and controller to controller
W_{s_i}	Load at switch s_i
U_{c_j}	Capacity of controller c_j
$x_{i,j}$, $y_{i,l}$	Binary variables for assignment and placement
γ_{C_j}	Switches managed by controller c_j
α	Objective weighting factor

This system model reflects both the geographic and traffic heterogeneity of real-world SD-IoT networks, providing the foundation for our proposed methodology.

5.5 Proposed methodology

The placement of controllers within SD-IoT networks presents significant challenges due to dynamic traffic patterns, heterogeneous node importance, and geographic constraints. Traditional controller placement techniques often rely on simplified assumptions, treating the network as static and ignoring the critical role of highly loaded or strategically positioned nodes. As a result, these methods frequently lead to increased latency, inefficient load distribution, and degraded resilience, particularly in large-scale IoT deployments.

To overcome these limitations, we propose the Weighted Betweenness Centrality-based Controller Placement Problem (WBC-CPP) methodology. This approach extends the classical BC metric by incorporating traffic heterogeneity, enabling more intelligent and adaptive controller placement decisions.

The methodology consists of four main phases:

A) Computing weighted betweenness centrality

Traditional BC identifies nodes that frequently appear on shortest paths, highlighting their topological importance. To adapt this for heterogeneous SD-IoT networks, we introduce the WBC metric:

$$WBC_{s_i} = \sum_{\substack{s_k, s_l \in V \setminus \{s_i\} \\ s_k \neq s_l}} \frac{\sigma_{s_k,s_l}(s_i)}{\sigma_{s_k,s_l}} \times W_{s_k} \times W_{s_l} \tag{5.7}$$

Where $\sigma_{s_k,s_l}(s_i)$ denotes the total number of shortest paths connecting nodes s_k and s_l that pass through node s_i, and σ_{s_k,s_l} represents the total number of shortest paths connecting s_k and s_l. In addition, W_{s_k} and W_{s_l} correspond to the traffic load weights associated with nodes s_k and s_l, respectively.

This formulation ensures that communication between heavily loaded nodes contributes more significantly to centrality scores, aligning controller placement with both structural importance and traffic intensity.

B) Candidate selection and scenario definition

After computing WBC for all nodes, they are ranked in descending order of importance. To manage computational complexity while exploring optimal placements, we define five candidate groups comprising the top 10%, 20%, 30%, 40%, and 50% of high-WBC nodes. This strategy ensures coverage

of both core and peripheral nodes while maintaining algorithmic scalability.

C) Placement evaluation using cost function

For each candidate group, all possible combinations of K controller placements are evaluated using a composite cost function:

$$\text{Cost} = \alpha \cdot SC_{\text{AvgLat}} + (1 - \alpha) \cdot CC_{\text{AvgLat}} \tag{5.8}$$

The parameter $\alpha \in [0, 1]$ governs the trade-off between minimizing switch to controller and inter-controller latency, offering deployment flexibility based on network priorities.

D) Optimal placement determination

The placement combination yielding the lowest overall cost across all scenarios is selected as the optimal solution. This ensures minimal network latency, balanced controller workloads, and enhanced fault tolerance.

Algorithms 1 and 2 outline the main steps of the proposed approach, while Figure. 5.1 illustrates its global workflow and major functional phases.

Algorithm 1 FITNESS($Dist\_short$, K_{pos})

Require:
$Dist\_short$: Shortest-path matrix,
K_{pos}: Positions of the selected K controllers.

Ensure:
$comb\_cost$: Fitness value of the evaluated combination.

1: **for** each switch $s \in S$ **do**
2: Assign s to its nearest controller based on $Dist\_short$, while accounting for the switch load and the number of switches currently connected to each controller.
3: **end for**
4: Compute the switch to controller average latency: $SC_{AvgLat} \leftarrow$ Eq. (4.8)
5: Compute the controller to controller average latency: $CC_{AvgLat} \leftarrow$ Eq. (5.2)
6: Evaluate the overall fitness value: $Cost_{Fitness} \leftarrow$ Eq. (5.8)
7: $comb\_cost \leftarrow Cost_{Fitness}$
8: **return** $comb\_cost$

This methodology offers a scalable, traffic-aware, and topology-sensitive approach to controller placement, making it highly suitable for dynamic, large-scale SD-IoT networks.

Algorithm 2 WBC-CPP: Controller Placement Based on Weighted Betweenness Centrality

Require:

$Dist\_short$: Shortest-path distance matrix,
K: Number of controllers to deploy,
W_{score}: Vector of node weights representing their relative importance.

Ensure:

$Cntr\_pos$: Optimal controller positions for five scenarios,
$Cost$: Corresponding cost values for each scenario.

Initialize all $W_{score}[v] \leftarrow 0$
for $p \leftarrow 1$ to 5 **do**
    $Cntr\_pos[p] \leftarrow \{\}$
    $Cost[p] \leftarrow \infty$
end for
for each node $v \in V$ **do**
    Compute the weighted betweenness centrality of v using Eq. 5.7
    Assign the result to $W_{score}[v]$
end for
Sort all nodes in ascending order according to W_{score}
for $p \leftarrow 1$ to 5 **do**
    $m \leftarrow 0.1 \times p \times |V|$
    Select the top m nodes: $TopNodes_{scores} \leftarrow W_{score}[1 : m]$
    Generate all possible combinations of K controllers:
    $CombList \leftarrow combinations(TopNodes_{scores}, K)$
    for each candidate set K_{pos} in CombList **do**
        $Cost_{Fitness} \leftarrow fitness(Dist\_short, K_{pos})$ [Algo. 1]
        if $Cost_{Fitness} < Cost[p]$ **then**
            $Cost[p] \leftarrow Cost_{Fitness}$
            $Cntr\_pos[p] \leftarrow K_{pos}$
        end if
    end for
end for
return $Cost, Cntr\_pos$

5.6 Results and discussion

This section presents a comprehensive evaluation of the proposed WBC-CPP model. The evaluation is conducted through extensive simulations designed to assess the model's effectiveness and efficiency in comparison with established benchmark methods.

The WBC-CPP model is examined under five scenarios, each defined by a different proportion of nodes retained as potential controller locations. Specifically, nodes are ranked according to their Weighted Betweenness Centrality (WBC) scores, and the top 10%, 20%, 30%, 40%, and 50% of nodes are considered for controller placement in each scenario.

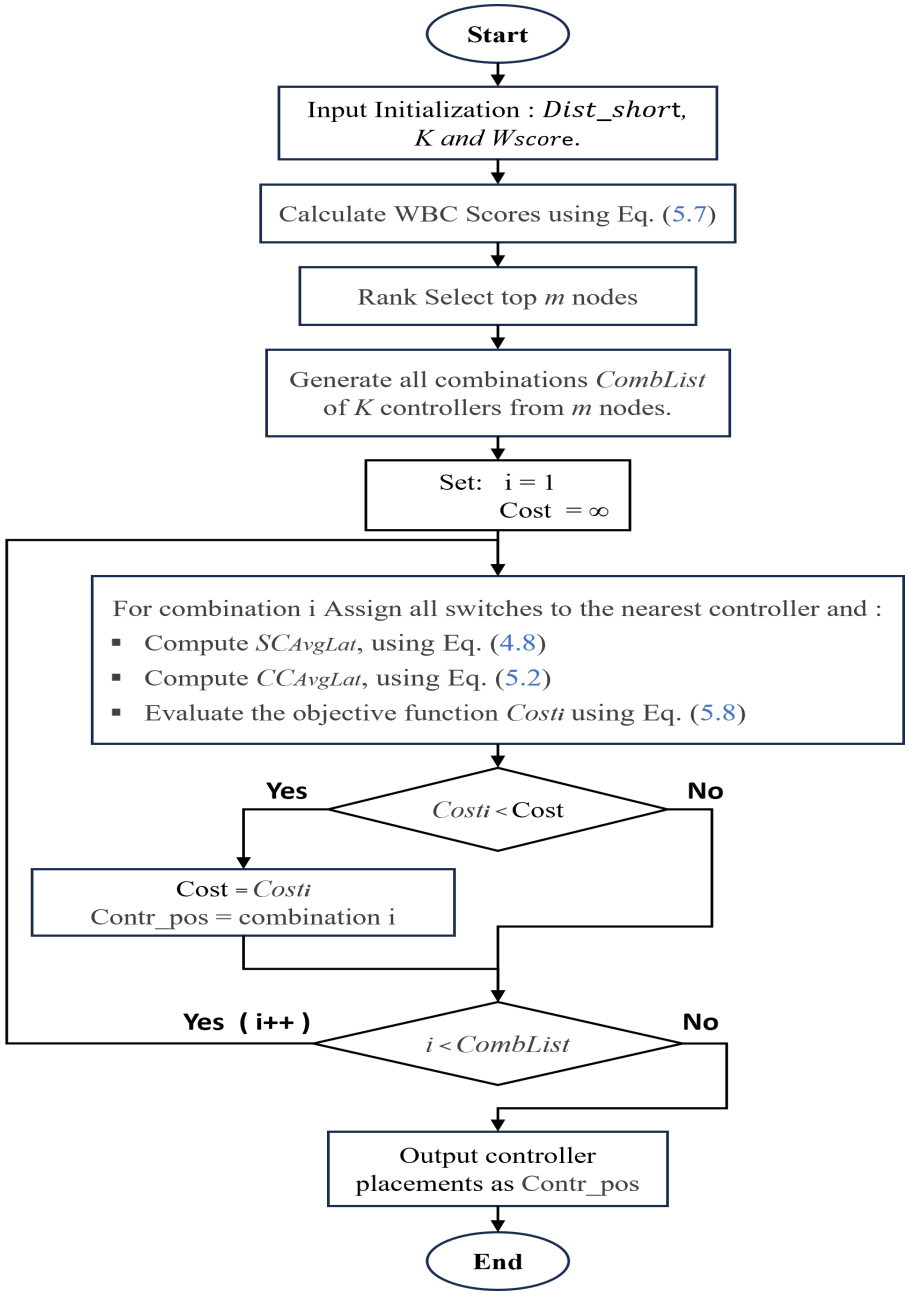


Figure 5.1: The WBC-CPP algorithm workflow.

For benchmarking purposes, WBC-CPP is compared against three notable approaches: (i) the Exact method, which performs an exhaustive search to identify the globally optimal controller placement, (ii) the Hierarchical Clustering with Betweenness Centrality (HC-BC) approach proposed in [97], and (iii) the Louvain-based Betweenness Centrality method (Louvain-BC) presented in [137]. These techniques are selected due to their relevance in state-of-the-art research on controller placement, particularly their scalability for large-scale network environments.

The performance analysis focuses on four key metrics: (i) switch to controller latency (SC latency), (ii) controller to controller latency (CC latency), (iii) network-wide average flow rate, and (iv)

computational execution time. In the Exact approach, all network nodes are treated as potential controller locations, resulting in an exhaustive combinatorial search. In contrast, WBC-CPP strategically reduces the candidate set to key nodes with high WBC values, effectively balancing computational efficiency and placement quality.

The network considered in this evaluation consists of 113 nodes. For the WBC-CPP approach in the scenario requiring the deployment of a single controller, the reduced candidate sets include the top 57 nodes (50%), 45 nodes (40%), 34 nodes (30%), 23 nodes (20%), and 11 nodes (10%) based on their WBC scores. Since only one controller is to be deployed, these values directly represent the number of possible placements. Table 5.2 presents the corresponding number of placement combinations for other scenarios involving multiple controllers.

Table 5.2: Number of possible controller placement combinations for each scenario as a function of k (controllers) and N (candidate nodes)

	Optimal	**WBC-CPP**				
k	$N = 113$	$50\% \times N = 57$	$40\% \times N = 45$	$30\% \times N = 34$	$20\% \times N = 23$	$10\% \times N = 11$
1	113	57	45	34	23	11
2	6 328	1 596	990	561	253	55
3	234 136	29 260	14 190	5 984	1 771	165
4	6 438 740	395 010	148 995	46 376	8 855	330
5	**140 364 532**	4 187 106	1 221 759	278 256	33 649	462
6	**2 526 561 576**	36 288 252	8 145 060	1 344 904	100 947	462
7	**38 620 298 376**	**264 385 836**	45 379 620	5 379 616	245 157	330

Figure 5.2 illustrates the spatial distribution of key nodes retained under each WBC-CPP scenario within the Deltacom topology.

5.6.1 Simulation setup

The simulations were executed on a workstation equipped with an Intel Core i7-1165G7 processor (2.80 GHz), 32 GB of DDR3 RAM (1600 MHz), running Microsoft Windows 11 Professional. All algorithms, including WBC-CPP and benchmark methods, were implemented in Python, utilizing the NetworkX library for network operations and NumPy for numerical computations.

The chosen network topology is the Deltacom network from the ITZ [131], consisting of 113 nodes and 183 edges, with a mean node degree of 3.24. This topology represents a large-scale, realistic network structure, reflecting the complexity typical of modern IoT and SD-IoT deployments. Its heterogeneous architecture and moderate density make it a suitable environment for evaluating the

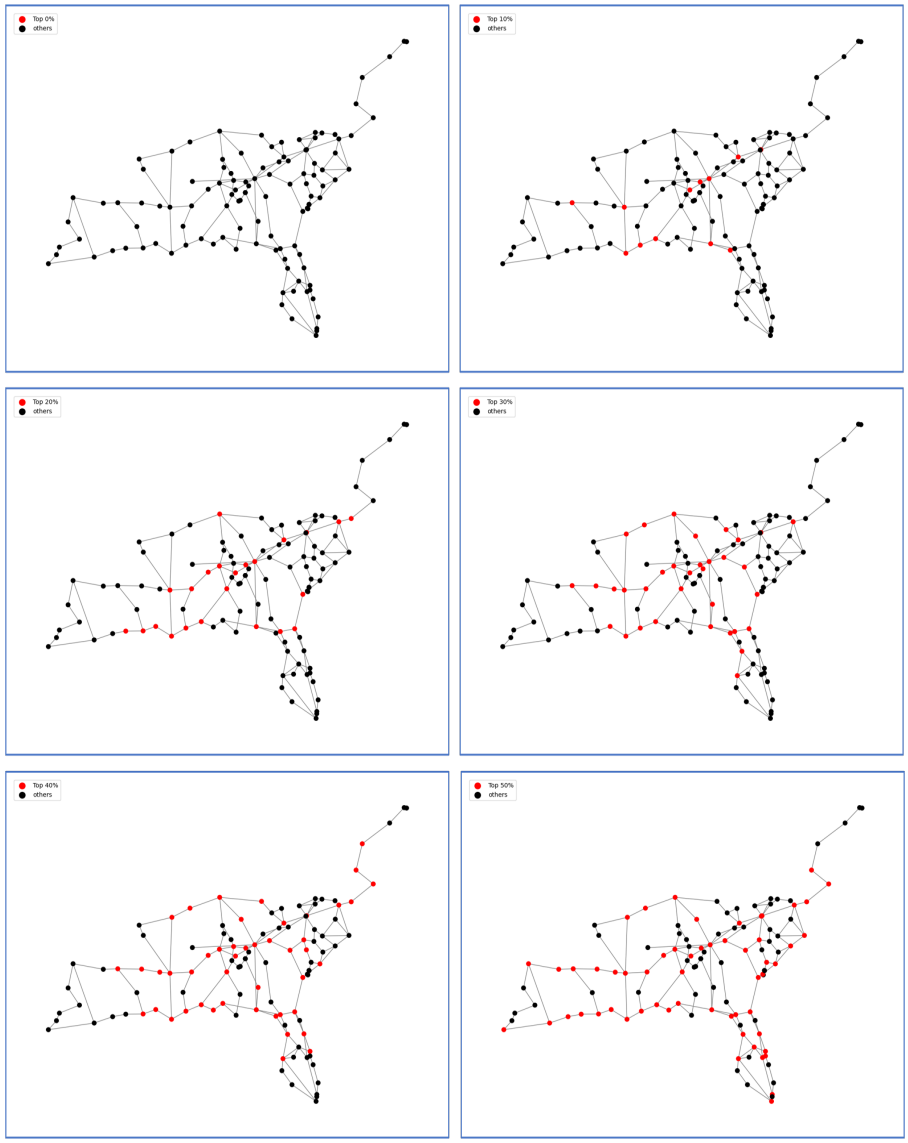


Figure 5.2: Distribution of top 10%–50% key nodes in Deltacom topology.

scalability and efficiency of CP strategies.

To emulate realistic traffic conditions, each switch is assigned a load value (in KFlows/s) representing the volume of flows it handles, randomly distributed within a defined range. The simulation parameters, determined through preliminary analysis, are summarized in Table 7.3.

A weighting factor α is introduced to balance the emphasis between switch to controller and inter-controller latency. In this study, α is set to 0.8, prioritizing switch to controller latency, justified by the predominance of such interactions in SD-IoT environments, where real-time data transmission between switches and their assigned controllers significantly influences overall performance.

Importantly, in cases where the execution time becomes prohibitively large, the corresponding results are omitted from the visual figures for clarity. This applies to the Optimal method for $k = 5, 6, 7$

Table 5.3: Simulation Parameters.

Parameter	**Value**
Topology	Deltacom
Total Nodes	113
Total Edges	183
Avg. Node Degree	3.24
Controllers (K)	1–7
Switch Load	1–300 KFlows/s
Controller Capacity	10,000 KFlows/s
Weight (α)	0.8

and to WBC-CPP(50%) for $k = 7$ in subsequent result figures.

5.6.2 Switch to controller average latency

This subsection presents a comparative analysis of the average latency between switches and their assigned controllers, which constitutes a critical performance indicator in SD-IoT networks, as it directly influences data forwarding efficiency and network responsiveness.

The results shown in Figure. 5.3 present the evolution of switch to controller latency across four strategies: optimal, Louvain-BC, HC-BC, and the proposed WBC-CPP. For the WBC-CPP method, results are provided for five reduction scenarios, representing the proportion of top-ranked nodes retained as potential controller locations (50%, 40%, 30%, 20%, and 10%).

As expected, the Optimal approach yields the lowest latency values across all tested configurations, representing the theoretical performance bound. WBC-CPP with 50% reduction exhibits near-optimal behavior, achieving comparable latency with a significantly reduced computational cost, which will be detailed in subsequent sections. Similarly, WBC-CPP at 40% and 30% reductions maintains low latency, offering a practical trade-off between solution quality and scalability.

However, as the candidate set is reduced to 20% and 10% of nodes, a gradual increase in latency is observed. This increase remains within acceptable limits and still outperforms both Louvain-BC and HC-BC approaches for all controller quantities $K = 1 \ldots 7$. The degradation at lower percentages highlights the importance of balancing reduction aggressiveness with the preservation of high-centrality nodes essential for minimizing control plane latency.

Notably, both Louvain-BC and HC-BC display considerably higher latency across all scenarios, with HC-BC consistently yielding the poorest performance. This confirms the limitations of conventional clustering-based placement strategies, which neglect node-level traffic heterogeneity and fail to

exploit topological centrality effectively.

These findings demonstrate that the WBC-CPP framework provides a scalable, adaptive solution capable of substantially minimizing switch to controller latency, even when applied to large-scale SD-IoT networks with constrained computational resources.

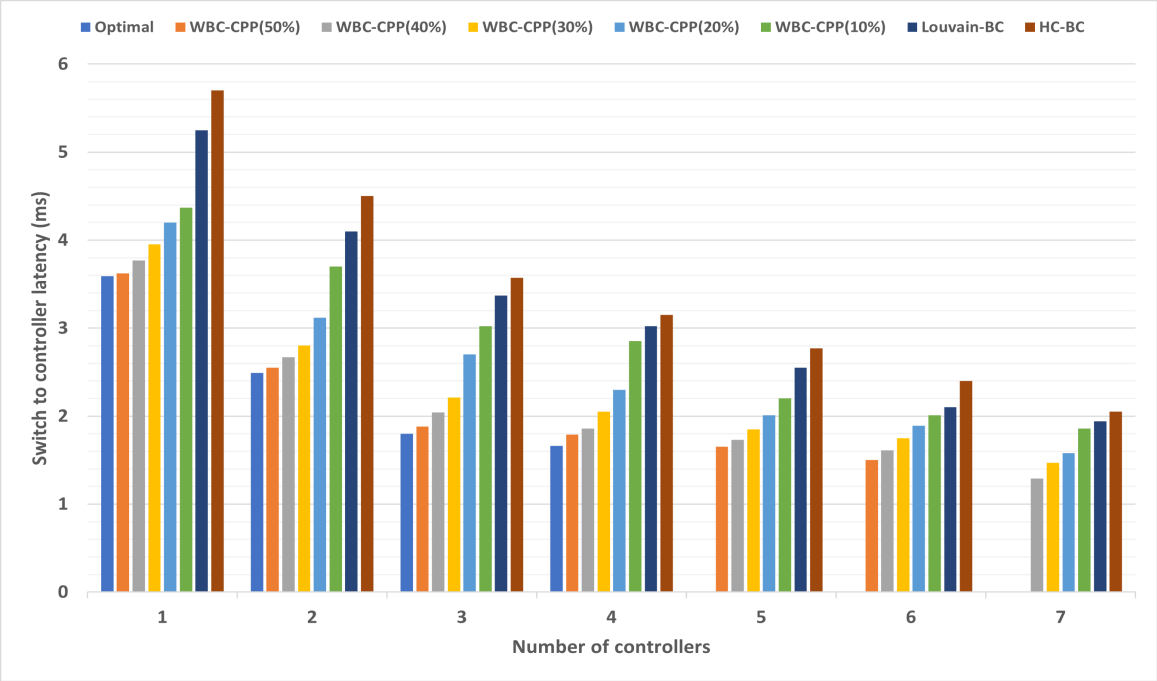


Figure 5.3: Average switch to controller latency for $K = [1 - 7]$.

5.6.3 Controller to controller average latency

In distributed SD-IoT architectures, controller to controller (C2C) latency represents a crucial performance metric, directly affecting control plane synchronization, state consistency, and fault recovery efficiency. Figure. 5.4 illustrates the average C2C latency for the four evaluated placement strategies, highlighting their respective impacts as the number of deployed controllers increases from $K = 2$ to $K = 7$.

As anticipated, the exact placement approach achieves the lowest C2C latency across all scenarios, establishing a theoretical performance baseline. The proposed WBC-CPP method exhibits a controllable trade-off between latency and computational overhead, with performance varying according to the reduction percentage of candidate nodes.

Specifically, WBC-CPP configured with 10% and 20% of top-ranked nodes consistently delivers near-optimal C2C latency, indicating that reducing the candidate space to highly influential nodes preserves critical network control paths. Conversely, as the reduction percentage increases to 30%,

40%, and 50%, a progressive rise in C2C latency is observed, attributable to the wider spatial distribution of selected controllers.

This behavior can be explained by examining Figure. 5.2, where it becomes evident that WBC-CPP(10%) concentrates controller placements within densely connected, high-centrality regions, minimizing inter-controller distances. In contrast, higher reduction scenarios result in more dispersed placements, naturally increasing average C2C latency.

The benchmark approaches, Louvain-BC and HC-BC, exhibit the highest latency values across all K configurations, with HC-BC performing particularly poorly. This is primarily due to their disregard for traffic heterogeneity and node weight distributions, which are essential for identifying strategic controller locations that minimize latency.

Overall, these results confirm the effectiveness of WBC-CPP in maintaining low C2C latency, especially when applied with targeted candidate reduction strategies. The method offers an adaptable balance between placement precision and computational scalability, outperforming conventional clustering-based techniques in dynamic, large-scale IoT environments.

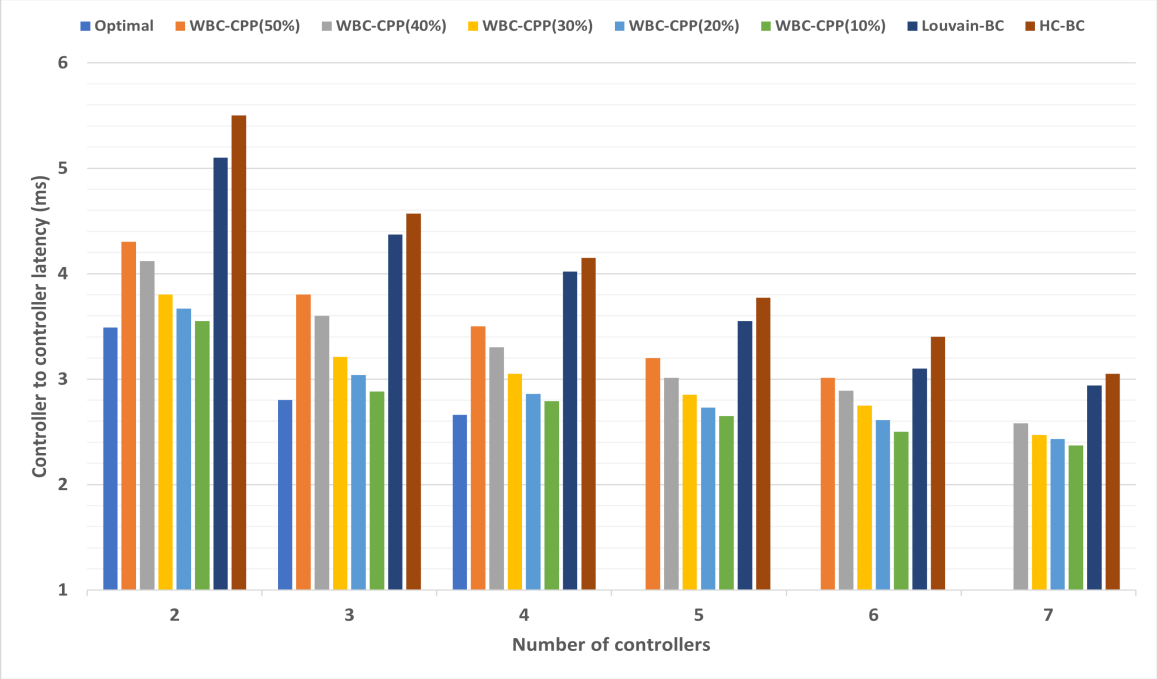


Figure 5.4: Average controller to controller latency for $K = [2 - 7]$.

5.6.4 Average flow rate

The average network flow rate, depicted in Figure. 5.5, serves as a critical performance indicator, reflecting the network's ability to handle traffic demands efficiently under different controller placement

strategies.

As expected, the Optimal approach consistently yields the highest flow rates across all controller configurations, providing an upper performance bound. This result is attributed to the exhaustive nature of the exact placement method, which identifies controller positions that maximize load balancing and minimize network bottlenecks. However, such optimality comes at the cost of prohibitive execution times, particularly in large-scale scenarios.

The proposed WBC-CPP method demonstrates a controlled compromise between flow rate and computational complexity, with its performance closely tied to the percentage of candidate nodes considered. Specifically, WBC-CPP(50%) achieves flow rates comparable to the Optimal method, particularly for smaller values of K. This alignment is due to the broader candidate space, which retains sufficient high-impact nodes to ensure effective load distribution.

As the reduction percentage decreases to 40% and 30%, a moderate decline in flow rate is observed, though the performance remains competitive. This trade-off reflects the progressive restriction of candidate nodes, which, while improving computational feasibility, marginally limits the solution space for optimal controller placement.

A significant performance degradation emerges in WBC-CPP(20%) and WBC-CPP(10%), where the drastic reduction in candidate nodes restricts placement options, particularly in scenarios with higher K values. The resulting imbalance in domain sizes and suboptimal controller-switch assignments lead to noticeable reductions in overall flow capacity.

In contrast, the benchmark strategies Louvain-BC and HC-BC consistently exhibit inferior flow rates across all controller configurations. Their inability to integrate node weights representative of heterogeneous traffic demands results in poor domain partitioning and inefficient traffic distribution, ultimately creating network bottlenecks. Notably, HC-BC consistently delivers the lowest flow rates, reaffirming the limitations of traditional clustering approaches that overlook traffic variability.

These observations underscore the significance of considering node-level traffic heterogeneity during controller placement. The weighted betweenness centrality-based approach employed in WBC-CPP effectively addresses this challenge, enabling improved traffic distribution and higher flow rates while preserving computational scalability. These performance trends are clearly reflected in Figure. 5.5.

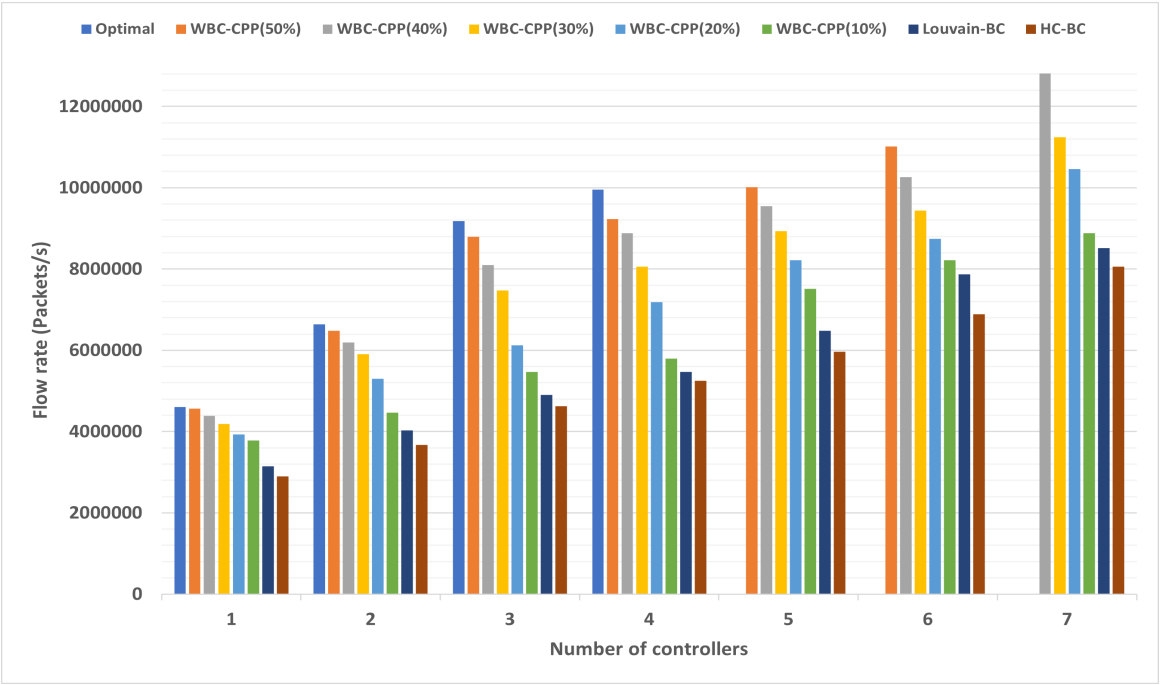


Figure 5.5: Average network flow rate on *packets/s*.

5.6.5 Execution time

The execution time of each algorithm, as shown in Figure. 5.6, provides valuable insight into the computational efficiency of various controller placement approaches.

The Optimal method demonstrates a steep increase in execution time as the number of controllers rises, primarily because the number of possible combinations expands exponentially. As K increases from 1 to 7, the combinatorial explosion becomes evident, resulting in excessively long execution times (refer to Table 7.4). Although this method theoretically ensures the best placement results, its computational cost renders it impractical for large-scale networks.

Conversely, the WBC-CPP algorithm maintains a balanced trade-off between efficiency and scalability. The execution time for WBC-CPP(50%) grows moderately with the number of controllers, yet remains significantly lower than that of the Optimal approach. The lower-percentage variants, WBC-CPP(40%) and WBC-CPP(30%), further reduce execution time, confirming the method's ability to optimize performance while keeping computational demands manageable. This makes WBC-CPP particularly well-suited for networks with limited processing capacity.

The most time-efficient cases are WBC-CPP(20%) and WBC-CPP(10%), which achieve minimal execution times for all tested values of K. These configurations are ideal for real-time and large-scale environments where reducing computational overhead is essential. Nonetheless, these versions present a minor compromise in optimization quality compared to higher-percentage configurations.

In contrast, the Louvain-BC and HC-BC methods maintain consistently short execution times regardless of K. Despite their computational efficiency, these algorithms perform less effectively in terms of latency and flow optimization, as they do not consider node load or the dynamic behavior of the network (as discussed in *Sections 5.6.2, 5.6.3, and 5.6.4*). Consequently, they fall short of the performance optimization achieved by the WBC-CPP framework.

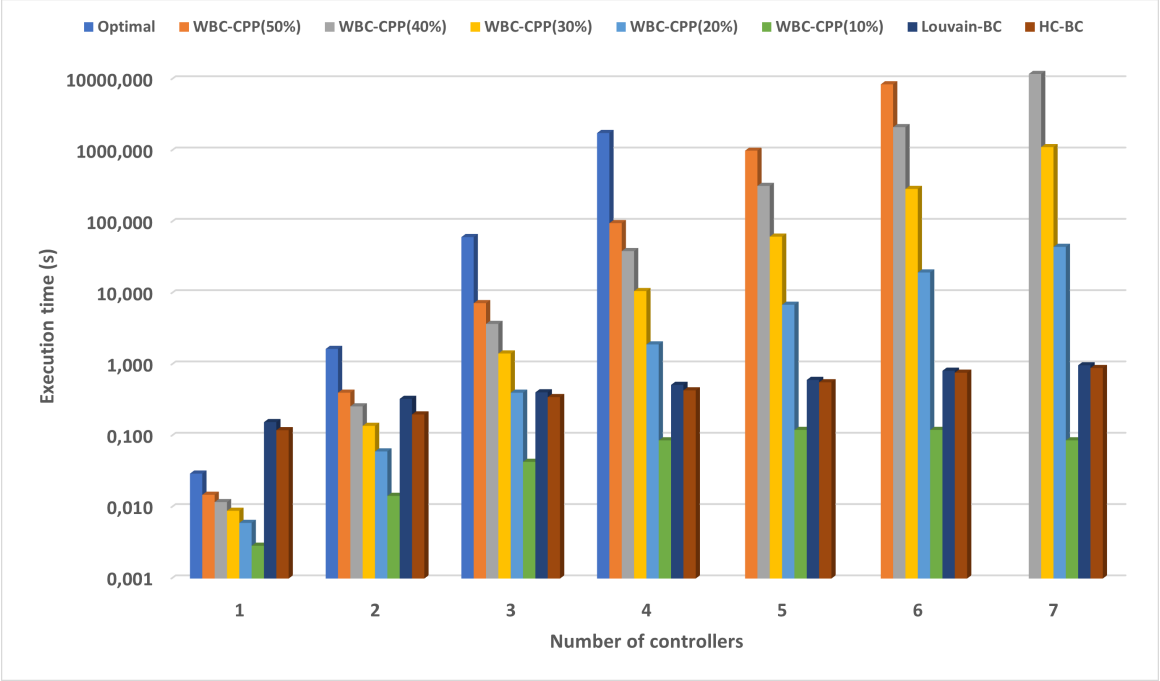


Figure 5.6: Execution time.

5.7 Conclusion

The results of this chapter demonstrate that WBC-CPP significantly outperforms existing methods by reducing latency and enhancing flow distribution, while preserving computational scalability. By incorporating node weights, the approach offers better adaptability to heterogeneous traffic patterns, making it suitable for large-scale, resource-constrained SD-IoT networks. However, it is important to note that the weighted nodes selected as controllers require enhanced reliability, which will be addressed in our next proposed method, along with a more in-depth investigation of computational efficiency.

6

Integrating GWO with WBC for scalable Controller Placement in dynamic SD-IoT

6.1 Introduction

In the previous chapter, a WBC-based controller placement strategy was proposed, but it suffered from high execution time. To overcome this limitation, this chapter integrates Grey Wolf Optimization (GWO) into the WBC framework, leveraging its efficiency in solving complex problems. In addition, we introduce three dynamic traffic scenarios to better capture the realistic behavior of IoT networks. The aim is to enhance controller placement by improving computational performance while addressing traffic variability in SD-IoT.

6.2 Grey wolf optimization in SD-IoT: A brief overview

Grey Wolf Optimization, a metaheuristic algorithm inspired by nature, was introduced by [138], and modeled after the leadership hierarchy and cooperative hunting strategies of grey wolves in nature. The algorithm simulates the encircling, hunting, and attacking behaviors of wolves to iteratively guide a population of candidate solutions toward optimal or near-optimal points in the search space. Its design emphasizes a balance between exploration and exploitation, allowing efficient convergence while avoiding premature stagnation. Due to its simplicity, flexibility, and strong optimization capability, GWO has been successfully applied across a wide range of complex optimization problems.

Within the scope of Software-Defined IoT networks, GWO offers a promising approach to solving the CPP. By efficiently searching the solution space, GWO helps identify optimal controller locations that improve network latency, scalability, and resilience. Unlike exhaustive search methods, which become computationally infeasible for large-scale topologies, GWO provides a powerful and resource-efficient alternative, making it well suited for dynamic and heterogeneous SD-IoT environments.

6.2.1 Fundamentals and mathematical formulation of Grey Wolf Optimization

The GWO emulates the leadership structure and collective hunting strategies of grey wolves. In this algorithm, each search agent represents a wolf, and the population is organized into four hierarchical levels according to fitness values: alpha (α), beta (β), delta (δ), and omega (ω). The alpha wolf symbolizes the best solution found so far and acts as the decision leader. The beta and delta wolves serve as advisors, helping guide the optimization process, while the omegas constitute the remainder of the population and follow the guidance of the top three wolves.

The algorithm seeks to iteratively refine the positions of wolves (candidate solutions) in the search space by exploiting information from the best-performing agents. Each position is assessed using an objective function that determines the quality of the corresponding solution. The main advantages of GWO are its ability to reduce computational cost by avoiding exhaustive evaluations of all possible configurations and its capacity to maintain an effective balance between exploration, which entails investigating new areas of the solution space, and exploitation, which concentrates on improving high-quality solutions. This balance helps prevent premature convergence and improves overall optimization performance.

The GWO position updating process consists of four principal steps:

1. **Hierarchy initialization.** The wolves are ranked based on the objective function, with roles

assigned as follows:

- α. best-performing wolf (leader),
- β. second-best wolf (assistant to α),
- δ. third-best wolf (supporting role),
- ω. remaining wolves (followers).

The top three wolves guide the optimization, and the omegas update their positions according to the influence of these leaders.

2. **Encircling and exploration mechanism.** The wolves simulate surrounding a prey (the optimal solution) by adjusting their positions based on the following relations:

$$\mathbf{D} = \left|\mathbf{C} \cdot \mathbf{X}_{prey} - \mathbf{X}\right| \tag{6.1}$$

$$\mathbf{X}(t+1) = \mathbf{X}_{prey} - \mathbf{A} \cdot \mathbf{D} \tag{6.2}$$

Here, $\mathbf{X}$ is the position vector of a wolf, while $\mathbf{X}_{prey}$ denotes the estimated optimal location. The vector $\mathbf{D}$ indicates the distance between a wolf and the prey, influenced by coefficient vector $\mathbf{C}$. The coefficient vectors $\mathbf{A}$ and $\mathbf{C}$ adjust the movement direction and intensity, ensuring a balance between exploration and exploitation.

$$\mathbf{A} = 2 \cdot a \cdot \mathbf{r}_1 - a \tag{6.3}$$

$$\mathbf{C} = 2 \cdot \mathbf{r}_2 \tag{6.4}$$

$$\mathbf{a} = 2 - \frac{2 \times i}{max_i tr} \tag{6.5}$$

The control parameter a decreases linearly from 2 to 0 as iterations progress, shifting the algorithm's focus from global exploration to local exploitation. Random vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ are uniformly distributed within $[0, 1]$.

3. **Coordinated hunting and position adjustment.** Wolves coordinate to hunt the prey by updating their positions according to the three best solutions, α, β, and δ, as follows:

$$\begin{aligned} \mathbf{D}\alpha &= |\mathbf{C}1 \cdot \mathbf{X}\alpha - \mathbf{X}| \\ \mathbf{D}\beta &= |\mathbf{C}2 \cdot \mathbf{X}\beta - \mathbf{X}| \\ \mathbf{D}_\delta &= |\mathbf{C}3 \cdot \mathbf{X}\delta - \mathbf{X}| \end{aligned} \tag{6.6}$$

$$\begin{aligned} \mathbf{X}1 &= \mathbf{X}\alpha - \mathbf{A}1 \cdot \mathbf{D}\alpha \\ \mathbf{X}2 &= \mathbf{X}\beta - \mathbf{A}2 \cdot \mathbf{D}\beta \\ \mathbf{X}3 &= \mathbf{X}\delta - \mathbf{A}3 \cdot \mathbf{D}\delta \end{aligned} \tag{6.7}$$

The updated position for each wolf is obtained as the average of the three leaders' influences:

$$\mathbf{X}(t+1) = \frac{\mathbf{X}_1 + \mathbf{X}_2 + \mathbf{X}_3}{3} \tag{6.8}$$

4. **Exploitation and attack phase.** As the search progresses and the parameter a declines, the algorithm transitions from global exploration to local exploitation. When $|\mathbf{A}| < 1$, wolves focus their movement around the top leaders to refine the current best solutions, simulating the final attack. Conversely, when $|\mathbf{A}| > 1$, the population disperses to explore new areas of the search space.

Together, these stages enable GWO to effectively balance exploration of the search space with exploitation of the best-found solutions. The hierarchical leadership, adaptive parameter adjustment, and cooperative hunting strategy collectively ensure that the algorithm avoids premature convergence and maintains a strong capacity to locate near-optimal or optimal solutions across diverse optimization problems.

6.2.2 Relevance of Grey Wolf Optimization to SD-IoT

Software-Defined IoT environments exhibit highly dynamic and heterogeneous characteristics, driven by massive device connectivity, diverse traffic patterns, and variable workloads. These conditions demand optimization techniques that are adaptive, scalable, and capable of handling complex interactions within the network. GWO, inspired by the hierarchical leadership and cooperative hunting strategy of grey wolves, provides a robust framework that naturally balances exploration and exploitation when navigating large and dynamic search spaces.

In the context of SD-IoT, GWO's decentralized decision-making and adaptive search dynamics make it particularly well-suited for addressing challenges such as resource allocation, load balancing, energy efficiency, and latency reduction. Its iterative update process allows solutions to evolve in response to environmental changes, ensuring resilience in networks where static strategies are often ineffective.

However, conventional implementations of GWO may not fully account for the unique traffic diversity and heterogeneity inherent in IoT ecosystems. This motivates the integration of domain-specific knowledge into GWO's formulation, enabling more reliable, scalable, and context-aware solutions tailored to the demands of SD-IoT.

6.3 Motivation

As demonstrated in the previous chapter, recognizing that each switch carries a different load significantly improves the accuracy of controller placement decisions when integrated with the standard betweenness centrality measure. This insight led to the formulation of the WBC-CPP approach, which identifies controller positions by considering both network topology and the heterogeneous loads of switches. While this strategy proved effective, it faces a major challenge: the execution time of the algorithm increases sharply with the number of switches and controllers, reducing its practicality for large-scale IoT environments.

To overcome this limitation, we sought a more efficient optimization mechanism capable of maintaining placement quality while scaling to larger networks. GWO emerged as a promising candidate due to its balance of exploration and exploitation in high-dimensional search spaces. By integrating GWO into the WBC-CPP framework, we obtained the GWO-WBC-CPP approach, which reduces computational burden while preserving the quality of placements, especially when restricting candidate nodes to the top-ranked 50% based on their centrality values.

Beyond computational efficiency, another motivation arises from the characteristics of IoT itself. Considering switch load alone although an important factor does not fully capture the dynamic behavior of IoT traffic. In practice, traffic intensity changes over time due to mobility, bursty communication patterns, and application-specific demands. This temporal variability creates additional challenges for maintaining balanced controller to switch assignments and stable end to end performance.

To better reflect these dynamics, we extend our evaluation by introducing three traffic model variants that emulate different temporal flow patterns. These models enable us to demonstrate the

robustness and adaptability of the proposed GWO-WBC-CPP approach under realistic, time-varying IoT conditions.

6.4 Problem statement

The following section is dedicated to presenting the problem statement within the SD-IoT framework. We first introduce the system assumptions that define the operational environment and constraints. Next, we provide the formal formulation of the controller placement problem. Finally, we describe the traffic modeling approach used to capture the dynamic nature of IoT flows.

6.4.1 System assumptions

The same system assumptions considered previously (see Section 5.3, page 88) are adopted here. However, beyond the heterogeneity of flows across switches, we additionally emphasize their dynamic nature, where traffic intensity varies over time. This temporal variability reflects a fundamental characteristic of IoT environments. In Subsection 6.4.3, we detail three traffic models that capture these dynamics.

6.4.2 Problem formulation

In this chapter, we adopt the same problem formulation as described previously (see Section 5.4, page 89). However, the propagation delay is calculated using the Euclidean distance 3.9 instead of the Haversine formula 3.8, to better suit the characteristics of the synthetically generated network considered. Additional details about this generated network are provided later in Section 6.6.

6.4.3 Traffic modeling

In IoT environments, traffic behavior is inherently dynamic due to factors such as node mobility, energy harvesting variations, limited resources, and the diversity of tasks performed by devices. As a result, the data flows directed toward network switches exhibit significant temporal instability and variation. Therefore, evaluation setups that rely solely on static or switch-specific heterogeneous traffic patterns fail to adequately represent the real operating conditions of IoT systems.

To accurately reflect this dynamic nature and thoroughly evaluate the resilience of the proposed controller placement strategy, three different dynamic traffic models are considered. Each of these

models aims to reproduce realistic IoT conditions by combining gradual traffic fluctuations with abrupt traffic surges (referred to as fork events). This design enables a deeper assessment of the adaptability and robustness of our approach under diverse and evolving network conditions.

A) Synchronized fork-timed dynamic traffic across switches

In this configuration, every switch has its own maximum traffic capacity. Under normal circumstances, traffic values fluctuate randomly between 40% and 70% of that maximum limit. To emulate large-scale network events, simultaneous traffic spikes referred to as forks are introduced, where all switches experience a surge at the same instant. This setup captures real-world IoT scenarios involving global triggers, such as system-wide firmware updates, emergency broadcasts, or synchronized sensing actions that lead to simultaneous network congestion.

As an example, consider three switches with maximum capacities of 300k, 50k, and 1k packets per second. Each switch's traffic varies randomly within the 40–70% range of its respective capacity, while synchronized forks are periodically introduced to simulate collective events. During the training period (200 seconds), ten synchronized forks occur, followed by four synchronized forks during the testing phase (50 seconds), as shown in Figure. 6.1.

Figure 6.1: Synchronized fork events per switch under dynamic traffic.

B) Independently fork-timed dynamic traffic across switches

This second model follows the same principle of dynamic traffic generation and fork introduction; however, the timing of the forks differs for each switch. This reflects decentralized IoT networks where nodes act autonomously and experience traffic peaks at different moments. Such cases occur frequently in practice for instance, when sensors respond to distinct local events, when energy harvesting rates differ per device, or when user actions trigger device-specific transmissions. Each switch thus undergoes ten fork events at random times, independent of other nodes, as depicted in Figure. 6.2.

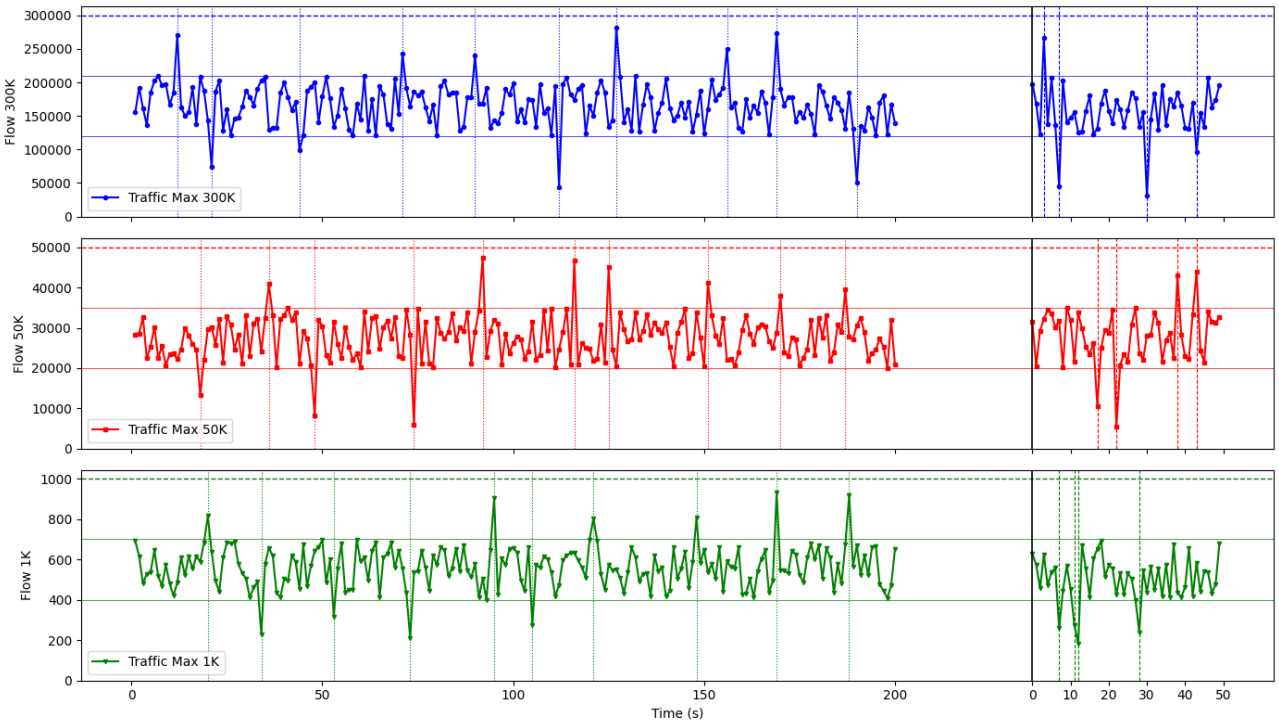


Figure 6.2: Independent fork events per switch under dynamic traffic.

C) Four-level stepwise dynamic traffic

The third pattern introduces discrete, stepwise variations in network load across four defined levels, each representing a different traffic range. Within a given level, traffic fluctuates randomly, but unlike the previous cases, no fork events are generated. This model is inspired by predictable, periodic variations in IoT environments such as increased activity during daytime and reduced load at night. It serves to evaluate the ability of the proposed method to adapt to regular, time-dependent traffic changes commonly observed in industrial and smart city networks. The generated traffic pattern for this scenario is shown in Figure. 6.3.

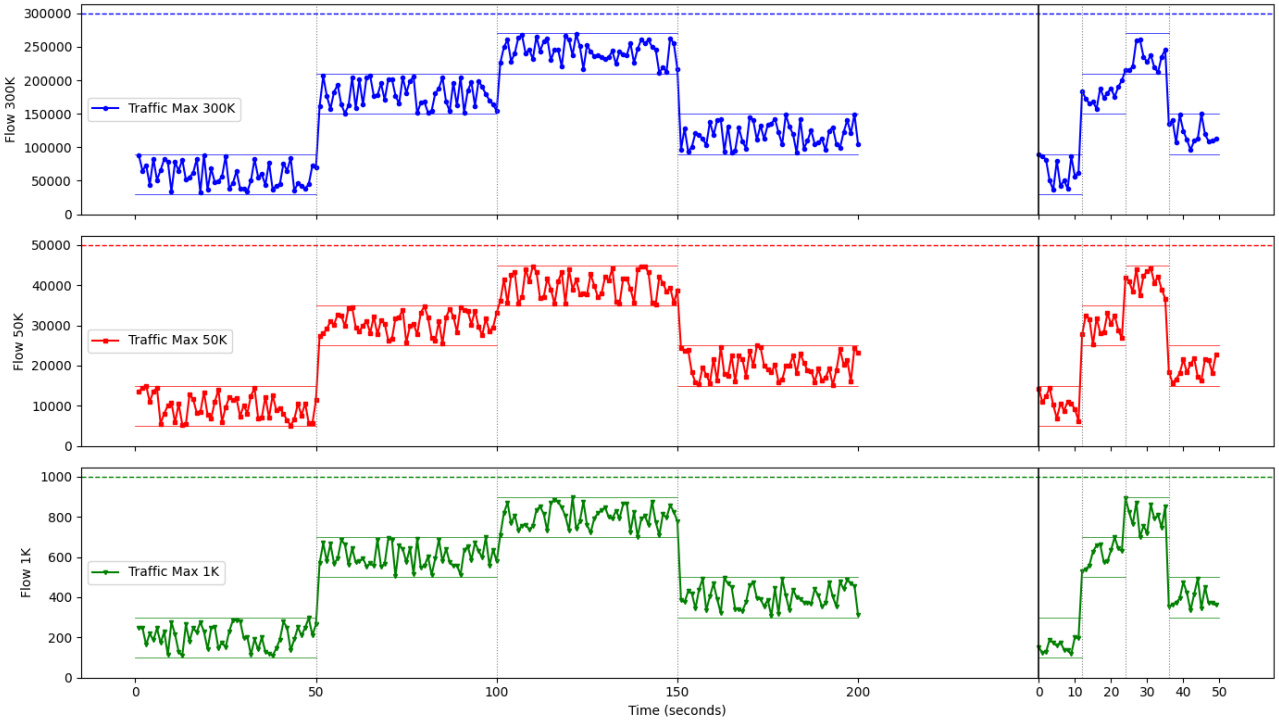


Figure 6.3: Four-level stepwise load under dynamic traffic without fork events.

6.5 Proposed GWO-WBC-CPP strategy

To address the issue of CP in SD-IoT networks, we present an extended strategy called GWO-WBC-CPP, which integrates the Grey Wolf Optimizer with Weighted Betweenness Centrality. The method aims to reduce the search space, prioritize structurally important nodes, and achieve an effective balance between global exploration and local exploitation.

Algorithm 3 receives as input the shortest-path matrix (`Dist_short`), the number of controllers K, a weight vector W_{score} indicating the maximum load for each switch, a training traffic vector (`train_traffic`) gathered over a 200-second interval, the wolf population size (`ppl`), and the maximum iteration count (`Max_itr`).

A) Candidate node selection

First, the WBC score of each node is computed to capture both topological significance and traffic load influence. Nodes are sorted in ascending order, and the top 50% with the highest scores are retained as potential controller candidates (lines 1-6 Algorithm 3). This filtering step substantially reduces the search space and directs the optimization toward nodes with higher structural relevance.

B) Initialization

The wolf population is initialized, where each individual represents a candidate placement of K controllers selected from the filtered node set. The average ($mean_L$) and standard deviation (std_L) of the training traffic are computed to simulate dynamic variations in traffic load (lines 7-8).

C) Dynamic fitness evaluation

For every candidate solution, the fitness function is computed three times using $W = mean_L$, $W = mean_L + std_L$, and $W = mean_L - std_L$ (lines 14-17). The final fitness value is the average of the three results, ensuring robustness under varying traffic conditions. The best three solutions are designated as the α, β, and δ wolves (lines 18-21).

D) Iterative optimization

At each iteration, the wolves update their location coordinates according to the GWO update rules, guided by the α, β, and δ wolves. Once positions are updated, all candidate solutions undergo re-evaluation using the multi-point fitness function, followed by re-ranking of the population (lines 23-30). This iterative process balances global search (exploring new placements) with local refinement (exploiting promising areas).

E) Stopping criterion and final solution

The optimization runs until the set maximum number of iterations is achieved (line 22). Finally, the position of the α wolf is returned as the optimal CP (`Cntr_pos`), with its corresponding cost value (`Cost`) (32).

The complete pseudo-code of the GWO-WBC-CPP algorithm is presented in Algorithm 3, while a flowchart illustrating its workflow is shown in Figure. 6.4. Together, they highlight the main stages of the proposed strategy, from candidate selection to iterative optimization and final placement decision.

In summary, the GWO-WBC-CPP approach provides a robust and efficient mechanism to address CPP by:

– Narrowing the search space via WBC-based filtering,

– Incorporating dynamic traffic-aware evaluation,

– Leveraging GWO's exploration–exploitation balance.

This enables the algorithm to efficiently navigate the large solution space of CPP in SD-IoT networks, avoiding local optima while maintaining reduced computational complexity.

6.6 Results and discussion

To assess the robustness and scalability of the proposed approach in modeling dynamic IoT network behaviors, we generated random topologies of different sizes comprising 100, 200, and 300 switches, representing small-, medium-, and large-scale networks, respectively. Each switch was assigned a random maximum load value to emulate heterogeneous traffic conditions. The nodes were uniformly distributed over a specified area, each defined by its coordinates (x, y) and corresponding traffic load. Euclidean distances between connected switches were computed using Eq. 3.9, followed by a preprocessing phase designed to eliminate excessively close nodes and enhance the structural balance of the topology.

For every generated topology, dynamic traffic was simulated using two input datasets: a training traffic file and a testing traffic file. The training dataset records traffic flows from IoT devices to their associated switches, and subsequently to the controller, at one-second intervals over a duration of 200 seconds. The traffic evolution follows one of the three predefined dynamic traffic models described earlier. The testing dataset adopts the same traffic behavior but over a shorter 50-second duration and includes ten distinct samples to verify the method's consistency. The training phase is used to determine optimal CP, and the obtained configuration is then applied to the test data to evaluate its performance under unseen traffic conditions.

To validate the efficiency of the proposed strategy, GWO-WBC-CPP(50%) which limits the controller placement search to the top 50% of nodes with the highest WBC is compared against several benchmark methods.This includes GWO-WBC-CPP (100%), which considers all nodes as candidate controller locations and serves as a near-optimal benchmark, and two baseline clustering-based algorithms, HC-BC and Louvain-BC. Unlike our proposed method, both HC-BC and Louvain-BC neglect the heterogeneous load distribution among nodes, which often results in less adaptive controller placements.

Extensive simulation trials were performed under different parameter configurations, and the finalized settings are summarized in Table 40. Experimental observations indicated that exceeding 100 iterations for GWO-WBC-CPP (100%) and 50 iterations for GWO-WBC-CPP (50%) produced negligible improvements in solution quality. Furthermore, varying the population size from 5 to

Algorithm 3 GWO-WBC-CPP

Require: $Dist\_short$: Distance matrix,
K : Number of controllers,
W_{score} : Weight nodes vector (Maximum load of each switch),
$train\_traffic$: Training traffic vector (the dynamic training traffic in 200s),
ppl : Population of wolfs,
Max_itr : Number of iterations

Ensure: $Cntr\_pos$: Optimal CP
$Cost$: Cost fitness function

$W_{scores} \leftarrow 0$
for $v \in V$ **do**
  $W_{score}[v] \leftarrow$ Compute WBC of v using Eq. 5.1
end for
Sort W_{score} in ascending order
$TopNodes_{scores} \leftarrow W\_scores[1 : 0.5 \times |V|]$
$mean_L \leftarrow$ average($train\_traffic$)
$std_L \leftarrow$ standard_deviation($train\_traffic$)
$ppl \leftarrow$ Initialize wolf population of size $ppls = |ppl|$
$Max\_itr \leftarrow$ Initialize maximum iteration
Initialize A, C, and a using Eqs. 6.3–6.5
$Cntr\_pos, Cost \leftarrow \{\}, \infty$
$A_{sco} \leftarrow \{\}$
for $i \leftarrow 1$ to $ppls$ **do**
  $A_{pos}[i] \leftarrow$ generate k random solutions from $TopNodes_{scores}$
  $A_{sco}[i] \leftarrow \frac{1}{3} \sum_{W \in \{mean_L, mean_L + std_L, mean_L - std_L\}} \text{fitness}(Dist\_short, A_{pos}[i])$ [Algo. 1]
end for
$Y \leftarrow \text{Sort}(A_{sco})$
$X_{\alpha p}, X_{\alpha s} \leftarrow Y[1]$
$X_{\beta p}, X_{\beta s} \leftarrow Y[2]$
$X_{\delta p}, X_{\delta s} \leftarrow Y[3]$
while $it \leq max\_itr$ **do**
  for $j \leftarrow 4$ to $ppls$ **do**
    update the positions of remaining individuals according to $X_{\alpha p}$, $X_{\beta p}$, and $X_{\delta p}$ using Eqs. (6.6)–(6.8)
    update A, C, and a using Eqs. 6.3–6.5
    $A_{sco}[j] \leftarrow \frac{1}{3} \sum_{W \in \{mean_L, mean_L + std_L, mean_L - std_L\}} \text{fitness}(Dist\_short, A_{pos}[i])$ [Algo. 1]
  end for
  $Y \leftarrow \text{sort}(A_{sco})$
  Update $X_{\alpha p}, X_{\alpha s}, X_{\beta p}, X_{\beta s}, X_{\delta p}, X_{\delta s}$
  $it \leftarrow it + 1$
end while
$Cntr\_pos, Cost \leftarrow Z_{\alpha p}, Z_{\alpha s}$
return $Cost, Cntr\_pos$

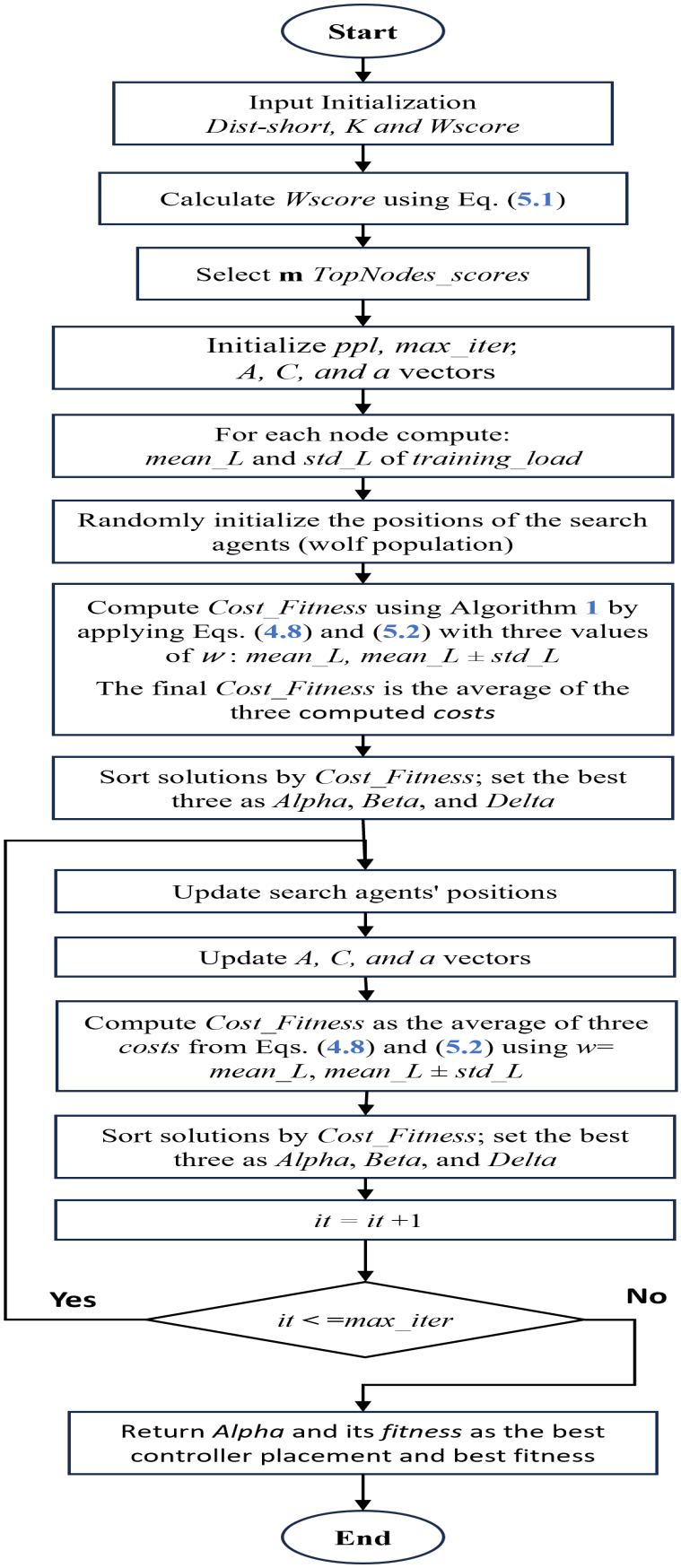


Figure 6.4: The GWO-WBC-CPP algorithm workflow.

50 showed that a population of 20 agents offered a stable trade-off between convergence speed and computational cost; thus, it was adopted for the final simulations. The detailed configuration of all simulation parameters is provided in Table 6.1.

Table 6.1: Simulation settings for GWO-WBC-CPP performance evaluation.

Parameter	**Specification**
Total Switches (N)	100, 200, and 300
Switch Distribution	Random, Uniform
Network Area (km^2)	0.5 M
Controllers (K)	1–7
Switch Load (W_{s_i})	1–300 KFlows/s
Controller Capacity (U_{c_i})	10,000 KFlows/s
Training Duration	200 s
Testing Duration	50 s × 10
Traffic Pattern	Synchronized, Independent Fork, Stepwise
Weight Coefficient (α)	0.8
ICFR Range	[15%, 25%]
Max Iterations (Max_itr)	50, 100
Population Size (|ppl|)	20
Parameter a	Decreasing from 2 to 0
Random Coefficients (r_1, r_2)	[0, 1]

6.6.1 Performance based on switch to controller average latency

The average switch to controller latency serves as a key indicator of control-plane responsiveness in SD-IoT networks, as it determines how quickly controllers can process and respond to switch requests. Figure 6.5 illustrates the results obtained under synchronized traffic conditions. The proposed GWO-WBC-CPP(50%) approach consistently achieves lower latency values than both HC-BC and Louvain-BC. As the number of controllers increases from 1 to 7, a noticeable reduction in latency is observed with our method, demonstrating the effectiveness of its controller distribution strategy. Conversely, HC-BC and Louvain-BC exhibit limited improvement with additional controllers, primarily due to their rigid clustering or modular placement mechanisms that tend to produce less optimal configurations.

A similar pattern appears under independent traffic conditions, as shown in Figure 6.6. However, the performance gap between the approaches becomes even more pronounced. The proposed algorithm dynamically adapts to fluctuating and distributed traffic loads, maintaining significantly lower average latency. Louvain-BC, in particular, performs poorly in this context, since its community-based

partitioning process is not inherently designed to minimize latency, leading to inefficient controller-to-switch mappings under diverse flow conditions.

Figure 6.7 presents the results for the phased traffic model, where traffic intensity varies stepwise to emulate realistic and uneven load transitions. Once again, GWO-WBC-CPP(50%) demonstrates strong robustness, maintaining low latency levels even during high-demand intervals. Although GWO-WBC-CPP(100%) occasionally achieves marginally better performance, the improvement is negligible relative to its higher computational complexity. The disparity between our method and the baseline approaches widens as network dynamics increase, underscoring the superior adaptability and efficiency of the proposed GWO-WBC-CPP framework across multiple traffic scenarios.

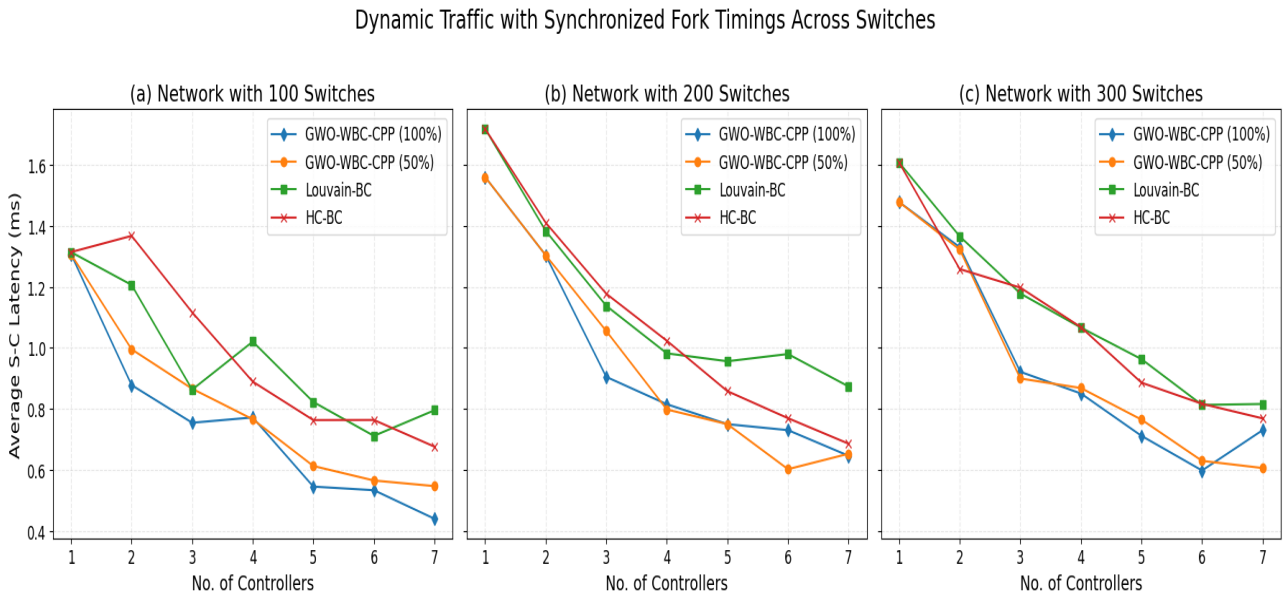


Figure 6.5: Switch to controller latency under synchronized fork timing.

6.6.2 Performance based on controller to controller average latency

Inter-controller latency represents the time required for controllers to exchange and synchronize state information. This metric is particularly crucial in distributed SDN architectures, where timely coordination ensures consistent network behavior. In Figure. 6.8, which corresponds to synchronized fork traffic, our proposed GWO-WBC-CPP (50%) consistently achieves lower CC_{AvgLat} compared to the other methods. This performance stems from its strategic selection of candidate nodes that are not only well-positioned for switch connectivity but also topologically advantageous for inter-controller communication. In contrast, HC-BC and Louvain-BC often deploy controllers in distant regions of the topology, leading to greater synchronization delays.

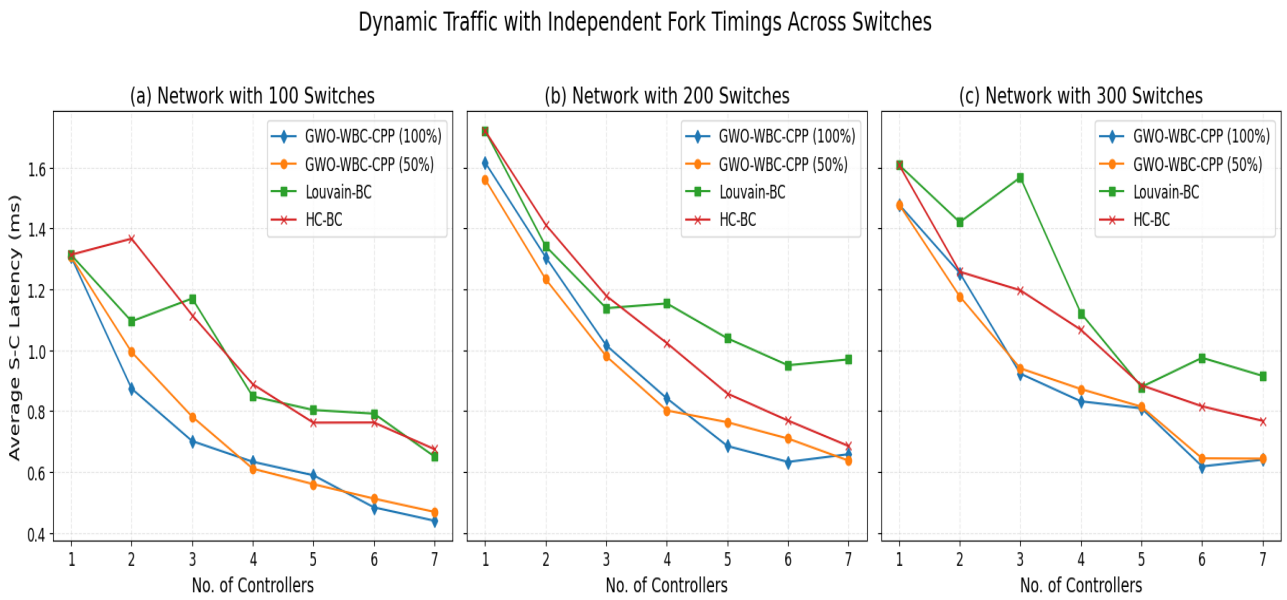


Figure 6.6: Switch to controller latency under independent fork timing.

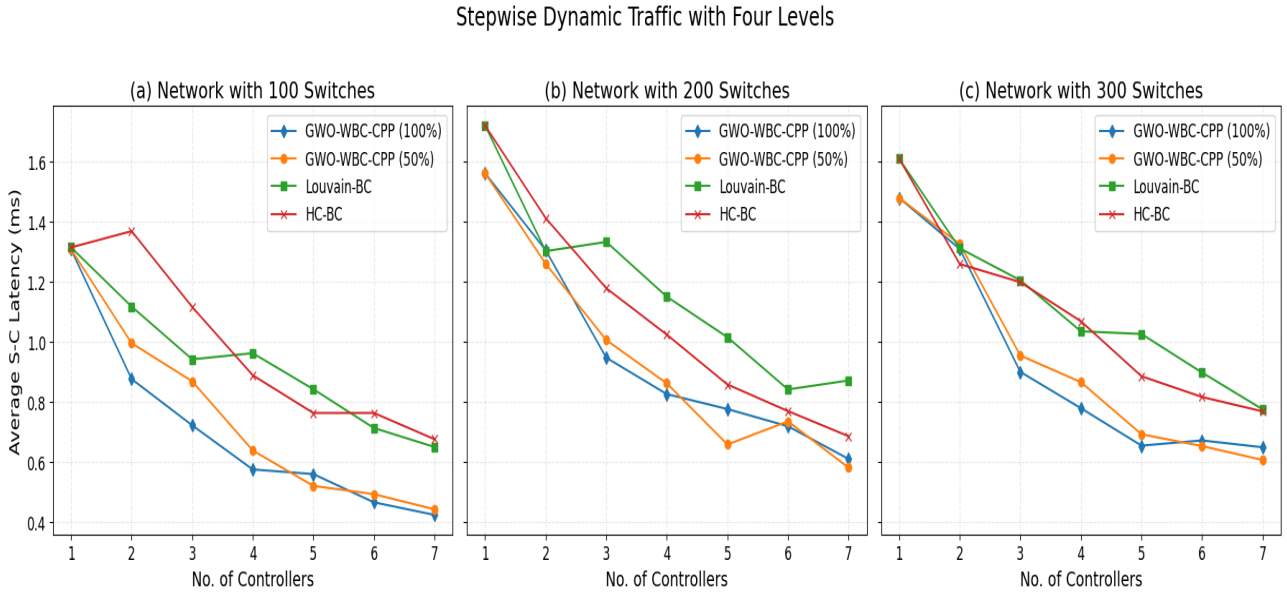


Figure 6.7: Switch to controller latency under four level stepwise load.

Under independent traffic, shown in Figure. 6.9, the differences become even more pronounced. The irregularity of flow patterns demands a placement strategy that simultaneously minimizes switch–controller latency and maintains tight inter-controller proximity. Our hybrid approach combining traffic awareness, centrality measures, and an efficient search-space reduction mechanism achieves this balance effectively, resulting in significantly reduced CC_{AvgLat}. Conversely, Louvain-BC's reliance on community-based partitions can unintentionally separate controllers across different regions, increasing communication latency.

Finally, the phased traffic model illustrated in Figure. 6.10 introduces staggered load variations that challenge both SC_{AvgLat} and CC_{AvgLat}. Even under these dynamic conditions, GWO-WBC-CPP (50%) maintains stable and low CC_{AvgLat} values, demonstrating resilience to temporal traffic fluctuations. Meanwhile, HC-BC and Louvain-BC exhibit sharp latency increases as traffic intensity shifts, underscoring their limited adaptability. Although the full candidate version (GWO-WBC-CPP (100%)) occasionally records slightly higher CC_{AvgLat}, this can be attributed to its broader search space, which does not emphasize controller proximity as strongly.

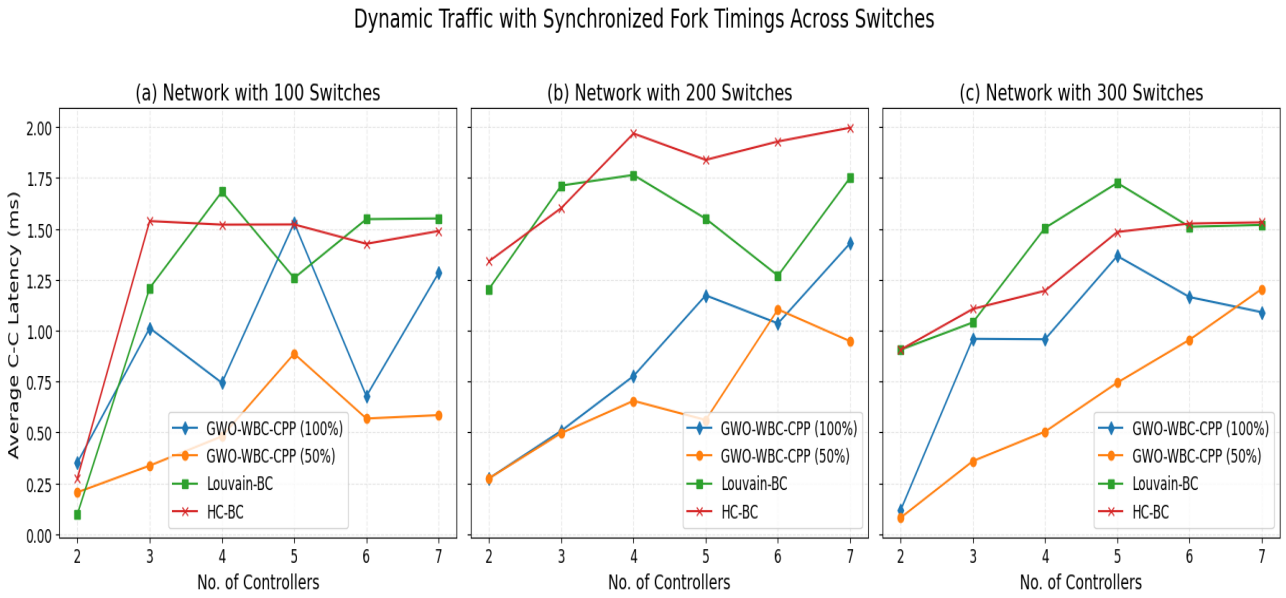


Figure 6.8: Controller to controller latency under synchronized fork timing.

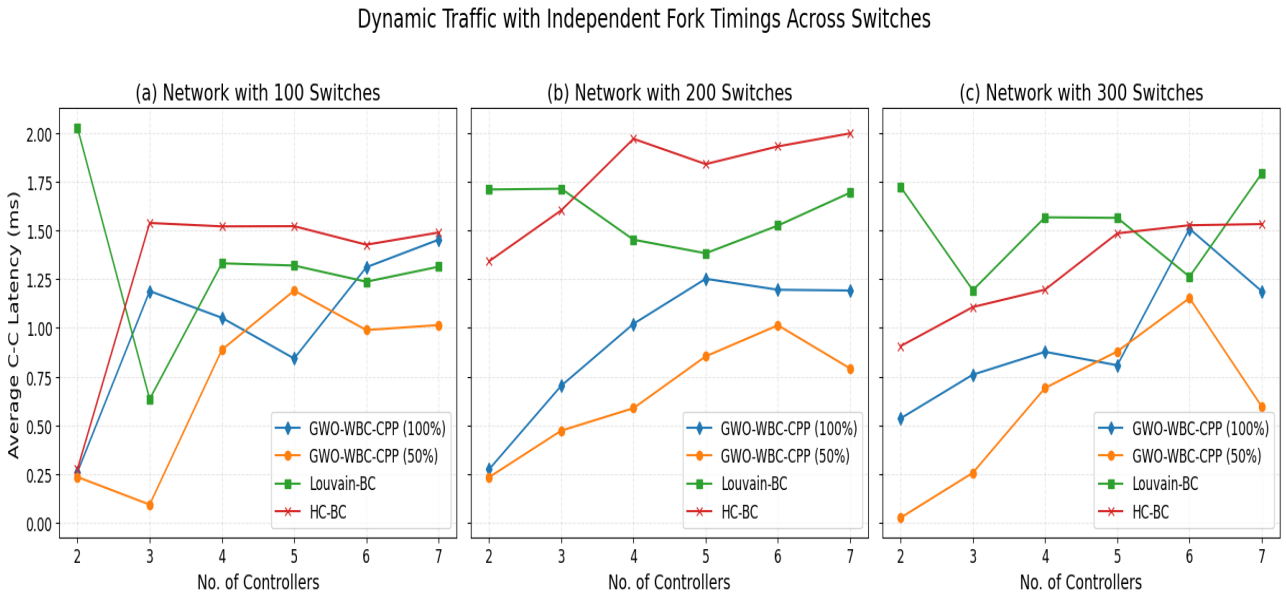


Figure 6.9: Controller to controller latency under independent fork timing.

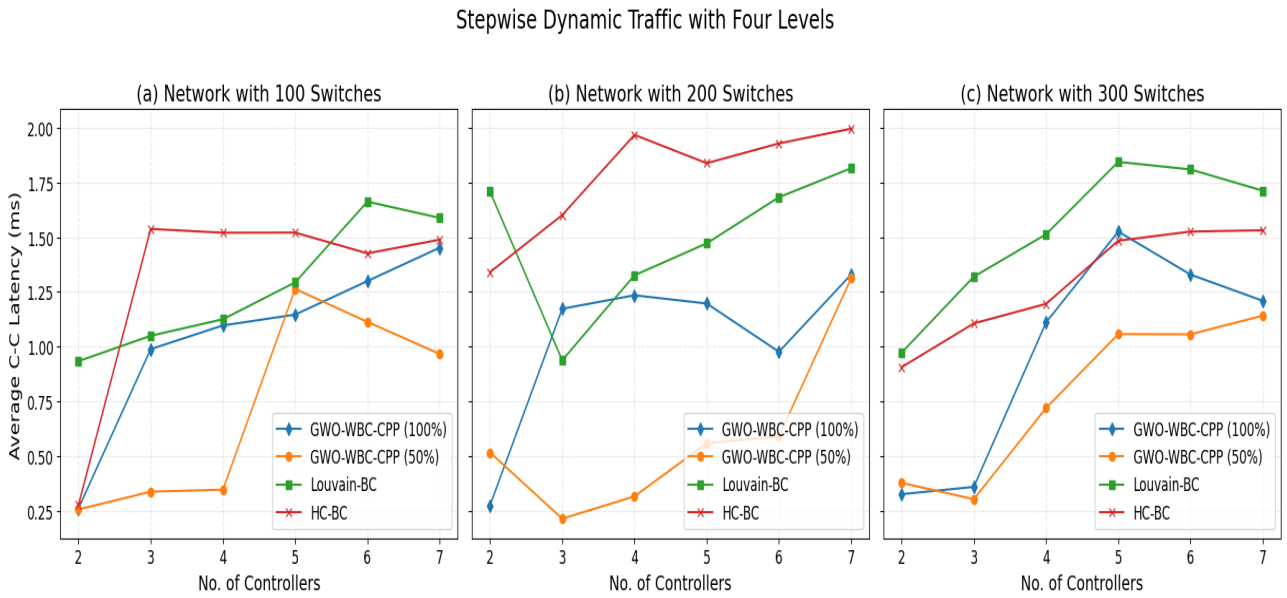


Figure 6.10: Controller to controller latency under four level stepwise load.

6.6.3 Performance based on average flow rate

Switch to controller flow rate quantifies the total data successfully transmitted between switches and their assigned controllers. It serves as a key indicator of how effectively a CP strategy manages network load and supports scalability. As shown in Figure. 6.11 for synchronized traffic, GWO-WBC-CPP (50%) consistently achieves a higher flow rate than HC-BC and Louvain-BC, particularly as the number of controllers increases. This improvement results from the optimized spatial distribution of controllers, which balances flow requests, mitigates bottlenecks, and prevents central overloads.

For independent traffic, illustrated in Figure. 6.12, our approach continues to outperform the competing methods. The inherently random and distributed nature of this traffic model emphasizes the need for adaptive and evenly distributed CP. HC-BC, which relies solely on centrality metrics without accounting for dynamic traffic variations, struggles under uneven load conditions causing congestion in high-demand regions and underutilization elsewhere. Similarly, Louvain-BC, while capable of forming communities, fails to explicitly optimize for flow rate and often assigns controllers to low-demand areas, limiting network throughput.

In the phased traffic scenario, depicted in Figure. 6.13, the advantages of our method become even more pronounced. At all four traffic intensity levels, GWO-WBC-CPP (50%) consistently achieves a high and stable flow rate, demonstrating resilience to fluctuating loads. Its adaptive placement mechanism effectively redistributes control responsibilities in response to changing flow volumes. Although GWO-WBC-CPP (100%) occasionally yields slightly higher flow rates, the performance

gain is marginal compared to its doubled computational cost. Overall, the proposed approach strikes an optimal balance between flow maximization and deployment efficiency, establishing it as a scalable and practical solution for large or resource-constrained SD-IoT environments.

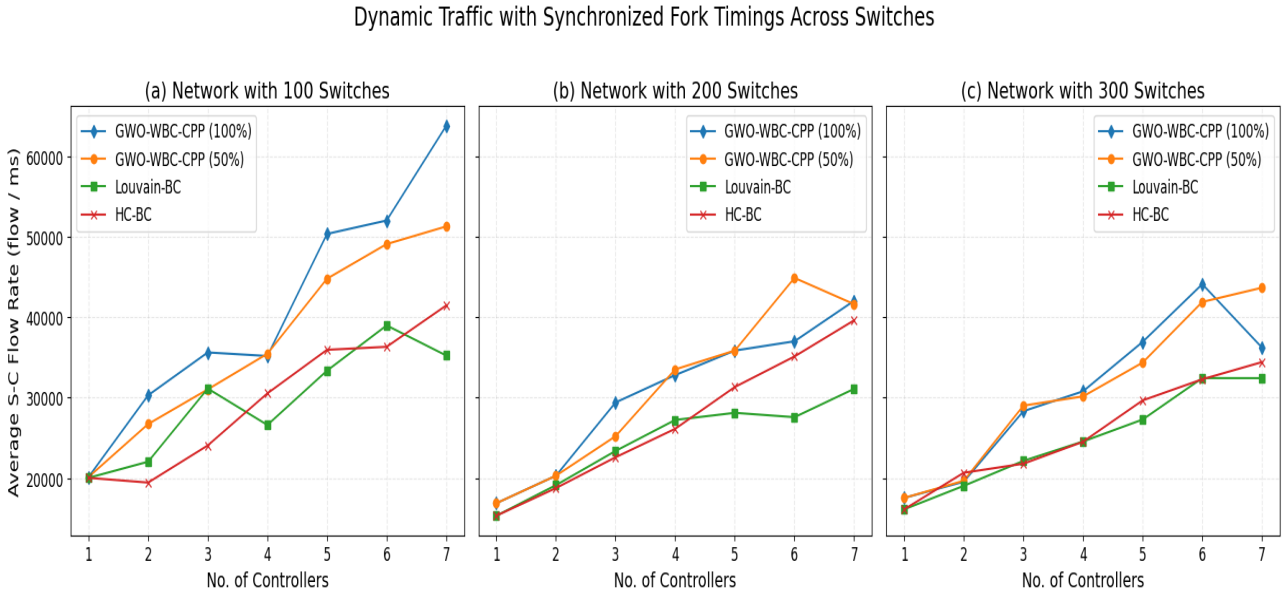


Figure 6.11: Switch to controller flow rate under synchronized fork timing.

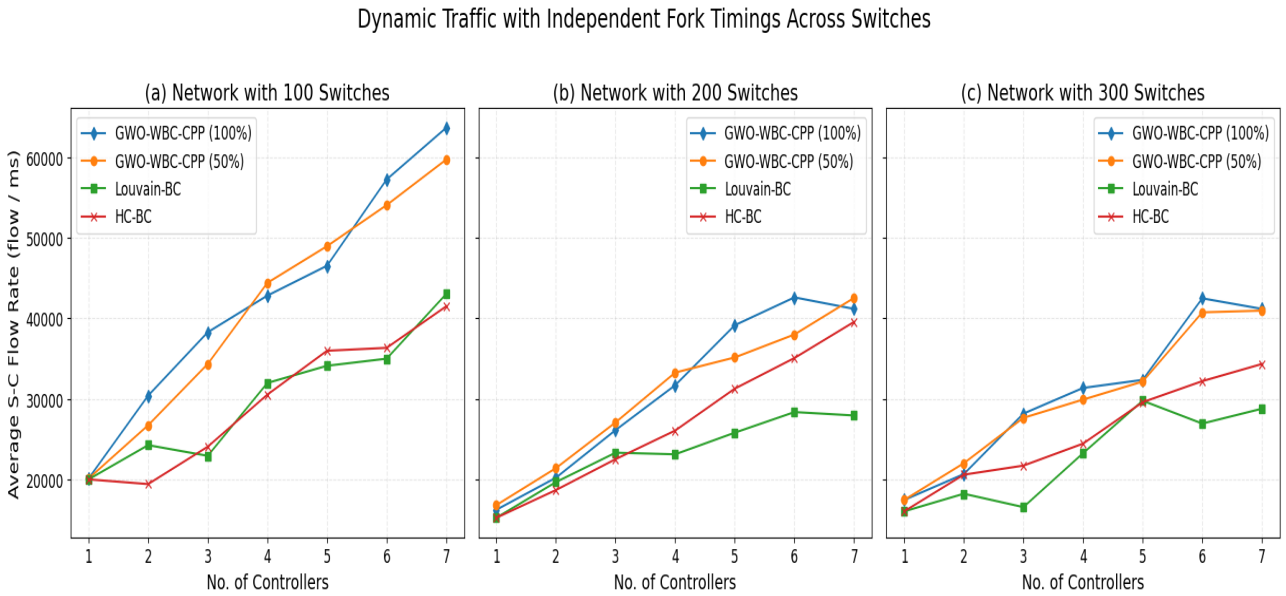


Figure 6.12: Switch to controller flow rate under independent fork timing.

6.6.4 Performance based on execution time

Execution time was analyzed for networks comprising 100, 200, and 300 switches under three traffic models: synchronized fork timing, independent fork timing, and phased four-step levels. For each

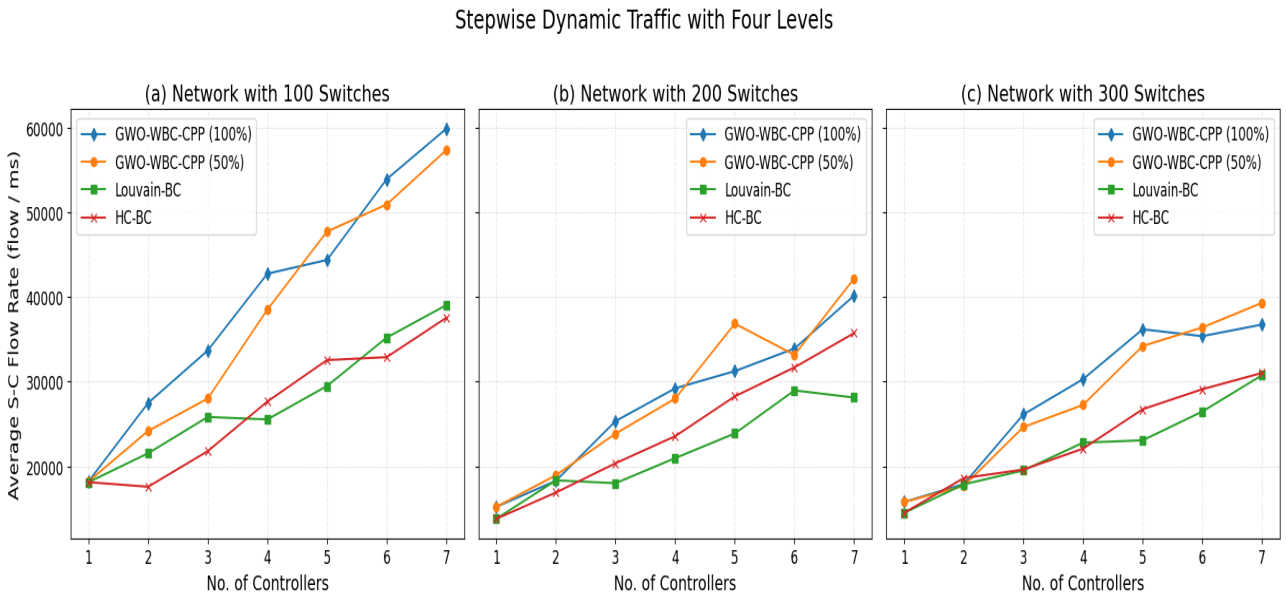


Figure 6.13: Switch to controller flow rate under four level stepwise load.

configuration, the number of controllers ranged from 1 to 7. The results, summarized in Figure. 6.14, provide an overall assessment of the scalability and computational efficiency of each method.

Both configurations of GWO-WBC-CPP, namely the 100% and 50% variants, exhibit an expected increase in execution time as the network size grows. The difference between the two primarily arises from the size of the candidate controller set and the number of optimization iterations. The 100% version evaluates all nodes and employs a higher iteration count, offering a more exhaustive search but at a higher computational cost. In contrast, the 50% configuration significantly reduces the search space and iteration count, achieving faster convergence while maintaining near-optimal performance. In several instances, it even surpasses the 100% version in terms of efficiency, validating the benefits of candidate space reduction highlighted in the preceding analyses.

Conversely, Louvain-BC and HC-BC consistently demonstrate lower execution times due to their structure-based selection mechanisms. Both algorithms rapidly form groups and assign controllers according to maximum betweenness centrality, without engaging in iterative optimization. This results in minimal computational overhead and rapid convergence, largely independent of traffic variability or the number of controllers. However, this computational simplicity comes at the cost of reduced adaptability and performance, as these methods overlook traffic heterogeneity and temporal dynamics factors crucial for realistic SD-IoT environments.

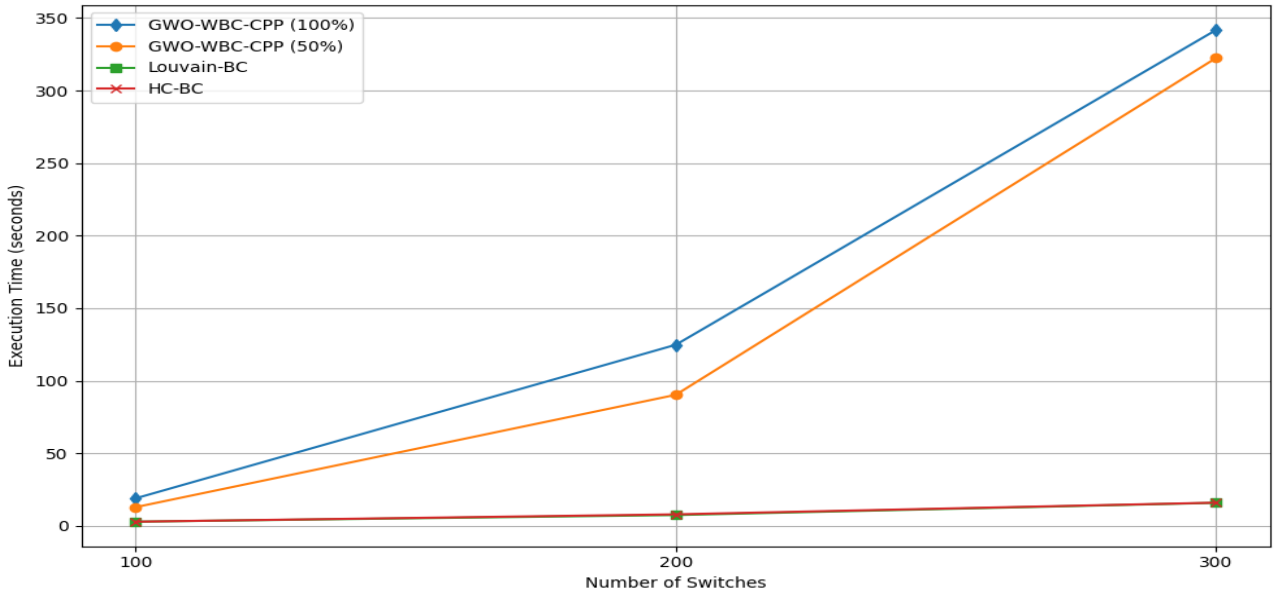


Figure 6.14: Execution time in networks.

6.7 Discussion

In this work, we proposed and evaluated two complementary approaches to address the controller deployment problem in SD-IoT networks under heterogeneous and dynamic traffic conditions. The first approach, WBC-CPP, focuses on heterogeneous switch traffic by integrating load information into the Weighted Betweenness Centrality (WBC) metric. The second, enhanced version GWO-WBC-CPP extends this logic through the Grey Wolf Optimizer (GWO), enabling efficient and scalable controller placement under both heterogeneous and dynamic traffic scenarios. Both methods were benchmarked against HC-BC, Louvain-BC, and the Optimal placement solution.

The WBC-CPP method exploits centrality-based insights by identifying nodes that act as high-load communication bridges. Assigning controllers to such nodes improves the partitioning of the control plane and enhances both data and control traffic dissemination. Experimental results confirm its strong performance in minimizing SC and CC latencies while achieving higher flow rates within the delay constraint. However, its major drawback is the elevated execution time caused by the exhaustive search over half of the network's nodes as candidate controller positions.

To overcome this limitation, the GWO-WBC-CPP approach integrates the WBC-based selection logic with the metaheuristic search capability of GWO. This hybridization efficiently guides the exploration process toward optimal controller placements, significantly reducing computational time while maintaining high placement quality. The algorithm was rigorously evaluated on scalable topologies and across three dynamic traffic models, confirming its robustness and adaptability.

Table 6.2 summarizes the comparative results across all evaluation metrics, while Table 6.3 quantifies the relative performance gains achieved by the two GWO-WBC-CPP variants over HC-BC and Louvain-BC. Together, these results provide a comprehensive perspective on both absolute efficiency and percentage-based improvement.

Overall, GWO-WBC-CPP (50%) achieves performance nearly equivalent to the exhaustive GWO-WBC-CPP (100%) in terms of average SC latency and flow rate, while significantly outperforming it in CC latency and computational efficiency. This confirms the effectiveness of reducing the candidate space without sacrificing placement quality. In comparison with HC-BC and Louvain-BC, our approach consistently outperforms across all evaluated metrics, particularly by lowering communication delays and improving flow delivery under dynamic and uneven traffic conditions limitations inherent to the static, topology-focused design of the baseline algorithms.

Despite these promising results, one limitation of the proposed framework lies in its reliance on predefined traffic parameters such as traffic rates and node weights for dynamic modeling. In real deployments, accurately estimating or measuring these features can be challenging in highly volatile or unpredictable IoT environments. Future work could focus on integrating online traffic learning or adaptive parameter estimation mechanisms to enhance autonomy and applicability in real-world SD-IoT systems.

Table 6.2: Performance comparison of algorithms, where bold values indicate the best outcomes across all approaches.

Algorithm	**Avg. SC Latency (ms)**	**SC Flow Rate (flow/ms)**	**Avg. CC Latency (ms)**
GWO-WBC-CPP (100%)	**55.01**	**2,117,304.90**	52.16
GWO-WBC-CPP (50%)	55.94	2,065,097.00	**35.36**
HC-BC	66.40	1,671,981.15	78.59
Louvain-BC	68.68	1,589,986.53	77.01

Table 6.3: Improvement percentage of GWO-WBC-CPP over HC-BC and Louvain-BC, with bold values showing the highest gain.

Algorithm	**GWO-WBC-CPP (100%)**		**GWO-WBC-CPP (50%)**	
Baseline	HC-BC	Louvain-BC	HC-BC	Louvain-BC
Avg. SC Latency (ms)	17.15%	**19.89%**	15.75%	**18.54%**
SC Flow Rate (flow/ms)	26.63%	**33.16%**	23.51%	**29.88%**
Avg. CC Latency (ms)	**33.63%**	32.27%	**55.01%**	54.08%

6.8 Conclusion

In this chapter, we enhanced the WBC-based CP by integrating Grey Wolf Optimization, addressing the execution time limitations identified earlier. The hybrid method improved efficiency while maintaining placement accuracy. A major contribution was the introduction of three dynamic traffic models, enabling a more realistic evaluation of controller placement under varying IoT flow conditions. These models highlight the importance of considering traffic variability rather than relying on static assumptions. Despite these advances, challenges remain unresolved, particularly regarding reliability in the event of controller failures. Addressing this limitation is essential to ensure robustness and continuity in SD-IoT networks. The next chapter therefore focuses on strategies to enhance fault tolerance and resilience in controller placement.

7

PSO-based reliable controller placement under heterogeneous switch load in SD-IoT

7.1 Introduction

The CP issues in SD-IoT becomes significantly more complex when accounting for heterogeneous switch loads induced by the uneven distribution of IoT devices and dynamic traffic generation. Existing solutions often neglect load variations, resulting in degraded network performance and poor scalability. In this chapter, we propose PSO-HSL, a Particle Swarm Optimization (PSO) framework specifically designed to handle heterogeneous load environments by integrating load-aware latency and reliability into the placement decision process.

7.2 Particle Swarm Optimization in SD-IoT: a brief overview

PSO is a well-established population-based meta-heuristic algorithm, originally introduced by [139], inspired by the social dynamics of flocking birds and schooling fish. The algorithm represents a group of particles working together to explore a multidimensional search space, where every particle denotes a possible solution. Through cooperation and individual learning, the swarm evolves iteratively, aiming for the best possible solution. The simplicity of PSO's design, coupled with its ability to achieve competitive results with relatively low computational overhead, has led to its widespread application in various complex optimization domains [140–142].

In the context of SD-IoT, PSO is particularly appealing for addressing the CPP, where identifying optimal controller positions has a direct effect on key network performance metrics, including latency, scalability, and fault tolerance. Traditional exhaustive search methods become impractical as network sizes increase, motivating the use of PSO to efficiently navigate large solution spaces with acceptable computational resources.

7.2.1 Fundamentals of PSO

For an optimization problem defined in a k-dimensional search space, PSO operates by initializing a collection of p particles, with each particle corresponding to a potential solution defined by its position vector $\mathbf{x}_i$ and velocity vector $\mathbf{v}_i$, with $i \in \{1, 2, \ldots, p\}$. The swarm evolves iteratively, where each particle adjusts its trajectory based on its own historical experience and the collective knowledge of the swarm.

For every iteration t, the velocity and position of particle i are updated according to these rules:

$$\mathbf{v}_i(t+1) = \omega(t) \cdot \mathbf{v}_i(t) + c_1 \cdot r_1 \cdot (pBest_i - \mathbf{x}_i(t)) + c_2 \cdot r_2 \cdot (gBest - \mathbf{x}_i(t)) \tag{7.1}$$

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{7.2}$$

Here, $\omega(t)$ denotes the inertia factor that moderates the contribution of the particle's previous velocity. The parameters c_1 and c_2 are acceleration constants reflecting the particle's own experience and the influence of the swarm, respectively. The terms r_1 and r_2 are random numbers drawn from a uniform distribution between 0 and 1. The symbol $pBest_i$ indicates the best position that particle i has achieved individually, while $gBest$ represents the best position identified collectively by all particles

in the swarm.

The conceptual behavior of particles is depicted in Figure. 7.1, illustrating how individuals explore the search space by balancing self-exploration and social learning mechanisms [143].

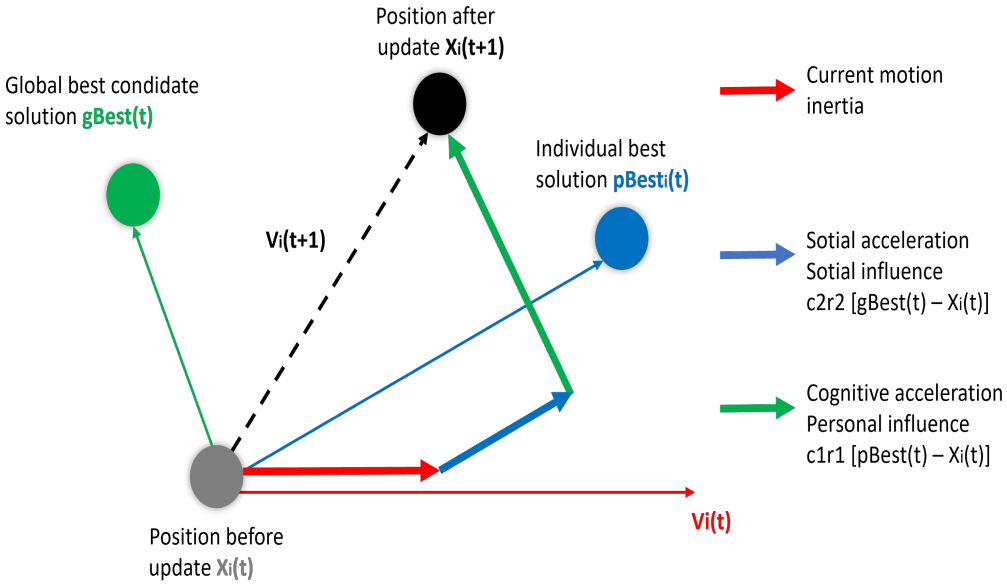


Figure 7.1: Visualization of the PSO optimization process, showing particles moving through the search space and progressively approaching the optimal solution. Components including *gBest(t)*, *pBest(t)*, and $v_i(t)$ demonstrate the evolution of particle positions and velocities across iterations.

7.2.2 Enhancing PSO with inertia weight adaptation

Standard PSO implementations often face a trade-off between exploration and exploitation, where large velocity steps facilitate global exploration but risk instability, while small steps promote convergence but may cause premature stagnation. To address this, the Linearly Decreasing Inertia Weight (LDIW) approach, proposed by Shi and Eberhart [144, 145], introduces an adaptive mechanism to control particle movement dynamically.

The inertia weight $\omega(t)$ decreases linearly over iterations, defined as:

$$\omega(t) = \omega_{\max} - (\omega_{\max} - \omega_{\min}) \times \frac{t}{\text{MaxIter}} \tag{7.3}$$

Where $\omega_{\max}$ and $\omega_{\min}$ represent the upper and lower bounds of the inertia weight, typically set to 1 and 0.1, respectively, MaxIter corresponds to the maximum iteration limit, and t refers to the current iteration step.

This adaptive strategy encourages global search in the early stages by allowing larger velocity

updates, while gradually favoring local exploitation as the swarm converges, thereby improving both solution quality and convergence speed.

7.2.3 Applicability to SD-IoT controller placement

PSO's decentralized, lightweight, and adaptive nature aligns well with the characteristics of SD-IoT networks. Given the dynamic and heterogeneous structure of IoT environments, where device distributions, traffic patterns, and loads vary considerably, static controller placement is often suboptimal. PSO provides a robust alternative by continuously exploring candidate controller placements that minimize critical metrics.

However, conventional PSO methods may overlook traffic heterogeneity, a critical factor in SD-IoT performance. This limitation motivates our proposed extension, PSO-HSL (Heterogeneous Switch Load awareness), introduced later in this chapter, which integrates load variations into the optimization process, resulting in more reliable and scalable controller placement solutions tailored to complex IoT settings.

7.3 Motivation

Optimizing CP in SD-IoT networks represents a significant challenge due to the inherent heterogeneity of IoT infrastructures. In SD-IoT environments, switches experience varying load levels driven by both the count and the operational behavior of IoT devices connected to them. These devices emit highly dynamic and unpredictable traffic patterns, resulting in substantial disparities in switch load distribution across the network.

Conventional controller placement strategies often rely on static assumptions or unweighted network models, rendering them ineffective in adapting to such heterogeneity. To demonstrate the critical impact of switch load awareness, Figure. 7.2 presents an illustrative example of a simplified network comprising 10 switches and 15 weighted edges representing inter-node distances. The load distribution among the switches is given by $\{4, 1, 2, 3, 1, 3, 1, 4, 1, 5\}$.

In the first scenario (Figure. 7.2 (a)), the network is modeled without considering switch loads, assuming uniform traffic conditions. In contrast, the second scenario (Figure. 7.2 (b)) incorporates heterogeneous loads by assigning weights to nodes proportional to their respective switch loads. This load-aware representation reveals a notable difference in the optimal controller placement outcome.

When loads are disregarded, the optimal controller position corresponds to node 6, resulting in

an average network latency of 3.21 units. However, incorporating load-awareness by weighting the average distance as $\frac{1}{n-1}\sum_{j=1}^{n} \text{Dist}(i,j) \times W_j$, where $\text{Dist}(i,j)$ is the path length linking nodes i and j, and W_j is the load associated with node j, the optimal controller position shifts to node 7, leading to an average latency improvement to 3.06 units.

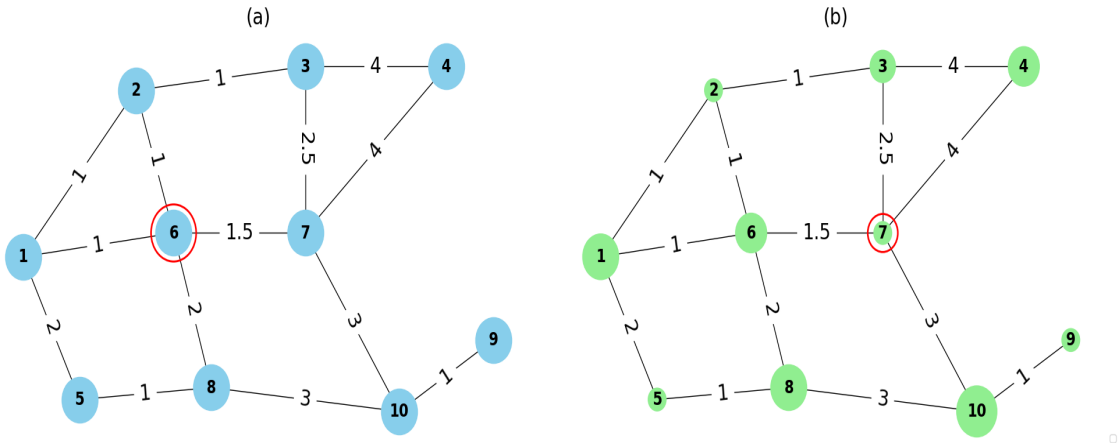


Figure 7.2: Scenarios for network modeling: (a) ignoring switch load vs. (b) accounting for switch loads.

This comparative analysis underscores the necessity of integrating heterogeneous load information into controller placement decisions to enhance network performance, particularly in latency-sensitive SD-IoT applications.

Considering the problem's complexity, which combines large solution spaces with multiple conflicting objectives, exact optimization approaches become impractical. To this end, we employ a meta-heuristic PSO-based solution for the CPP in SD-IoT networks.

The selection of PSO is motivated by its proven efficiency in exploring complex, multidimensional search spaces and its ability to converge to near-optimal solutions while maintaining computational feasibility [146]. Moreover, PSO is well-suited to optimizing multiple objectives simultaneously, including latency reduction, flow rate improvement, reliability, load balancing, and scalability enhancement.

The key contributions of this study are summarized as follows:

– Development of a PSO-based controller placement approach for SD-IoT networks that explicitly considers heterogeneous switch loads to optimize latency and network efficiency.

– Formulation of an adapted network model incorporating load variability through mathematical formulations, enabling accurate representation of heterogeneous SD-IoT environments.

– Extensive simulation-based evaluation of the proposed method, demonstrating its effectiveness in optimizing key performance metrics, including latency, flow rate, reliability, and scalability.

– Comparative analysis with existing approaches, highlighting the proposed method's superior scalability and reduced computational complexity.

7.4 Problem statement

This section presents the formal formulation of the CPP within the SD-IoT context, following the notation summarized in Table 7.1. We begin by outlining the key assumptions underlying the system model, followed by the detailed problem formulation.

7.4.1 System assumptions

The CPP in SDN consists of determining the optimal locations for controllers to minimize network latency and ensure efficient communication between switches and controllers. This problem becomes more challenging when applied to SD-IoT networks, which include numerous spatially distributed switches responsible for managing large numbers of IoT devices. The complexity arises from factors such as dynamic traffic patterns, heterogeneous device distribution, and the need for efficient resource utilization.

The network is represented as a graph, where nodes correspond to switches and edges denote communication links between them. Each switch has a fixed geographical position, and controllers are restricted to be placed only at predefined switch locations within the deployment region. Even with these simplifying assumptions, the CPP remains computationally challenging, as exact optimization methods become infeasible due to the large search space. The main system assumptions are outlined below:

- ***Network structure.*** The SD-IoT network is modeled as a connected graph comprising switches as nodes and communication links as edges, ensuring all switches are part of the network structure.

- ***Controller placement.*** Controllers can be deployed only at predefined switch locations. The number of deployed controllers K is selected within a specified range to maintain a balance between network scalability, performance, and resource efficiency.

- ***Latency and communication.*** The latency is modeled as a function of the Euclidean distance separating nodes. Every switch within a controller's domain interacts directly with that controller.

- ***Clustered environment.*** The network can be partitioned into clusters or domains, each governed by a distinct controller. Clustering is determined based on switch proximity, connectivity, or load distribution to optimize network performance.

- ***Heterogeneous load.*** Switches experience heterogeneous traffic loads, driven by the uneven distribution of IoT devices. Consequently, some switches handle significantly higher data volumes than others.

- ***Optimization objectives.*** The primary objective is minimizing the overall latency between switches and controllers while ensuring balanced resource utilization. Additional constraints may include the processing limitations of controllers and the maximum number of switches per controller.

- ***Resource constraints.*** Switches possess restricted flow generation capacity, and controllers are limited in both processing and communication capabilities, necessitating careful placement to prevent resource bottlenecks.

Table 7.1: Notation Summary for the System Model.

Notation	**Definition**
S, C, E	Sets of switches, controllers, and links
N, K	Number of switches and controllers ($K \leq \|S\|$)
W_{s_i}	Traffic load at switch s_i
U_{c_j}	Capacity of controller c_j
D_{sc}	Shortest path between s_i and c_j
ϕ_{c_j}	Switches managed by controller c_j
$x_{i,j,l}$	Assignment variable between s_i and c_j at level l
$y_{i,j,l}$	Placement variable for c_j at s_i
α	Objective weighting factor
BC_{c_j}	Weighted betweenness of controller c_j
$\sigma_{s_i,s_{i'}}(c_j)$	Paths between s_i and $s_{i'}$ via c_j
$\sigma_{s_i,s_{i'}}$	Total shortest paths between s_i and $s_{i'}$

7.4.2 Problem formulation

The CPP in SD-IoT consists of determining optimal controller placement to minimize switch to controller latency while ensuring load balancing, high reliability, scalability, and effective flow management in the presence of heterogeneous switch loads.

We represent the SD-IoT network using an undirected graph $G = (V, E)$, where $V = S \cup C$ includes the set of switches $S = \{s_1, s_2, \ldots, s_N\}$ and the set of potential controller locations $C = \{c_1, c_2, \ldots, c_K\}$, with $K \leq |V|$. Controllers are colocated with switches, and each switch s_i has an associated load W_{s_i} reflecting its generated traffic. Controller c_j has a processing capacity U_{c_j}.

The matrix *sp_dist* stores the shortest-path distances among all switches, and D_{sc} denotes the communication latency between switch s_i and controller c_j. Classical latency formulations based solely on distances fail to account for traffic heterogeneity. Thus, we define enhanced latency models incorporating switch loads:

$$SC_{AvgLat^l}(G) = \frac{1}{K} \sum_{j=1}^{K} \frac{1}{\phi_{c_j}} \sum_{i=1}^{N} D_{sc} \times W_{s_i} \times x_{i,j,l} \tag{7.4}$$

$$SC_{MaxLat^l}(G) = \frac{1}{K} \sum_{j=1}^{K} \max_{s_i \in S} D_{sc} \times W_{s_i} \times x_{i,j,l} \tag{7.5}$$

These equations yield a more realistic latency assessment by integrating the traffic load contributions of individual switches into the evaluation process.

Beyond latency minimization, we address network reliability by analyzing the system's resilience to single-controller failures. Reliability is quantified using flow-weighted betweenness centrality, where controllers with higher BC_{c_j} values are more critical due to handling significant network flows:

$$BC_{c_j} = \sum_{i=1}^{N} \sum_{i'=1}^{N} \frac{\sigma_{s_i,s_{i'}}(c_j)}{\sigma_{s_i,s_{i'}}} \times W_{s_i} \times W_{s_{i'}} \tag{7.6}$$

We define the reliability metric as the ratio of average latency before and after a controller failure:

$$Rel(G) = \frac{SC_{AvgLat^0}}{SC_{AvgLat^1}} \tag{7.7}$$

After a failure, the impacted switches are connected to the closest available controllers, considering both load balancing and traffic-aware latency to maintain network efficiency.

The overall objective function combines latency and reliability optimization, weighted by parameter α:

$$f(G) = \alpha \times SC_{AvgLat}(G) + (1 - \alpha) \times Rel(G) \tag{7.8}$$

Additionally, we consider load balancing through flow rate optimization, defined as:

$$SC_{AvgFlowLoad} = \frac{1}{K} \sum_{j=1}^{K} \frac{1}{\phi_{c_j}} \sum_{i=1}^{N} \frac{W_{s_i}}{D_{sc}} \times x_{i,j} \tag{7.9}$$

This metric reflects the effective distribution of traffic loads across controllers, promoting fair resource utilization and preventing network congestion.

7.5 Proposed PSO-HSL strategy

In this study, we propose an advanced optimization approach referred to as *PSO-HSL*, which stands for PSO with Heterogeneous Switch Load awareness. The goal of PSO-HSL is to efficiently solve the CPP in SD-IoT networks while explicitly incorporating the impact of heterogeneous and fluctuating switch loads, a critical factor influencing network performance.

Conventional controller placement methods often overlook load variability across switches, resulting in suboptimal controller distributions, increased latencies, and inefficient network resource utilization. PSO-HSL addresses these shortcomings by integrating switch load-awareness directly into the optimization process, ensuring placement decisions account for both topological distances and the uneven traffic patterns characteristic of IoT environments.

PSO-HSL builds upon the PSO algorithm, a well-established, load-based meta-heuristic inspired by the social behavior of swarms in nature. PSO is recognized for its ability to efficiently explore complex solution spaces, low computational overhead, and strong convergence characteristics, making it suitable for the multi-objective, large-scale nature of SD-IoT controller placement.

The PSO-HSL algorithm, detailed in Algorithm 4, operates as follows. It requires as inputs:

- The network switches S,
- The all-pairs shortest path matrix sp_dist,
- The count of controllers to be positioned, K,
- The maximum number of iterations Mx_itr, serving as the stopping condition,
- The vector of switch loads W reflecting the heterogeneous traffic demands.

The algorithm begins by initializing a swarm of particles, where each particle encodes a candidate controller placement, represented by a vector selecting K controllers randomly from S. For each particle, the algorithm assigns switches to their nearest controller based on sp_dist, while also considering their corresponding load W_{s_i} during assignment.

The fitness value of each particle is calculated using the cost function described in Algorithm 5 and Eq. (7.8), which integrates key performance indicators such as latency minimization, load balancing, reliability, and flow rate optimization. The resulting cost P_{cost} quantifies the quality of the placement, with P_{loca} storing the associated locations.

Each particle maintains:

– Its personal best cost PL_{cost} and location PL_{loca},

– The global best cost PG_{cost} and global best location PG_{loca} shared among all particles.

Each particle's position and velocity are iteratively updated as follows:

$$P_{V_i}^{t+1} = \omega(t) \times P_{V_i}^{t} + c_1 \times r_1 \times \left(P_{BL_i}^{t} - P_{CL_i}^{t}\right) + c_2 \times r_2 \times \left(P_{BG_p} - P_{CL_i}^{t}\right) \tag{7.10}$$

$$P_{CL_i}^{t+1} = P_{CL_i}^{t} + P_{V_i}^{t+1} \tag{7.11}$$

Where $P_{V_i}^{t}$ and $P_{CL_i}^{t}$ denote the velocity and position of particle i during iteration t, $P_{BL_i}^{t}$ is the personal best location of particle i, P_{BG_p} is the global best location, c_1 and c_2 are cognitive and social acceleration constants, r_1 and r_2 are random variables uniformly sampled from $[0, 1]$, and $\omega(t)$ is the inertia weight, linearly decreasing for exploration and exploitation trade-off.

The algorithm iteratively refines particle positions over Mx_itr iterations, continually updating personal and global best solutions based on fitness evaluations. Upon completion, PSO-HSL outputs the global best fitness value bestFit_val and the corresponding controller location Cntr_pos.

The complete workflow of PSO-HSL, from initialization to convergence, is visually illustrated in Figure. 7.3, providing a comprehensive overview of the algorithmic process.

By embedding heterogeneous load-awareness into the PSO framework, PSO-HSL ensures improved controller placement decisions, reduced network latency, enhanced scalability, and superior performance compared to conventional placement strategies, particularly in heterogeneous, large-scale SD-IoT networks.

Algorithm 4 PSO-HSL Algorithm.

Require: sp_dist, Mx_itr, k, S

Ensure: $Cntr\_pos$, $bestFit\_val$

$k \leftarrow$ number of controllers
$ppl \leftarrow$ population and its size ppls
Initialize c_1, c_2, w, ppl_s
$PL_{cost} \leftarrow \infty$
for $i \leftarrow 1$ to $ppls$ **do**
    $P_{CL_i} \leftarrow$ randomly select k controllers from S
    $P_{V_i} \leftarrow 0$
    $P_{cost}, P_{loca} \leftarrow$ fitness(sp_dist, P_{CL_i})
    if $P_{cost} < PL_{cost}$ **then**
        $PL_{cost} \leftarrow P_{cost}$
        $PL_{loca} \leftarrow P_{loca}$
    end if
end for
$PG_{cost} \leftarrow PL_{cost}$
$PG_{loca} \leftarrow PL_{loca}$
$Plbvl \leftarrow \infty$
for $t \leftarrow 1$ to Mx_itr **do**
    for $I \leftarrow 1$ to $ppls$ **do**
        $P^t_{V_i} \leftarrow$ update $P^t_{V_i}$ using Eq. 7.10
        $P^t_{CL_i} \leftarrow$ update $P^t_{CL_i}$ using Eq. 7.11
        $P_{cost}, P_{loca} \leftarrow$ fitness(sp_dist, $P^t_{CL_i}$)
        if $P_{cost} < PL_{cost}$ **then**
            $PL_{cost} \leftarrow P_{cost}$
            $PL_{loca} \leftarrow P_{loca}$
            if $PL_{cost} < PG_{cost}$ **then**
                $PG_{cost} \leftarrow PL_{cost}$
                $PG_{loca} \leftarrow PL_{loca}$
            end if
        end if
    end for
    $\omega \leftarrow$ update ω using Eq. 7.10
end for
$bestFit\_val \leftarrow PG_{cost}$
$Cntr\_pos \leftarrow PG_{loca}$
return $bestFit\_val$, $Cntr\_pos$

Algorithm 5 Fitness Calculation for Controller Placement

Require: sp_dist, K
Ensure: mbr_cst
1: sp_dist ← shortest path distance matrix
2: K ← number of controllers
3: ppl ← population ▷ |ppl| = population size
4: mbr_i ← i^{th} member in ppl
5: mbr_cost ← member cost (fitness)
6: mbr_laca ← member location
7: **for** i = 1 to |ppl| **do**
8: mbr_cst[i] ← ∅
9: **for** j = 1 to k **do**
10: Allocate each switch to the nearest controller using sp_dist, while considering the switch load and the number of switches already assigned to each controller.
11: **end for**
12: Compute mbr_cost[i] using Eq. 7.8 after that we get the mbr_laca
13: **end for**
14: **return** mbr_cst, mbr_laca

7.6 Results and discussion

The proposed approach was implemented in Python, and all network topologies were generated using Jupyter Notebook. Three network sizes 50, 100, and 150 switches were evaluated to represent small, medium, and large-scale environments. To assess scalability and the impact of network density, two simulation scenarios were defined, summarized in Table 7.2.

1. The network area scales proportionally with the number of switches (Scenario 1).

2. The network area is fixed, and only the number of switches changes (Scenario 2).

This configuration enables an in-depth analysis of how variations in node density influence the performance of CP algorithms, particularly regarding latency, scalability, and load distribution. As the network becomes denser, the challenges related to resource allocation and traffic management intensify, making them key factors in optimizing SD-IoT environments.

Both scenarios are evaluated using average switch to controller latency, flow rate, and maximum latency, with reliability and execution time also assessed for a comprehensive performance analysis.

Switches are uniformly randomly deployed within the area, each characterized by its coordinates (x, y) and corresponding traffic load. The Euclidean distance linking connected switches, s_i and $s_{i'}$,

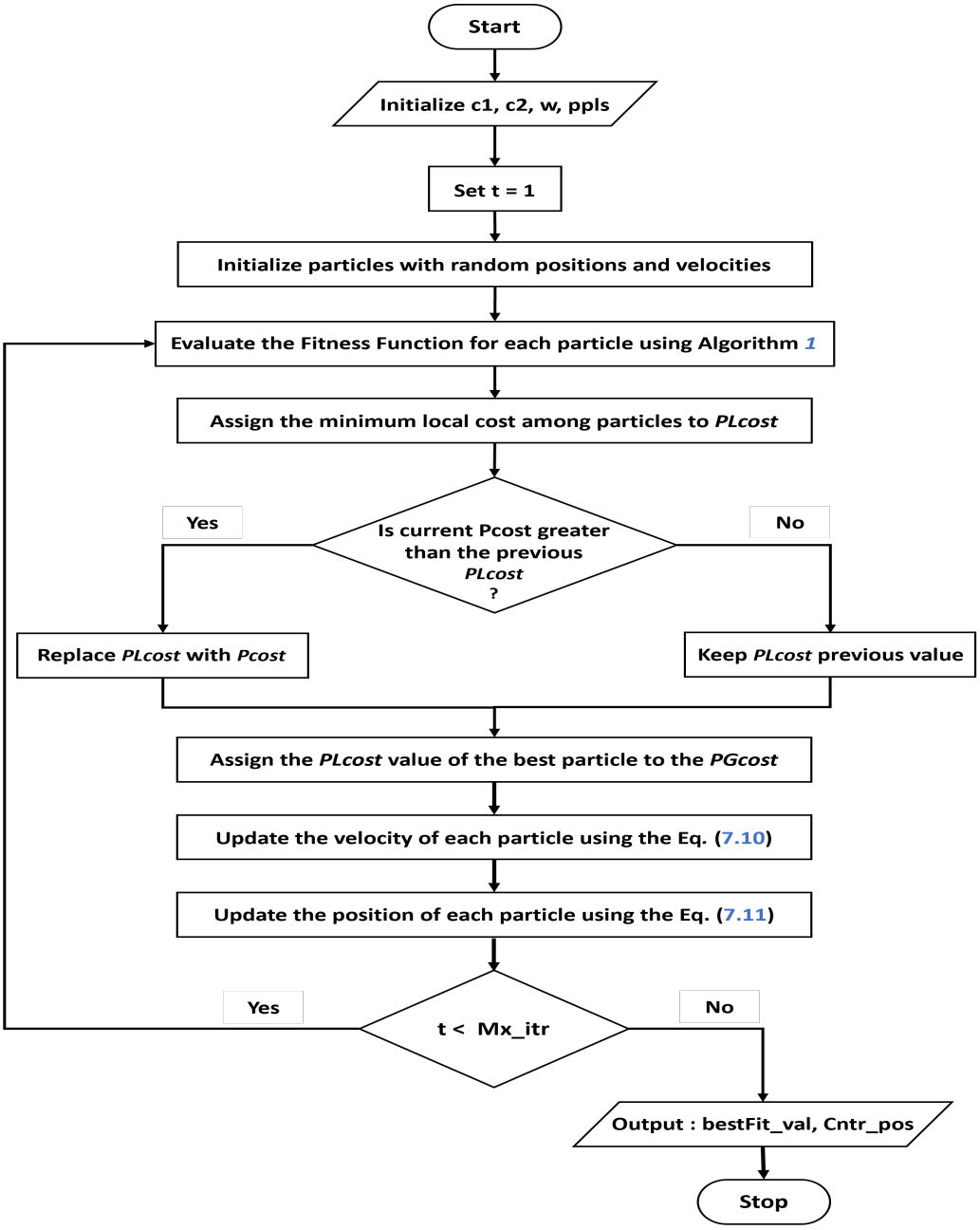


Figure 7.3: The PSO-HSL algorithm workflow.

is calculated using Eq. (3.9). Prior to the simulation, topologies are preprocessed to enhance realism by filtering closely located switches.

To benchmark the effectiveness of the proposed approach, we evaluate it against four reference methods:

- ***Opt-HSL.*** An exact optimization technique that determines the optimal controller placement by exhaustively exploring all configurations while considering heterogeneous switch loads.
- ***Opt-WHSL.*** A variant of Opt-HSL that ignores switch load heterogeneity.

Table 7.2: Network specifications for each scenario.

Scenario	**Network Type**	**Maximum Switches**	**Switches Distribution**	**Network Surface (km²)**
Scenario 1	Small Medium Large	50 100 150	Uniform, Random	0.5 M 1.0 M 1.5 M
Scenario 2	Small Medium Large	50 100 150	Uniform, Random	0.5 M

- ***Louvain-BC.*** The method proposed in [123], which integrates Louvain community detection with betweenness for CP.

- ***K-means-BC.*** Controllers assigned to top-betweenness nodes in k-means clusters.

Simulation parameters from multiple runs are summarized in Table 7.3. To identify suitable parameter values, the algorithm was executed with iterations ranging from 10 to 300. The results showed that beyond 100 iterations, the improvement in solution quality became negligible. Similarly, varying the population size from 5 to 50 with 100 iterations indicated that stable performance was observed for a population size of 20.

Table 7.3: Simulation parameters.

Parameters	Value
Maximum iterations (Mx_itr)	100
Total switchess (N)	50, 100 and 150
Total controllers (K)	1-7
Individual switch load (W_{s_i})	1-500 KFlows/s
Controller processing capacity (U_{c_i})	10000 KFlows/s
Swarm size |ppl|	20
α	0.7
W_{max}, W_{min}	1, 0.1
$c1, c2$	2
$r1, r2$	[0,1]

In the following, we present the simulation results. The total number of possible solutions for different combinations of controllers (K) and network sizes (N) is summarized in Table 7.4.

Table 7.4: Combination values C(N, K) .

N	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$	$K=6$	$K=7$
50	50	1 225	19 600	230 300	2 118 760	15 890 700	99 884 400
100	100	4 950	161 700	3 921 225	75 287 520	**1 192 052 400**	**16 007 560 800**
150	150	11 175	551 300	20 260 275	**591 600 030**	**14 297 000 725**	**294 109 729 200**

7.6.1 Average latency analysis

Figure. 7.4 illustrates that *PSO-HSL* demonstrates strong performance regarding the average switch to controller latency in small-scale networks, achieving results comparable to the exact *Opt-HSL* method. Moreover, *PSO-HSL* significantly outperforms *Opt-WHSL*, which neglects switch load heterogeneity. In contrast, both *k-means-BC* and *Louvain-BC* exhibit higher latency values overall, indicating lower efficiency. Nevertheless, all methods generally show latency reduction as the number of controllers increases, except for *k-means-BC* in medium-sized networks with five controllers and *Louvain-BC* in small networks with four controllers, where their insensitivity to switch load leads to suboptimal placements.

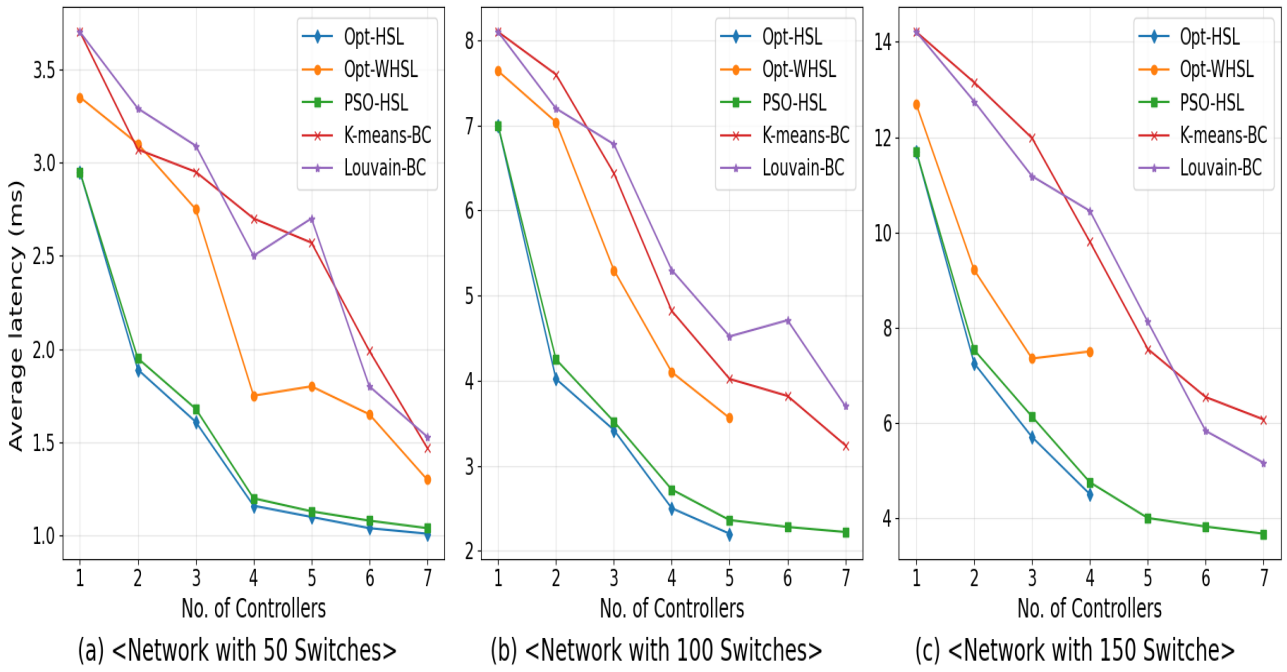


Figure 7.4: Average switch to controller latency for Scenario 1 (see Table 7.2).

7.6.2 Average flow rate analysis

As shown in Figure. 7.5, all methods generally maintain stable flow rates with increasing network load, except for Louvain-BC and k-means-BC in 50- and 100-nodes. *PSO-HSL* consistently matches the performance of the exact *Opt-HSL* approach across all scenarios, demonstrating its effectiveness. Furthermore, *PSO-HSL* yields increased flow rates than *Opt-WHSL*, highlighting the benefits of accounting for switch load in the CP strategy. In contrast, *Louvain-BC* and *k-means-BC* yield lower average rates due to their neglect of switch load and resulting cluster imbalance, highlighting the importance of load-aware methods for network performance.

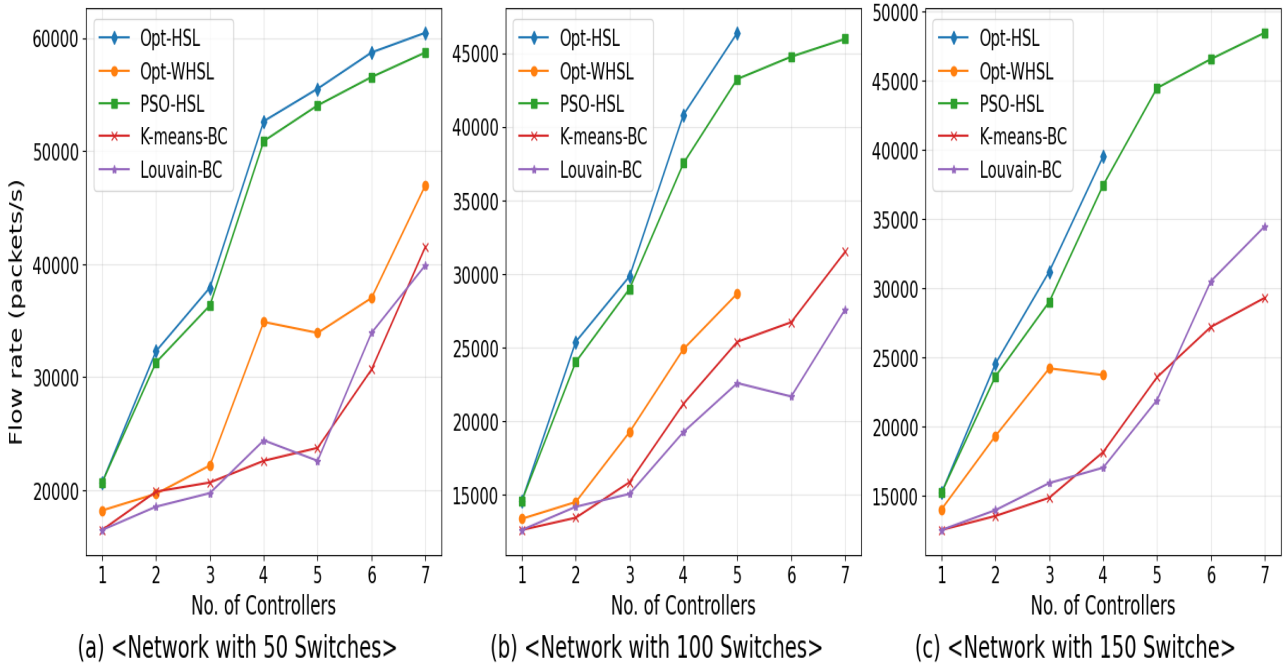


Figure 7.5: Average switch to controller flow rate for Scenario 1 (see Table 7.2).

7.6.3 Maximum latency analysis

Figure. 7.6 illustrates the effectiveness of the proposed method in minimizing maximum latency. All approaches Demonstrate lower latency as controller count increases. For the case of a single controller, *Opt-HSL* and *PSO-HSL* yield identical latency values due to the limited search space of only 50 possible placements. Similarly, *Louvain-BC* and *k-means-BC* exhibit the same latency in this scenario, as the network forms a single cluster resulting in identical controller placement. As the number of controllers grows, *PSO-HSL* consistently aligns with *Opt-HSL*, maintaining optimal performance across all cases.

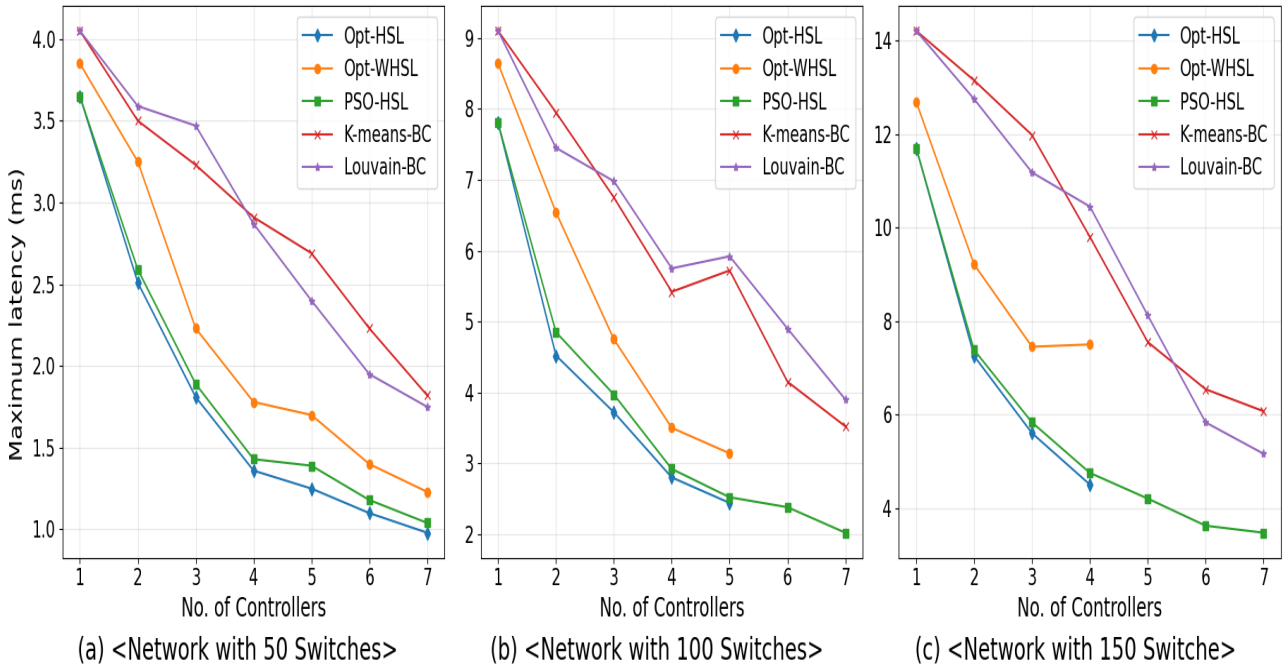
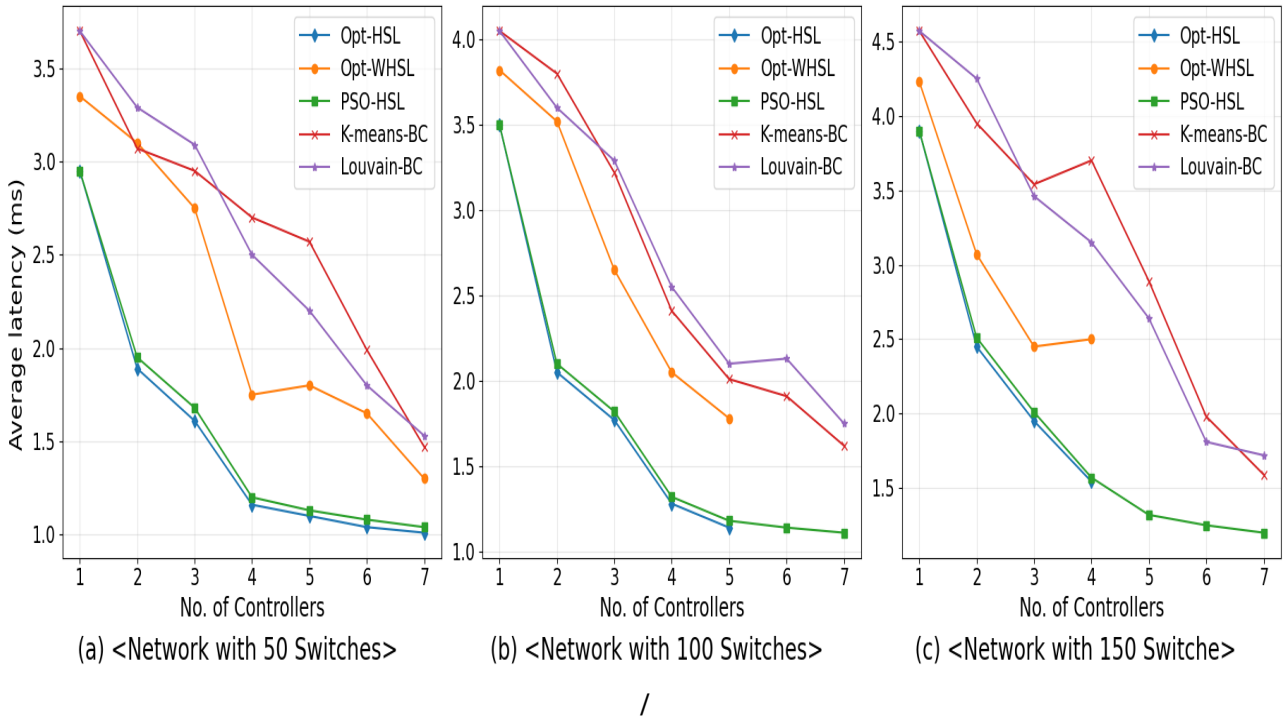


Figure 7.6: Maximum switch to controller latency for Scenario 1 (see Table 7.2).

7.6.4 Enhancing Scalability and Efficiency in Dense Fixed-Network Environments: Analysis of Scenario 2

Under *Scenario 2*, where node density increases within a fixed network area, the algorithms are evaluated for their capability to optimize controller placement under high traffic and potential load imbalances. This setup allows assessment of how network congestion and load distribution evolve as additional devices are introduced within the same geographical limits.

Figures. 7.7, 7.8, and 7.9 show the simulation results for average latency, average flow rate, and maximum latency, respectively, across three networks of fixed area. Regarding average latency, *PSO-HSL* aligns closely with *Opt-HSL* in all cases, showing its efficiency. Both methods, which consider heterogeneous switch loads, achieve significantly lower latency compared to the other approaches. For average flow rate, *PSO-HSL*, *Opt-HSL*, and *Opt-WHSL* exhibit steady increases, outperforming the remaining methods and reflecting superior load management. In terms of maximum latency, all approaches produce similar results with minor variations, as only the highest latency per cluster is considered. Within this context, *Opt-HSL* and *PSO-HSL* maintain their advantage by accounting for switch load, resulting in better performance than other methods.



/

Figure 7.7: Average switch to controller latency for Scenario 2 (see Table 7.2).

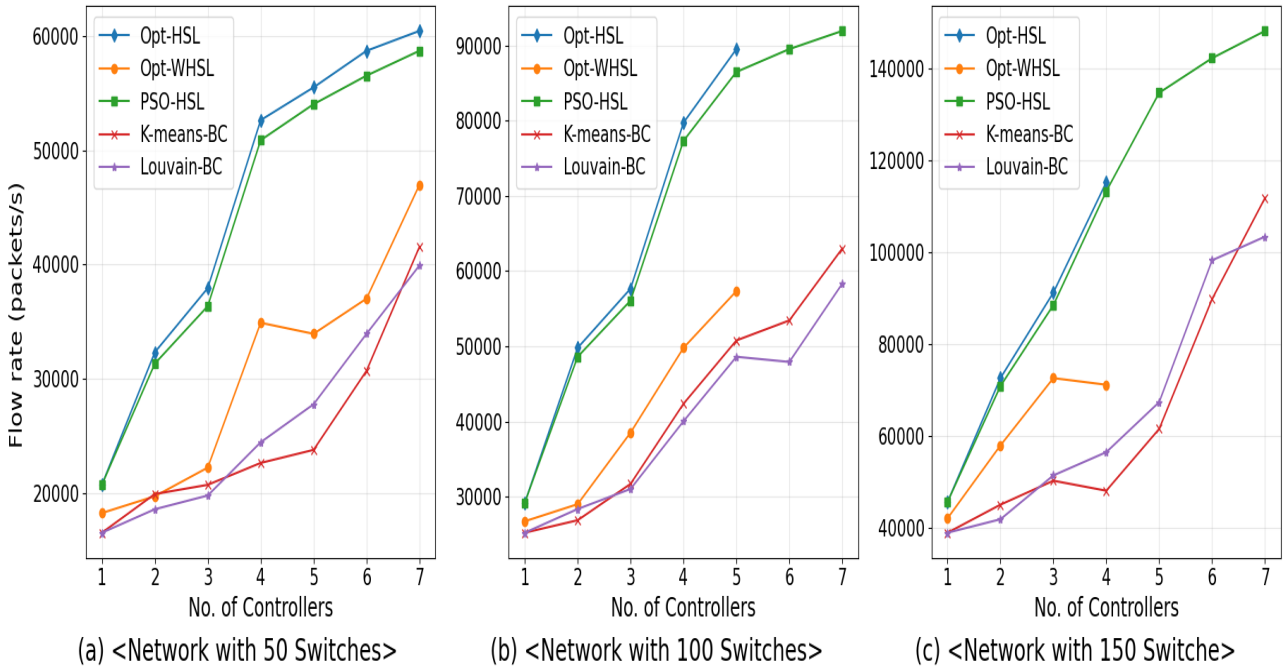


Figure 7.8: Average switch to controller flow rate for Scenario 2 (see Table 7.2).

7.6.5 Reliability analysis

To assess the proposed approaches under controller failures, with the failed controller chosen by betweenness centrality (Eq. 7.6) we analyzed the network performance when a single controller fails. Figure. 7.10 presents the reliability as the number of controllers ranges from 2 to 7. Findings indicate

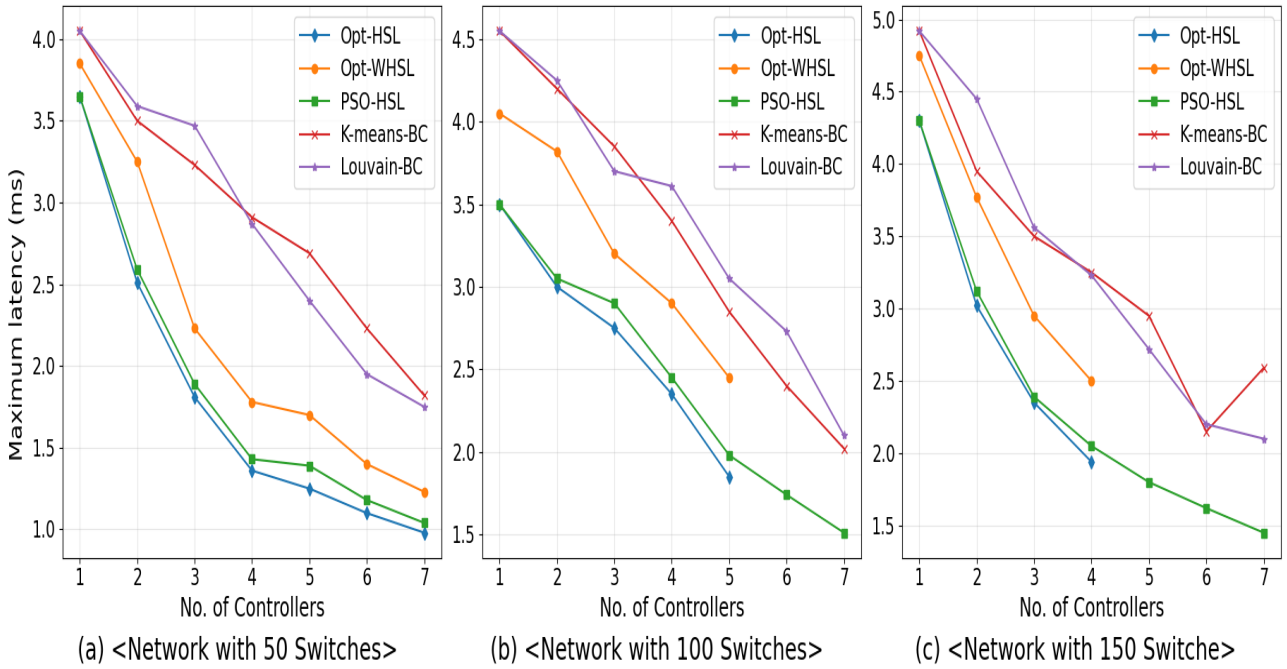


Figure 7.9: Maximum switch to controller latency for Scenario 2 (see Table 7.2).

reliability remains near-optimal with our method, whereas *Louvain-BC* and *K-means-BC* consistently exhibit lower reliability across all network configurations. Variations in the percentage improvement are influenced by the structural characteristics of specific topologies and the spatial distribution of switches and controllers, which affect algorithm performance under different network setups.

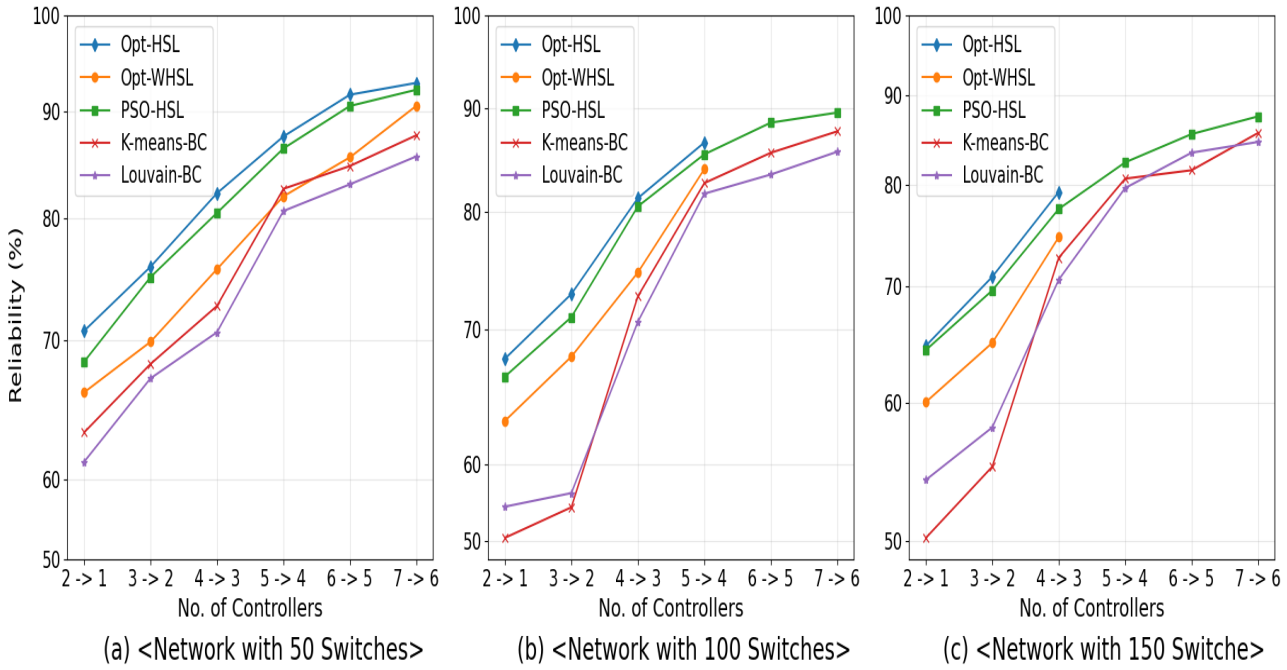


Figure 7.10: Network reliability under controller failure scenarios.

7.6.6 Execution time analysis

Figure. 7.11 presents the execution time of the five approaches across different scenarios. *PSO-HSL*, *Louvain-BC*, and *K-means-BC* maintain relatively stable runtimes, whereas the optimal solutions (*Opt-HSL* and *Opt-WHSL*) exhibit a notable rise in execution time with more controllers, regardless of whether switch load is considered. Specifically, the optimal methods perform well for fewer than three controllers, but their execution time grows exponentially with additional controllers due to the combinatorial explosion, as indicated in Table 7.4. In contrast, *PSO-HSL* consistently achieves lower execution times across all network sizes and controller counts, owing to its fixed number of iterations and population size, with only a slight increase as the number of controllers grows.

Among the heuristic approaches, *Louvain-BC* achieves the quickest execution in most network setups, followed by *K-means-BC*, except when only a single controller is deployed. These results highlight PSO-HSL's efficiency in larger networks, maintaining much lower execution times than computationally expensive optimal solutions.

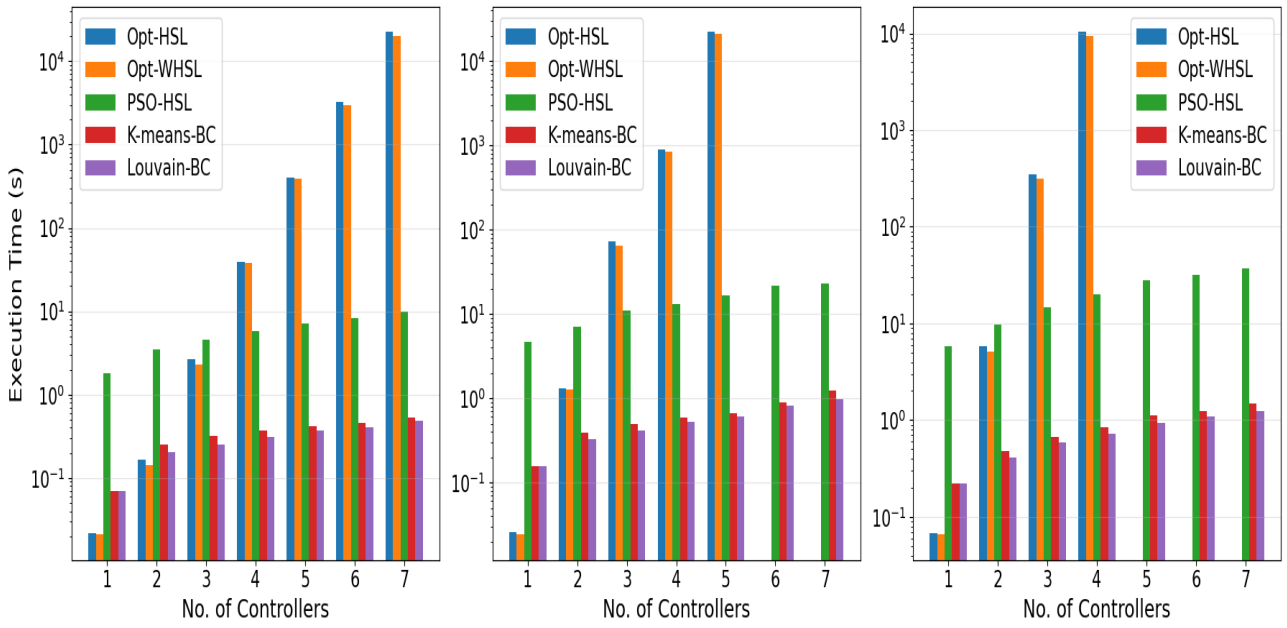


Figure 7.11: Execution time of different approaches across network scenarios.

The results confirm that the proposed *PSO-HSL* method outperforms the other approaches for several reasons. First, it explicitly accounting for diverse switch traffic in CP optimization, unlike conventional methods, which assume uniform traffic distribution. By considering switch-specific loads, *PSO-HSL* adapts to real network conditions, enabling lower latency, balanced traffic, and better controller utilization.

Moreover, leveraging the PSO meta-heuristic allows the problem to be treated as a multi-objective optimization task, simultaneously addressing switch to controller latency, network reliability, and traffic flow rate objectives that are inherently conflicting and not effectively balanced by single-objective heuristics like *K-means-BC* or *Louvain-BC*. The swarm-based nature of PSO enables efficient exploration of the solution space while avoiding local optima and directly accounting for heterogeneous switch loads (see Eq. 7.4 and Eq. 7.7). Additionally, the use of flow-WBC to identify critical controllers ensures a realistic simulation of failure scenarios, as these controllers are likely to handle the highest traffic loads or connect key subdomains.

7.7 Conclusion

This chapter introduced PSO-HSL, a meta-heuristic approach that efficiently optimizes controller placement in SD-IoT while considering heterogeneous switch loads. By incorporating traffic-aware latency and reliability into the objective function, PSO-HSL achieves enhanced performance in terms of latency reduction, load balancing, and network resilience, as confirmed by extensive simulations.

General conclusion

In this thesis, we have investigated the CPP in SD-IoT networks and proposed systematic solutions to improve network performance. We first analyzed existing approaches for CPP, highlighting their limitations in addressing the unique characteristics of IoT environments, such as heterogeneous traffic, dynamic flow patterns, and scalability constraints. Based on this analysis, we developed four main contributions, summarized in Table 7.5.

The first contribution (Section 4.2 in Chapter 4) presents an optimization approach that integrates traffic weights (flow rates) into the controller placement process. Initially, controllers were positioned in free space; subsequently, in the second contribution (Section 4.3 in Chapter 4), the model was refined to limit deployment to switch locations while supporting multiple controllers. This enhancement improves latency and load distribution, although it does not fully overcome the time complexity issue in large-scale networks.

The third contribution (Chapters 5 and 6) addresses this limitation by leveraging WBC to pre-elect candidate switches for controller deployment, thereby reducing computational complexity and accelerating the placement process. To further enhance adaptability, a meta-heuristic optimization technique GWO is integrated with the WBC-based model, incorporating dynamic traffic models that capture temporal variations in IoT flows. Three distinct flow models are evaluated to validate the method's robustness under heterogeneous traffic patterns.

Finally, the fourth contribution (Chapter 7) extends the previous work by introducing reliability as an additional optimization objective, explicitly considering controller failure scenarios. This extension ensures network resilience and sustained performance under varying traffic loads and fault conditions.

Table 7.5: Comparison of the main contributions and their characteristics.

Criteria	[147]	[148]	[149]	[150]
Method	Exact approach	Clustering K-means	Heuristic WBC & Metaheuristic GWO	Metaheuristic PSO
Number of Controllers	Single	Multiple	Multiple	Multiple
Topology Type	Real	Real	Generated	Generated
Topology Scale	9–34	58	100, 200 & 300	50, 100 & 150
Heterogenious traffic	Random	Population-based	Random	Random
Dynamic traffic	✗	✗	✓	✗
Latency	✓	✓	✓	✓
Flow Rate	✓	✗	✓	✓
Reliability	✗	✗	✗	✓
Time	✗	✗	✓	✓

Future Perspectives

While this dissertation presents significant contributions to the CPP in SD-IoT networks, several avenues remain for future research.

A key limitation of current approaches is the static nature of controller placement. Once deployed, controllers remain fixed, which reduces adaptability in response to dynamic changes in traffic load or IoT device activity. Future work could explore real-time, adaptive placement strategies using lightweight machine learning models or reinforcement learning to predict traffic patterns and dynamically reassign switches, ensuring continuous optimization of latency, flow rate, and reliability.

Scalability also remains a challenge. Larger networks with many controllers and switches may render complete controller to switch connectivity impractical. Future research should investigate efficient strategies for linking controllers to switches while maintaining performance, potentially through hierarchical or hybrid deployment architectures that combine terrestrial, aerial, and satellite nodes.

The assignment of node weights is another area for improvement. Current methods rely on simplified or approximate criteria, such as population-based proxies. Future studies could leverage real-world IoT datasets or predictive models to assign more accurate weights, reflecting actual traffic generation and device activity, thus improving placement precision.

Dynamic traffic variability presents additional challenges. Future work could integrate traffic forecasting, adaptive switch to controller assignments, or mobile controllers to respond to fluctuating network conditions, enhancing resilience, efficiency, and reliability.

Finally, emerging concerns in energy efficiency, fault tolerance, and secure operation suggest further directions. Research could focus on secure and privacy-preserving controller placement strategies, predictive mobility models for IoT devices, adaptive inter-controller coordination to reduce synchronization delays, and optimized energy consumption for constrained devices. These enhancements will support robust, scalable, and context-aware CP in heterogeneous, dynamic SD-IoT environments.

List of Publications

Journal Articles

- **Boudi, R**., Lalou, M., & Bouchemal, N. (2025). ”Enhancing controller placement in SD-IoT with dynamic heterogeneous switch traffic via weighted betweenness and GWO”. The Journal of Supercomputing, 81(14), 1-45. [149]

- **Boudi, R**., Lalou, M., & Bouchemal, N. (2025). ”PSO-HSL: A controller placement strategy using PSO with Heterogeneous Switch Load awareness in SD-IoT”. Cluster Computing. [150]

Conference Proceedings

- **Boudi, R**., Lalou, M., Bouchemal, N., & Gherbi, C. (2024, October). ”Optimizing Latency in SD-IoT through Heterogeneous Traffic Flow-Based Controller Placement”. In 2024 International Symposium on Networks, Computers and Communications (ISNCC) (pp. 1-6). IEEE. [147]

- **Boudi, R**., Lalou, M., & Bouchemal, N. (2024, November). ”On the controller placement problem: A new SD-IoT topology for the Algerian network.” In 2024 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM) (pp. 1-7). IEEE. [148]

Bibliography

[1] O. Vermesan, P. Friess, P. Guillemin, H. Sundmaeker, M. Eisenhauer, K. Moessner, F. Le Gall, and P. Cousin. *Internet of things strategic research and innovation agenda*. In *Internet of things*, pp. 7–151 (River Publishers, 2022).

[2] A. Zafar, F. Samad, H. J. Syed, A. O. Ibrahim, M. Alohaly, and M. Elsadig. *An advanced strategy for addressing heterogeneity in sdn-iot networks for ensuring qos*. Applied Sciences **13**(13), 7856 (2023). URL https://doi.org/10.3390/app13137856.

[3] I. Cheikh, S. Roy, E. Sabir, and R. Aouami. *Energy, scalability, data and security in massive iot: Current landscape and future directions*. arXiv preprint arXiv:2505.03036 (2025).

[4] K. Nisar, E. R. Jimson, M. H. A. Hijazi, I. Welch, R. Hassan, A. H. M. Aman, A. H. Sodhro, S. Pirbhulal, and S. Khan. *A survey on the architecture, application, and security of software defined networking: Challenges and open issues*. Internet of Things **12**, 100289 (2020). URL https://doi.org/10.1016/j.iot.2020.100289.

[5] S. H. Haji, S. R. Zeebaree, R. H. Saeed, S. Y. Ameen, H. M. Shukur, N. Omar, M. A. Sadeeq, Z. S. Ageed, I. M. Ibrahim, and H. M. Yasin. *Comparison of software defined networking with traditional networking*. Asian Journal of Research in Computer Science **9**(2), 1 (2021).

[6] A. Prajapati, A. Sakadasariya, and J. Patel. *Software defined network: Future of networking*. In *2018 2nd international conference on inventive systems and control (ICISC)*, pp. 1351–1354 (IEEE, 2018). URL https://doi.org/10.1109/ICISC.2018.8399028.

[7] S. K. Tayyaba, M. A. Shah, O. A. Khan, and A. W. Ahmed. *Software defined network (sdn) based internet of things (iot) a road ahead*. In *Proceedings of the international conference on future networks and distributed systems*, pp. 1–8 (2017). URL https://doi.org/10.1145/3102304.3102319.

[8] S. Siddiqui, S. Hameed, S. A. Shah, I. Ahmad, A. Aneiba, D. Draheim, and S. Dustdar. *Toward software-defined networking-based iot frameworks: A systematic literature review, taxonomy, open challenges and prospects*. IEEE Access **10**, 70850 (2022). URL https://doi.org/10.1109/ACCESS.2022.3188311.

[9] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti. *A survey of software-defined networking: Past, present, and future of programmable networks*. IEEE Communications surveys & tutorials **16**(3), 1617 (2014). URL https://doi.org/10.1109/SURV.2014.012214.00180.

[10] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. *Internet of things: A survey on enabling technologies, protocols, and applications*. IEEE communications surveys & tutorials **17**(4), 2347 (2015). URL https://doi.org/10.1109/COMST.2015.2444095.

[11] R. Horvath, D. Nedbal, and M. Stieninger. *A literature review on challenges and effects of software defined networking*. Procedia Computer Science **64**, 552 (2015). URL https://doi.org/10.1016/j.procs.2015.08.563.

[12] T. Huang, F. R. Yu, C. Zhang, J. Liu, J. Zhang, and Y. Liu. *A survey on large-scale software defined networking (sdn) testbeds: Approaches and challenges*. IEEE Communications Surveys & Tutorials **19**(2), 891 (2016). URL https://doi.org/10.1109/COMST.2016.2630047.

[13] Open Networking Foundation. *Open networking foundation* (2014). [Online; accessed 19-May-2025], URL https://www.opennetworking.org/.

[14] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. *Software-defined networking: A comprehensive survey*. Proceedings of the IEEE **103**(1), 14 (2014). URL https://doi.org/10.1109/JPROC.2014.2371999.

[15] M. Alsaeedi, M. M. Mohamad, and A. A. Al-Roubaiey. *Toward adaptive and scalable openflow-sdn flow control: A survey*. IEEE Access **7**, 107346 (2019). URL https://doi.org/10.1109/ACCESS.2019.2932422.

[16] S. A. Oladosu, C. C. Ike, P. A. Adepoju, A. I. Afolabi, A. B. Ige, and O. O. Amoo. *The future of sd-wan: A conceptual evolution from traditional wan to autonomous, selfhealing network systems*. Magna Scientia Advanced Research and Reviews **3**(2), 95 (2021). URL https://doi.org/10.30574/msarr.2021.3.2.0086.

[17] S. V. Arogundade, U. Sattar, and H. W. Khan. *Adopting open-source sd-wan: a comprehensive analysis of performance, cost, and security benefits over traditional wan architectures*. EAI Endorsed Transactions on Scalable Information Systems **12**(4) (2025).

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

[19] F. Bannour, S. Souihi, and A. Mellouk. *Distributed sdn control: Survey, taxonomy, and challenges*. IEEE Communications Surveys & Tutorials **20**(1), 333 (2017). URL https://doi.org/10.1109/COMST.2017.2782482.

[20] A. Lara, A. Kolasani, and B. Ramamurthy. *Network innovation using openflow: A survey*. IEEE communications surveys & tutorials **16**(1), 493 (2013). URL https://doi.org/10.1109/SURV.2013.081313.00105.

[21] T. Benson, A. Akella, and D. A. Maltz. *Network traffic characteristics of data centers in the wild*. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 267–280 (2010).

[22] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. *Are we ready for sdn? implementation challenges for software-defined networks*. IEEE Communications magazine **51**(7), 36 (2013). URL https://doi.org/10.1109/MCOM.2013.6553676.

[23] H. Kim and N. Feamster. *Improving network management with software defined networking*. IEEE Communications magazine **51**(2), 114 (2013). URL https://doi.org/10.1109/MCOM.2013.6461195.

[24] O. S. Specification. *Openflow switch specification: Version 1.5. 1*. Palo Alto, CA, USA (2015).

[25] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al. Onos: towards an open, distributed sdn os*. In *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6 (2014).

[26] L. Atzori, A. Iera, and G. Morabito. *Understanding the internet of things: definition, potentials, and societal role of a fast evolving paradigm*. Ad Hoc Networks **56**, 122 (2017).

[27] K. Ashton *et al. That 'internet of things' thing*. RFID journal **22**(7), 97 (2009).

[28] Oracle. *What is IoT?* https://www.oracle.com/ca-en/internet-of-things/ (no.date.). Accessed: 2025-05-21.

[29] IBM. *What is the IoT?* https://www.ibm.com/think/topics/internet-of-things (no.date.). Accessed: 2025-05-21.

[30] P. Sethi and S. R. Sarangi. *Internet of things: architectures, protocols, and applications*. Journal of electrical and computer engineering **2017**(1), 9324035 (2017). URL https://doi.org/10.1155/2017/9324035.

[31] S. M. Tahsien, H. Karimipour, and P. Spachos. *Machine learning based solutions for security of internet of things (iot): A survey*. Journal of Network and Computer Applications **161**, 102630 (2020). URL https://doi.org/10.1016/j.jnca.2020.102630.

[32] I. Mashal, O. Alsaryrah, T.-Y. Chung, C.-Z. Yang, W.-H. Kuo, and D. P. Agrawal. *Choices for interaction with things on internet and underlying issues*. Ad Hoc Networks **28**, 68 (2015).

[33] M. Wu, T.-J. Lu, F.-Y. Ling, J. Sun, and H.-Y. Du. *Research on the architecture of internet of things*. In *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, vol. 5, pp. V5–484 (IEEE, 2010).

[34] A. K. Sikder, G. Petracca, H. Aksu, T. Jaeger, and A. S. Uluagac. *A survey on sensor-based threats to internet-of-things (iot) devices and applications*. arXiv preprint arXiv:1802.02041 (2018). URL https://doi.org/10.48550/arXiv.1802.02041.

[35] R. Khan, S. U. Khan, R. Zaheer, and S. Khan. *Future internet: the internet of things architecture, possible applications and key challenges*. In *2012 10th international conference on frontiers of information technology*, pp. 257–260 (IEEE, 2012). URL https://doi.org/10.1109/FIT.2012.53.

[36] M. A. Iqbal, S. Hussain, H. Xing, and M. A. Imran. *Enabling the internet of things: fundamentals, design and applications* (John Wiley & Sons, 2020).

[37] N. M. Kumar, A. Dash, and N. K. Singh. *Internet of things (iot): an opportunity for energy-food-water nexus*. In *2018 International Conference on Power Energy, Environment and Intelligent Control (PEEIC)*, pp. 68–72 (IEEE, 2018). URL https://doi.org/10.1109/PEEIC.2018.8665632.

[38] N. M. Kumar and P. K. Mallick. *The internet of things: Insights into the building blocks, component interactions, and architecture layers*. Procedia computer science **132**, 109 (2018).

[39] C. Yizhan, W. Zhong, H. Da, and L. Ruosen. *6g is coming: Discussion on key candidate technologies and application scenarios*. In *2020 International Conference on Computer Communication and Network Security (CCNS)*, pp. 59–62 (IEEE, 2020). URL https://doi.org/10.1109/CCNS50731.2020.00022.

[40] F. M. Sallabi, H. M. Khater, A. Tariq, M. Hayajneh, K. Shuaib, and E. S. Barka. *Smart healthcare network management: A comprehensive review*. Mathematics **13**(6), 988 (2025). URL https://doi.org/10.3390/math13060988.

[41] F. Pereira, R. Correia, P. Pinho, S. I. Lopes, and N. B. Carvalho. *Challenges in resource-constrained iot devices: Energy and communication as critical success factors for future iot deployment*. Sensors **20**(22), 6420 (2020).

[42] A.-T. Shumba, T. Montanaro, I. Sergi, L. Fachechi, M. De Vittorio, and L. Patrono. *Leveraging iot-aware technologies and ai techniques for real-time critical healthcare applications*. Sensors **22**(19), 7675 (2022). URL https://doi.org/10.3390/s22197675.

[43] V. A. Shirsath and M. M. Chandane. *Beyond the basics: An in-depth analysis and multidimensional survey of programmable switch in software-defined networking*. International Journal of Networked and Distributed Computing **13**(1), 8 (2025). URL https://doi.org/10.1007/s44227-024-00049-6.

[44] A. S. George, A. H. George, and T. Baskar. *Sd-wan security threats, bandwidth issues, sla, and flaws: An in-depth analysis of ftth, 4g, 5g, and broadband technologies*. Partners Universal International Innovation Journal **1**(3), 1 (2023).

[45] T. M. C. Nguyen, D. B. Hoang, and T. D. Dang. *A software-defined model for iot clusters: Enabling applications on demand*. In *2018 International Conference on Information Networking (ICOIN)*, pp. 776–781 (IEEE, 2018). URL https://doi.org/10.1109/ICOIN.2018.8343223.

[46] S. Shafiq, M. S. Rahman, S. A. Shaon, I. Mahmud, and A. S. Hosen. *A review on software-defined networking for internet of things inclusive of distributed computing, blockchain, and mobile network technology: Basics, trends, challenges, and future research potentials*. International Journal of Distributed Sensor Networks **2024**(1), 9006405 (2024).

[47] A. H. Shamsan and A. R. Faridi. *Sdn-assisted iot architecture: a review*. In *2018 4th International Conference on Computing Communication and Automation (ICCCA)*, pp. 1–7 (IEEE, 2018). URL https://doi.org/10.1109/CCAA.2018.8777339.

[48] Y. Li, X. Su, J. Riekki, T. Kanter, and R. Rahmani. *A sdn-based architecture for horizontal internet of things services*. In *2016 IEEE international conference on communications (ICC)*, pp. 1–7 (IEEE, 2016). URL https://doi.org/10.1109/ICC.2016.7511053.

[49] M. Rostami and S. Goli-Bidgoli. *An overview of qos-aware load balancing techniques in sdn-based iot networks*. Journal of cloud computing **13**(1), 89 (2024).

[50] Y. Zhang, L. Cui, W. Wang, and Y. Zhang. *A survey on software defined networking with multiple controllers*. Journal of Network and Computer Applications **103**, 101 (2018). URL https://doi.org/10.1016/j.jnca.2017.11.015.

[51] A. Nain, S. Sheikh, M. Shahid, and R. Malik. *Resource optimization in edge and sdn-based edge computing: a comprehensive study*. Cluster Computing **27**(5), 5517 (2024). URL https://doi.org/10.1007/s10586-023-04256-8.

[52] J. Snehi, M. Snehi, D. Prasad, S. Simaiya, I. Kansal, and V. Baggan. *Sdn-based cloud combining edge computing for iot infrastructure*. Software Defined Networks: Architecture and Applications pp. 497–540 (2022).

[53] M. A. Ja'afreh, H. Adhami, A. E. Alchalabi, M. Hoda, and A. El Saddik. *Toward integrating software defined networks with the internet of things: a review*. Cluster Computing pp. 1–18 (2022). URL https://doi.org/10.1007/s10586-021-03402-4.

[54] I. E. Kamarudin, M. A. Ameedeen, M. F. Ab Razak, and A. Zabidi. *Software defined internet of things in smart city: a review*. Indonesian Journal of Electrical Engineering and Computer Science **32**(2), 915 (2023).

[55] R. Saha, N. Ahmed, and S. Misra. *Sdn-controller triggered dynamic decision control mechanism for healthcare iot*. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6 (IEEE, 2021). URL https://doi.org/10.1109/GLOBECOM46510.2021.9685911.

[56] J. Ali, C. Iwendi, G. Shan, H.-C. Wu, M. J. Alenazi, Z. B. Faheem, and C. N. Biamba. *Performance optimization of software-defined industrial internet-of-things (sd-iiot)*. IEEE Access (2024). URL https://doi.org/10.1109/ACCESS.2024.3466186.

[57] M. T. Abbas, A. Muhammad, and W.-C. Song. *Sd-iov: Sdn enabled routing for internet of vehicles in road-aware approach*. Journal of Ambient Intelligence and Humanized Computing **11**(3), 1265 (2020). URL https://doi.org/10.1007/s12652-019-01319-w.

[58] M. J. Alenazi, M. A. Al-Khasawneh, S. Rahman, and Z. B. Faheem. *Deep reinforcement learning based flow aware-qos provisioning in sd-iot for precision agriculture*. Computational Intelligence **41**(1), e70023 (2025). URL https://doi.org/10.1111/coin.70023.

[59] I. Alam, K. Sharif, F. Li, Z. Latif, M. M. Karim, S. Biswas, B. Nour, and Y. Wang. *A survey of network virtualization techniques for internet of things using sdn and nfv*. ACM Computing Surveys (CSUR) **53**(2), 1 (2020). URL https://doi.org/10.1145/3379444.

[60] A. Rahdari, A. Jalili, M. Esnaashari, M. Gheisari, A. A. Vorobeva, Z. Fang, P. Sun, V. M. Korzhuk, I. Popov, Z. Wu, *et al. Security and privacy challenges in sdn-enabled iot systems: Causes, proposed solutions, and future directions.* Computers, Materials & Continua **80**(2) (2024). URL https://doi.org/10.32604/cmc.2024.052994.

[61] A. Abderrahmane, H. Drid, and A. Behaz. *A survey of controller placement problem in sdn-iot network*. International Journal of Networked and Distributed Computing pp. 1–15 (2024). URL https://doi.org/10.1007/s44227-024-00035-y.

[62] Z. Eghbali and M. Z. Lighvan. *A hierarchical approach for accelerating iot data management process based on sdn principles*. Journal of Network and Computer Applications **181**, 103027 (2021). URL https://doi.org/10.1016/j.jnca.2021.103027.

[63] J. Ali, H. H. Song, and B.-h. Roh. *An sdn-based framework for e2e qos guarantee in internet-of-things devices*. IEEE Internet of Things Journal (2024). URL https://doi.org/10.1109/JIOT.2024.3465609.

[64] G. Wang, Y. Zhao, J. Huang, and W. Wang. *The controller placement problem in software defined networking: A survey*. IEEE network **31**(5), 21 (2017). URL https://doi.org/10.1109/MNET.2017.1600182.

[65] T. Hu, Z. Guo, P. Yi, T. Baker, and J. Lan. *Multi-controller based software-defined networking: A survey*. IEEE access **6**, 15980 (2018). URL https://doi.org/10.1109/ACCESS.2018.2814738.

[66] A. K. Singh and S. Srivastava. *A survey and classification of controller placement problem in sdn*. International Journal of Network Management **28**(3), e2018 (2018). URL https://doi.org/10.1002/nem.2018.

[67] J. Lu, Z. Zhang, T. Hu, P. Yi, and J. Lan. *A survey of controller placement problem in software-defined networking*. IEEE Access **7**, 24290 (2019). URL https://doi.org/10.1109/ACCESS.2019.2893283.

[68] T. Das, V. Sridharan, and M. Gurusamy. *A survey on controller placement in sdn*. IEEE communications surveys & tutorials **22**(1), 472 (2019). URL https://doi.org/10.1109/COMST.2019.2935453.

[69] B. P. R. Killi and S. V. Rao. *Controller placement in software defined networks: A comprehensive survey*. Computer Networks **163**, 106883 (2019). URL https://doi.org/10.1016/j.comnet.2019.106883.

[70] M. Dhar, A. Debnath, B. K. Bhattacharyya, M. K. Debbarma, and S. Debbarma. *A comprehensive study of different objectives and solutions of controller placement problem in software-defined networks*. Transactions on Emerging Telecommunications Technologies **33**(5), e4440 (2022). URL https://doi.org/10.1002/ett.4440.

[71] A. Kumari and A. S. Sairam. *A survey of controller placement problem in software defined networks*. arXiv preprint arXiv:1905.04649 (2019). URL https://doi.org/10.48550/arXiv.1905.04649.

[72] L. Agarwal, C. S. Negi, J. Sharma, and S. Jalal. *A survey on the controller placement problem in sdn*. International Journal of Advanced Networking and Applications **13**(2), 4896 (2021).

[73] B. Isong, R. R. S. Molose, A. M. Abu-Mahfouz, and N. Dladlu. *Comprehensive review of sdn controller placement strategies*. IEEE Access **8**, 170070 (2020). URL https://doi.org/10.1109/ACCESS.2020.3023974.

[74] A. Shirmarz and A. Ghaffari. *Taxonomy of controller placement problem (cpp) optimization in software defined network (sdn): a survey*. Journal of Ambient Intelligence and Humanized Computing **12**(12), 10473 (2021). URL https://doi.org/10.1007/s12652-020-02754-w.

[75] F. F. Jurado-Lasso, L. Marchegiani, J. F. Jurado, A. M. Abu-Mahfouz, and X. Fafoutis. *A survey on machine learning software-defined wireless sensor networks (ml-sdwsns): Current status*

and major challenges. IEEE Access **10**, 23560 (2022). URL https://doi.org/10.1109/ACCESS.2022.3153521.

[76] A. Abderrahmane, H. Drid, and A. Behaz. *A survey of controller placement problem in sdn-iot network*. International Journal of Networked and Distributed Computing pp. 1–15 (2024). URL https://doi.org/10.1007/s44227-024-00035-y.

[77] A. Mazumdar *et al. Software-defined wireless sensor network: A comprehensive survey*. Journal of Network and Computer Applications **215** (2023). URL https://doi.org/10.1016/j.jnca.2023.103636.

[78] W. Jiang. *Software defined satellite networks: A survey*. Digital Communications and Networks **9**(6), 1243 (2023). URL https://doi.org/10.1016/j.dcan.2023.01.016.

[79] A. M. Abdulghani, A. Abdullah, A. Rahiman, N. A. W. A. Hamid, B. O. Akram, and H. Raissouli. *Navigating the complexities of controller placement in sd-wans: A multi-objective perspective on current trends and future challenges.* Computer Systems Science & Engineering **49** (2025). URL https://doi.org/10.32604/csse.2024.058314.

[80] T. Darwish, T. A. Alhaj, and F. A. Elhaj. *Controller placement in software defined emerging networks: a review and future directions*. Telecommunication Systems **88**(1), 1 (2025). URL https://doi.org/10.1007/s11235-024-01252-0.

[81] N. S. Radam, S. Al-Janabi, and K. Shaker. *Optimisation methods for the controller placement problem in sdn: A survey*. Webology **19**(1), 3130 (2022). URL https://doi.org/10.14704/WEB/V19I1/WEB19207.

[82] H. H. Saleh, I. A. Mishkal, and D. S. Ibrahim. *Controller placement problem in software defined networks*. Indonesian Journal of Electrical Engineering and Computer Science **27**(3), 1704 (2022). URL https://doi.org/10.11591/ijeecs.v27.i3.pp1704-1711.

[83] H. Mojez, H. Kamel, R. Zanjani, and A. M. Bidgoli. *Controller placement issue in software-defined networks with different goals: a comprehensive survey*. The Journal of Supercomputing **80**(13), 19127 (2024). URL https://doi.org/10.1007/s11227-024-06230-6.

[84] B. Sapkota, B. R. Dawadi, and S. R. Joshi. *Controller placement problem during sdn deployment in the isp/telco networks: A survey*. Engineering Reports **6**(2), e12801 (2024). URL https://doi.org/10.1002/eng2.12801.

[85] B. Heller, R. Sherwood, and N. McKeown. *The controller placement problem*. ACM SIGCOMM Computer Communication Review **42**(4), 473 (2012). URL https://doi.org/10.1145/2377677.2377767.

[86] G. Yao, J. Bi, Y. Li, and L. Guo. *On the capacitated controller placement problem in software defined networks*. IEEE communications letters **18**(8), 1339 (2014). URL https://doi.org/10.1109/LCOMM.2014.2332341.

[87] B. P. R. Killi and S. V. Rao. *Optimal model for failure foresight capacitated controller placement in software-defined networks*. IEEE Communications Letters **20**(6), 1108 (2016). URL https://doi.org/10.1109/LCOMM.2016.2550026.

[88] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli. *Large-scale dynamic controller placement*. IEEE Transactions on Network and Service Management **14**(1), 63 (2017). URL https://doi.org/10.1109/TNSM.2017.2651107.

[89] B. Liu, B. Wang, and X. Xi. *Heuristics for sdn controller deployment using community detection algorithm*. In *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 253–258 (IEEE, 2016). URL https://doi.org/10.1109/ICSESS.2016.7883061.

[90] B. P. R. Killi and S. V. Rao. *Capacitated next controller placement in software defined networks*. IEEE Transactions on Network and Service Management **14**(3), 514 (2017). URL https://doi.org/10.1109/TNSM.2017.2720699.

[91] D. F. Rueda, E. Calle, and J. L. Marzo. *Improving the robustness to targeted attacks in software defined networks (sdn)*. In *DRCN 2017-Design of Reliable Communication Networks; 13th International Conference*, pp. 1–8 (VDE, 2017).

[92] G. Wang, Y. Zhao, J. Huang, and Y. Wu. *An effective approach to controller placement in software defined wide area networks*. IEEE Transactions on Network and Service Management **15**(1), 344 (2017). URL https://doi.org/10.1109/TNSM.2017.2785660.

[93] K. S. Sahoo, S. Sahoo, A. Sarkar, B. Sahoo, and R. Dash. *On the placement of controllers for designing a wide area software defined networks*. In *TENCON 2017-2017 IEEE Region 10 Conference*, pp. 3123–3128 (IEEE, 2017). URL https://doi.org/10.1109/TENCON.2017.8228398.

[94] K. S. Sahoo, D. Puthal, M. S. Obaidat, A. Sarkar, S. K. Mishra, and B. Sahoo. *On the placement of controllers in software-defined-wan using meta-heuristic approach*. Journal of Systems and Software **145**, 180 (2018). URL https://doi.org/10.1016/j.jss.2018.05.032.

[95] H. Kuang, Y. Qiu, R. Li, and X. Liu. *A hierarchical k-means algorithm for controller placement in sdn-based wan architecture*. In *2018 10th international conference on measuring technology and mechatronics automation (ICMTMA)*, pp. 263–267 (IEEE, 2018). URL 10.1109/ICMTMA.2018.00070.

[96] W. Chen, C. Chen, X. Jiang, and L. Liu. *Multi-controller placement towards sdn based on louvain heuristic algorithm*. IEEE Access **6**, 49486 (2018). URL https://doi.org/10.1109/ACCESS.2018.2867931.

[97] K. Alhazmi, A. Moubayed, and A. Shami. *Distributed sdn controller placement using betweenness centrality & hierarchical clustering*. In *Proceedings of the 8th ACM Symposium on Design and Analysis of Intelligent Vehicular Networks and Applications*, pp. 15–20 (2018). URL https://doi.org/10.1145/3272036.3272043.

[98] T. Hu, P. Yi, J. Zhang, and J. Lan. *Reliable and load balance-aware multi-controller deployment in sdn*. China Communications **15**(11), 184 (2018). URL https://doi.org/10.1109/CC.2018.8543099.

[99] T. Hirayama, T. Miyazawa, A. H. Al Mukutadir, H. Harai, and V. P. Kafle. *Salience-based distributed controllers placement in software defined networks*. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–7 (IEEE, 2018). URL https://doi.org/10.1109/GLOCOM.2018.8647498.

[100] Y. Qi, D. Wang, W. Yao, H. Li, and Y. Cao. *Towards multi-controller placement for sdn based on density peaks clustering*. In *ICC 2019-2019 IEEE international conference on communications (ICC)*, pp. 1–6 (IEEE, 2019). URL https://doi.org/10.1109/ICC.2019.8761814.

[101] S. Petale and J. Thangaraj. *Failure-based controller placement in software defined networks*. IEEE Transactions on Network and Service Management **17**(1), 503 (2019). URL https://doi.org/10.1109/TNSM.2019.2949256.

[102] A. K. Tran, M. J. Piran, and C. Pham. *Sdn controller placement in iot networks: An optimized*

submodularity-based approach. Sensors **19**(24), 5474 (2019). URL https://doi.org/10.3390/s19245474.

[103] R. Chai, Q. Yuan, L. Zhu, and Q. Chen. *Control plane delay minimization-based capacitated controller placement algorithm for sdn*. EURASIP Journal on Wireless Communications and Networking **2019**, 1 (2019). URL https://doi.org/10.1186/s13638-019-1607-x.

[104] K. Choumas, D. Giatsios, P. Flegkas, and T. Korakis. *Sdn controller placement and switch assignment for low power iot*. Electronics **9**(2), 325 (2020). URL https://doi.org/10.3390/electronics9020325.

[105] K. Kanodia, S. Mohanty, K. Kurroliya, and B. Sahoo. *Ccpgwo: A meta-heuristic strategy for link failure aware placement of controller in sdn*. In *2020 International Conference on Inventive Computation Technologies (ICICT)*, pp. 859–863 (IEEE, 2020). URL https://doi.org/10.1109/ICICT48043.2020.9112423.

[106] Y. Fan, L. Wang, and X. Yuan. *Controller placements for latency minimization of both primary and backup paths in sdns*. Computer Communications **163**, 35 (2020). URL https://doi.org/10.1016/j.comcom.2020.09.001.

[107] D. Santos, T. Gomes, and D. Tipper. *Sdn controller placement with availability upgrade under delay and geodiversity constraints*. IEEE Transactions on Network and Service Management **18**(1), 301 (2021). URL https://doi.org/10.1109/TNSM.2020.3049013.

[108] E. Calle, D. Martinez, M. Mycek, and M. Pióro. *Resilient backup controller placement in distributed sdn under critical targeted attacks*. International Journal of Critical Infrastructure Protection **33**, 100422 (2021). URL https://doi.org/10.1016/j.ijcip.2021.100422.

[109] S. K. Keshari, V. Kansal, and S. Kumar. *An intelligent way for optimal controller placements in software-defined–iot networks for smart cities*. Computers & Industrial Engineering **162**, 107667 (2021). URL https://doi.org/10.1016/j.cie.2021.107667.

[110] S. Hans, S. Ghosh, A. Kataria, V. Karar, and S. Sharma. *Controller placement in software defined internet of things using optimization algorithm*. CMC-COMPUTERS MATERIALS & CONTINUA **70**(3), 5073 (2022). URL https://doi.org/10.32604/cmc.2022.019971.

[111] J. Ali and B.-h. Roh. *An effective approach for controller placement in software-defined internet-of-things (sd-iot)*. Sensors **22**(8), 2992 (2022). URL https://doi.org/10.3390/s22082992.

[112] J. Ali and B.-h. Roh. *A novel scheme for controller selection in software-defined internet-of-things (sd-iot)*. Sensors **22**(9), 3591 (2022). URL https://doi.org/10.3390/s22093591.

[113] C. Liao, J. Chen, K. Guo, S. Liu, J. Chen, and D. Gao. *Modecp: a multi-objective based approach for solving distributed controller placement problem in software defined network*. Sensors **22**(15), 5475 (2022). URL https://doi.org/10.3390/s22155475.

[114] H. Mojez, A. M. Bidgoli, and H. H. S. Javadi. *Extended array model of star capacity-aware delay-based next controller placement problem for multiple controller failures in software-defined wide area networks*. Journal of Ambient Intelligence and Humanized Computing **14**(8), 11039 (2023). URL https://doi.org/10.1007/s12652-022-04384-w.

[115] S. Dou, L. Qi, C. Yao, and Z. Guo. *Exploring the impact of critical programmability on controller placement for software-defined wide area networks*. IEEE/ACM Transactions on Networking **31**(6), 2575 (2023). URL https://doi.org/10.1109/TNET.2023.3252639.

[116] A. Khera and U. S. Kurmi. *Enhancing performance of wide area ciot sdn by us-ml based optimum controller placement*. Research Reports on Computer Science pp. 112–121 (2023). URL https://doi.org/10.37256/rrcs.2320232637.

[117] I. O. Adebayo, M. O. Adigun, and P. Mudali. *Centrality based algorithms for controller placements in software defined wide area networks*. In *International conference on e-infrastructure and e-services for developing countries*, pp. 3–17 (Springer, 2022). URL https://doi.org/10.1007/978-3-031-34896-9_1.

[118] I. O. Adebayo, M. O. Adigun, and P. Mudali. *Neighbourhood centality based algorithms for switch-to-controller allocation in sd-wans*. In *2023 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*, pp. 1–6 (IEEE, 2023). URL https://doi.org/10.1109/icABCD59051.2023.10220485.

[119] G. D. Singh, V. Tripathi, A. Dumka, R. S. Rathore, M. Bajaj, J. Escorcia-Gutierrez, N. O. Aljehane, V. Blazek, and L. Prokop. *A novel framework for capacitated sdn controller placement: Balancing latency and reliability with pso algorithm*. Alexandria Engineering Journal **87**, 77 (2024). URL https://doi.org/10.1016/j.aej.2023.12.018.

[120] H. Babbar and S. Rani. *Pual-dbscp: Personalized ubiquitous adaptive learning for density-based splitting controller placement in software-defined networks*. Computers in Human Behavior **154**, 108135 (2024). URL https://doi.org/10.1016/j.chb.2024.108135.

[121] F. Zobary. *Optimizing sdn controller to switch latency for controller placement problem*. Informatica **48**(8) (2024). URL https://doi.org/10.31449/inf.v48i8.5846.

[122] W. wei Jiang, H. Han, Y. Zhang, and J. Mu. *Multi-controller placement in software defined satellite networks: A meta-heuristic approach*. In *2024 IEEE 99th Vehicular Technology Conference (VTC2024-Spring)*, pp. 1–7 (IEEE, 2024). URL https://doi.org/10.1109/VTC2024-Spring62846.2024.10683485.

[123] A. Abderrahmane and H. Drid. *Enhancing control placement in sdn-iot networks using the louvain algorithm and betweenness-centrality*. IEEE Access (2024). URL https://doi.org/10.1109/ACCESS.2024.3479916.

[124] S. Petale and J. Thangaraj. *Failure-based controller placement in software defined networks*. IEEE Transactions on Network and Service Management **17**(1), 503 (2019). URL https://doi.org/10.1109/TNSM.2019.2949256.

[125] G. Shafer. *Dempster-shafer theory*. Encyclopedia of artificial intelligence **1**, 330 (1992).

[126] K. Choumas, D. Giatsios, P. Flegkas, and T. Korakis. *Sdn controller placement and switch assignment for low power iot*. Electronics **9**(2), 325 (2020). URL https://doi.org/10.3390/electronics9020325.

[127] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. *Local search heuristic for k-median and facility location problems*. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pp. 21–29 (2001).

[128] B. Heller, R. Sherwood, and N. McKeown. *The controller placement problem*. ACM SIGCOMM Computer Communication Review **42**(4), 473 (2012). URL https://doi.org/10.1145/2377677.2377767.

[129] Z. Wu, Y. Gao, L. Li, J. Xue, and Y. Li. *Semantic segmentation of high-resolution remote sensing images using fully convolutional network with adaptive threshold*. Connection Science **31**(2), 169 (2019). URL https://doi.org/10.1080/09540091.2018.1510902.

[130] E. Welzl. *Smallest enclosing disks (balls and ellipsoids)*. In *New Results and New Trends in Computer Science: Graz, Austria, June 20–21, 1991 Proceedings*, pp. 359–370 (Springer, 2005). URL https://doi.org/10.1007/BFb0038202.

[131] D. Knight. *Internet topology zoo*. http://www.topology-zoo.org/ (2011). Accessed June 2024.

[132] Internet2. *Internet2: Optical services for the 21st century (os3e) network topology* (2013). https://www.internet2.edu/architectures/os3e/.

[133] S. Patro and K. K. Sahu. *Normalization: A preprocessing stage*. arXiv preprint arXiv:1503.06462 (2015).

[134] K. Okamoto, W. Chen, and X.-Y. Li. *Ranking of closeness centrality for large-scale social networks*. In *International workshop on frontiers in algorithmics*, pp. 186–195 (Springer, 2008). URL https://doi.org/10.1007/978-3-540-69311-6_21.

[135] P. Hage and F. Harary. *Eccentricity and centrality in networks*. Social networks **17**(1), 57 (1995). URL https://doi.org/10.1016/0378-8733(94)00248-9.

[136] U. Brandes. *A faster algorithm for betweenness centrality*. Journal of mathematical sociology **25**(2), 163 (2001). URL https://doi.org/10.1080/0022250X.2001.9990249.

[137] A. Abderrahmane and H. Drid. *Enhancing control placement in sdn-iot networks using the louvain algorithm and betweenness-centrality*. IEEE Access (2024). URL https://doi.org/10.1109/ACCESS.2024.3479916.

[138] S. Mirjalili, S. M. Mirjalili, and A. Lewis. *Grey wolf optimizer*. Advances in engineering software **69**, 46 (2014). URL https://doi.org/10.1016/j.advengsoft.2013.12.007.

[139] R. Eberhart and J. Kennedy. *A new optimizer using particle swarm theory*. In *MHS'95. Proceedings of the sixth international symposium on micro machine and human science*, pp. 39–43 (Ieee, 1995). URL https://doi.org/10.1109/MHS.1995.494215.

[140] Y. Zhang, S. Wang, and G. Ji. *A comprehensive survey on particle swarm optimization algorithm and its applications*. Mathematical problems in engineering **2015**(1), 931256 (2015). URL https://doi.org/10.1155/2015/931256.

[141] X.-F. Liu, Z.-H. Zhan, Y. Gao, J. Zhang, S. Kwong, and J. Zhang. *Coevolutionary particle swarm optimization with bottleneck objective learning strategy for many-objective optimization*.

IEEE Transactions on Evolutionary Computation **23**(4), 587 (2018). URL https://doi.org/10.1109/TEVC.2018.2875430.

[142] Y. Li, Z.-H. Zhan, S. Lin, J. Zhang, and X. Luo. *Competitive and cooperative particle swarm optimization with information sharing mechanism for global optimization problems*. Information Sciences **293**, 370 (2015). URL https://doi.org/10.1016/j.ins.2014.09.030.

[143] M. Jain, V. Saihjpal, N. Singh, and S. B. Singh. *An overview of variants and advancements of pso algorithm*. Applied Sciences **12**(17), 8392 (2022). URL https://doi.org/10.3390/app12178392.

[144] Y. Shi and R. Eberhart. *A modified particle swarm optimizer*. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE world congress on computational intelligence (Cat. No. 98TH8360)*, pp. 69–73 (IEEE, 1998). URL https://doi.org/10.1109/ICEC.1998.699146.

[145] Y. Shi and R. C. Eberhart. *Empirical study of particle swarm optimization*. In *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*, vol. 3, pp. 1945–1950 (IEEE, 1999). URL https://doi.org/10.1109/ICEC.1998.699146.

[146] C. Gao, H. Wang, F. Zhu, L. Zhai, and S. Yi. *A particle swarm optimization algorithm for controller placement problem in software defined network*. In *Algorithms and Architectures for Parallel Processing: 15th International Conference, ICA3PP 2015, Zhangjiajie, China, November 18-20, 2015, Proceedings, Part III 15*, pp. 44–54 (Springer, 2015). URL https://doi.org/10.1007/978-3-319-27137-8_4.

[147] R. Boudi, M. Lalou, N. Bouchemal, and C. Gherbi. *Optimizing latency in sd-iot through heterogeneous traffic flow-based controller placement*. In *2024 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6 (IEEE, 2024). URL https://doi.org/10.1007/s10586-025-05672-8.

[148] R. Boudi, M. Lalou, and N. Bouchemal. *On the controller placement problem: a new sd-iot topology for the algerian network*. In *2024 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)*, pp. 1–7 (IEEE, 2024). URL https://doi.org/10.1109/ICT-DM62768.2024.10798934.

[149] R. Boudi, M. Lalou, and N. Bouchemal. *Enhancing controller placement in sd-iot with dynamic*

heterogeneous switch traffic via weighted betweenness and gwo. The Journal of Supercomputing **81**(14), 1 (2025). URL https://doi.org/10.1007/s11227-025-07783-w.

[150] R. Boudi, M. Lalou, and N. Bouchemal. *Pso-hsl: A controller placement strategy using pso with heterogeneous switch load awareness in sd-iot*. Cluster Computing **28**(15), 977 (2025). URL https://doi.org/10.1007/s10586-025-05672-8.