Cost-Effective Algorithms for the Controller Placement Problem in Software-Defined Multihop Wireless Networks

by

Afsane Zahmatkesh, M.Sc.

A dissertation submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering,
Carleton University
Ottawa, Ontario
May, 2021

© Copyright
Afsane Zahmatkesh, 2021
The undersigned hereby recommends to the
Faculty of Graduate and Postdoctoral Affairs
acceptance of the dissertation

Cost-Effective Algorithms for the Controller Placement Problem
in Software-Defined Multihop Wireless Networks

submitted by Afsane Zahmatkesh, M.Sc.

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical and Computer Engineering

__________________________
Professor Chung-Horng Lung, Thesis Supervisor

__________________________
Professor Vojislav Misic, External Examiner

__________________________
Professor Amir H. Banihashemi, Chair
Department of Systems and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering,
Carleton University
May, 2021
Abstract

One key feature of software-defined networking (SDN) is a centralized control plane. However, having a central controller has some disadvantages, e.g., scalability and reliability, especially for multihop wireless networks (MWNs). Different models with multiple controllers have been proposed, each controller managing a smaller domain and collaborating with others. In an SDN architecture, solving the controller placement problem (CPP) in a multi-controller environment plays an important role on network performance in terms of delay, reliability, control overhead, etc. In this thesis, control overhead, referred to as the network cost, consists of controller-device communications to discover the network topology, exchange configurations and set up flow tables as well as inter-controller communications, if needed, to synchronize different network views and achieve the global view of the network. In software-defined multihop wireless networking (SDMWN), because of the capacity limitation and the effect of interference on wireless links, and an in-band architecture in some types of networks to exchange both data and control traffic, it is important to solve the CPP while minimizing the control overhead to reduce energy consumption, have lower packet losses and improve reliability. Therefore, the objective of the thesis is to solve the CPP in SDMWN while minimizing the number of required control packets to be exchanged in the control plane. The novelty of this work is proposing two different nonlinear optimization models while considering the characteristics of SDMWN and the capacity of wireless links to solve the CPP and select routes among network devices and controllers in the network. The results demonstrate the impact of different factors such as the number of controllers, the capacity of wireless links and the arrival rate of new flows in devices on control overhead, the average number of controller-device hops and the average number of inter-controller hops in SDMWN.

However, due to the high computational complexity of the proposed optimization models, the thesis further investigates and extends the existing heuristic algorithms in the literature to solve the simplified linear optimization model in SDMWN without considering the capacity of links as a constraint. A genetic algorithm (GA) based heuristic approach is
proposed to solve the simplified optimization problem. The existing GA-based approaches to the CPP are proposed for wired networks and the thesis proposes an adopted GA approach to the CPP in SDMWN. The results demonstrate that the proposed approach can derive the same optimal solution for a small network faster compared to the simplified optimization model and generate fewer control overhead compared to the extended heuristic algorithm based on the random placement. Moreover, results demonstrate that the GA-based heuristic can solve larger networks in a short period of time, making it feasible for non-trivial network sizes.

Moreover, three heuristic algorithms are proposed to the CPP in SDMWN: connectivity-degree based, distance-degree based and randomly-selected algorithms. The results demonstrate that using ranking degrees assigned to the possible controller placements, including the average distance to other devices as a degree or the connectivity degree of each placement, the extended heuristic algorithms are able to achieve the optimal solution in small-scale networks in terms of the generated control overhead, the number of controllers selected in the network as well as the average number of controller-device hops and the average number of inter-controller hops. In addition, these algorithms are able to generate near-optimal solutions and the computational times for all three heuristic algorithms are significantly lower than that of the optimal solution, which make them more practical.

Recently, deep reinforcement learning (Deep RL) has been received a significant attention in different areas and dynamic environments. In this thesis, a deep Q-network (DQN) algorithm is proposed to investigate the potential of using Deep RL in solving the CPP in SDMWN. In addition, a preliminary mobility scenario is considered in SDWMN with mobile devices and a minimal mobility. The results show that the proposed DQN algorithm is able to find optimal and near-optimal solutions in a small-scale network when devices are mobile.
To My Parents

for their endless love and support

and

for giving me the courage to follow my dreams
Acknowledgements

I would like to express my deep and sincere gratitude to my supervisor, Professor Chung-Horng Lung for his invaluable guidance, generous support, profound knowledge and precious patience. His guidance helped me in all the time of research and writing of this thesis.

I am grateful to Professor Vojislav Misic, from the Department of Computer Science, Ryerson University to serve as the external examiner of my Ph.D. defense. I would also like to thank other committee members of my defense: Professor Amiya Nayak, Professor Nicola Santoro and Professor Marc St-Hilaire for their valuable suggestions and discussions.

Last but not least, I would like to express my gratitude for my parents who always supported me and gave me the courage to follow my dreams. Words cannot truly express how grateful I am to my mother, my father and my brothers for their endless love, constant support and continuous encouragement throughout my life and my studies. This accomplishment would not have been possible without them. It is my great pleasure to dedicate this thesis to my family.
# Table of Contents

Abstract \hspace{1cm} i

Acknowledgements \hspace{1cm} iv

List of Tables \hspace{1cm} viii

List of Figures \hspace{1cm} x

Abbreviations \hspace{1cm} xii

List of Symbols \hspace{1cm} xiv

1 Introduction \hspace{1cm} 1

1.1 Background \hspace{1cm} 1

1.2 Problem Statement and Research Objective \hspace{1cm} 3

1.3 Contributions \hspace{1cm} 6

1.4 Organization \hspace{1cm} 8

1.5 Publications \hspace{1cm} 8

2 Literature Review \hspace{1cm} 10

2.1 Introduction \hspace{1cm} 10

2.2 Distributed Control Plane in SDMWN \hspace{1cm} 12
2.2.1 Flat Architecture ....................................................... 13
2.2.2 Hierarchical Architecture ............................................. 14
2.2.3 Proposed Control Plane Architectures in this Thesis .......... 15
2.3 Controller Placement Problem in Wired Networks ................. 15
2.4 Controller Placement Problem in Wireless Networks .............. 19
2.5 Heuristic Approaches to the CPP ....................................... 22
2.6 Deep RL Approaches to the CPP ....................................... 27
2.7 Open Research Areas ..................................................... 28

3 Optimization Models for SDMWN Controller Placement Problem 30
3.1 Introduction .............................................................. 30
3.2 Model Formulation ....................................................... 31
  3.2.1 Notations .......................................................... 32
  3.2.2 Model Outputs .................................................... 32
3.3 TDCD Optimization Model .............................................. 34
  3.3.1 Objective Function .................................................. 34
  3.3.2 Constraints ........................................................ 36
3.4 TDCCCD Optimization Model .......................................... 38
  3.4.1 Objective Function .................................................. 38
  3.4.2 Constraints ........................................................ 40
3.5 Model Results and Analysis ........................................... 41
  3.5.1 The Impact of the Number of Controllers On the Network Cost 42
  3.5.2 The Impact of $R_{Flow,Rq}$ On the Network Cost ................. 46
  3.5.3 The Impact of Link Capacity On the Number of Controllers .... 48
  3.5.4 The Impact of the Number of Controllers On the Average Number of Hops 49
  3.5.5 The Impact of Increasing the Number of Devices On the Execution time of the optimization problem 50
3.6 Summary ................................................................. 50
# List of Tables

3.1 Notations used in Chapter 3 ............................................. 33
3.2 Optimal cost of placing one controller in both TDCD model and TDCCCD model (control packets/second) ............................................. 43
3.3 Optimal cost of placing \( N \) controllers in TDCD model (control packets/second) 43
3.4 Optimal cost of placing \( N \) controllers in TDCCCD model (control packets/second) ............................................. 44
3.5 Optimal network cost (control packets/second) in TDCD vs. TDCCCD ............................................. 46
3.6 Average number of controller-device hops and average number of inter-controller hops in the topology as shown in Fig. 3.1 using the TDCCCD model ............................................. 49

4.1 Notations used in Chapter 4 to define the simplified TDCCCD model ............................................. 55

5.1 Minimum cost obtained from the proposed algorithms and the optimal solution from the simplified TDCCCD model (control packets/second) ............................................. 74
5.2 Minimum cost obtained from the proposed heuristic algorithms in large-scale networks (control packets/second) ............................................. 76
5.3 Execution time of the simplified TDCCCD model vs. execution time of the proposed heuristic algorithms (second) ............................................. 77
5.4 Execution time of the proposed heuristic algorithms in large-scale networks (second) ............................................. 77

6.1 Values used for the defined parameters in the proposed DQN ............................................. 88
6.2 Optimal results vs. the results from the proposed DQN algorithm when \( N_{ep} = 30 \) ............................................. 89
6.3 Optimal results vs. the results from the adapted DQN algorithm when \( N_{ep} = 30 \) and \( N = 3 \) ................................................................. 91

6.4 The selected controller placements in the simplified TDCCCD model presented in Section 4.2 vs. the adapted DQN Algorithm when \( N_{ep} = 30 \) and \( N = 3 \) ................................................................. 92

6.5 Minimum cost obtained from the simplified TDCCCD model presented in Section 4.2 vs. the results of the adapted DQN Algorithm (control packets/second) when \( N_{ep} = 30 \) and \( N = 3 \) ................................................................. 93

6.6 Controller placements, minimum cost achieved (control packets/seconds) and execution time (seconds) for the mobility scenario when \( N_{ep} = 30 \) .... 97

6.7 Average number of controller-device hops and average number of inter-controller hops for the mobility scenario ............................................. 97
List of Figures

3.1 An SDMWN with 6 network devices ................................. 42
3.2 The cost of topology discovery in the network achieved from the TDCCCD model (control packets/second) ........................................ 45
3.3 The total network cost achieved from the TDCCCD model (control packets/second) ........................................ 45
3.4 Impact of number of controllers and controller placements on the network cost ........................................ 46
3.5 The impact of increasing the value of $R_{Flow,Rq}$ in the TDCCCD model on the network cost ................................. 47
3.6 The impact of increasing the value of $R_{Flow,Rq}$ in the TDCD model on the network cost ................................. 47
4.1 Chromosome description in the GA-based approach to the CPP in SDMWN ........................................ 58
4.2 A chromosome in SDMWN with 10 devices and 5 controllers .................... 58
4.3 Crossover operator in the GA-based approach to the CPP in SDMWN .................. 62
4.4 An SDMWN with 10 wireless network devices .......................... 63
4.5 Average total network cost in the topology in Fig. 4.4 for the proposed GA algorithm with different number of generations vs. random placement algorithm with 95% confidence interval ........................................ 64
4.6 Average number of controller-device hops vs. average number of inter-controller hops in the topology as shown in Fig. 4.4 ................................. 64
4.7 Average execution time (seconds) in the topology as shown in Fig. 4.4 when increasing the number of devices for different numbers of generations ................................. 65
4.8 Average execution time (seconds) when increasing the number of devices for different numbers of generations when $N = 2$ .......................... 66

5.1 Topology 1- an SDMWN with 10 wireless network devices ...................... 73
5.2 Topology 2- an SDMWN with 10 wireless network devices ...................... 73
5.3 The number of devices versus the number of controllers selected in the presented algorithms and the TDCCCD model .............................. 75
5.4 Optimal solution vs. the proposed heuristic algorithms .......................... 78
5.5 The number of devices versus the number of controllers selected in the presented algorithms in large-scale network .............................. 79
5.6 The number of devices versus the average number of hops between devices and their assigned controllers in the presented algorithms in large-scale network 79
5.7 The number of devices versus the average number of hops between controllers in the presented algorithms in large-scale network .................. 80

6.1 Execution time for running the proposed DQN algorithm for $N = 2$ and $N = 3$ when increasing the number of episodes .............................. 89
6.2 Execution time for running the proposed DQN when increasing the number of devices in the network for $N_{ep} = 30$ ................................. 90
6.3 An SDMWN with 10 wireless network devices ..................................... 91
6.4 The simplified TDCCCD model vs. the adapted DQN algorithm when $N_{ep} = 30$ and $N = 3$ in terms of the average number of inter-controller hops ................................. 94
6.5 The simplified TDCCCD model vs. the adapted DQN algorithm when $N_{ep} = 30$ and $N = 3$ in terms of the average number of controller-device hops ................................. 94
6.6 Running the proposed DQN algorithm on SDMWN with mobile devices .................. 96
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMPL</td>
<td>A Mathematical Programming Language</td>
</tr>
<tr>
<td>API</td>
<td>Southbound Application-programming Interface</td>
</tr>
<tr>
<td>CPP</td>
<td>Controller Placement Problem</td>
</tr>
<tr>
<td>CSMA</td>
<td>Carrier Sense Multiple Access</td>
</tr>
<tr>
<td>CSMA/CA</td>
<td>Carrier Sense Multiple Access With Collision Avoidance</td>
</tr>
<tr>
<td>CSMA/CD</td>
<td>Carrier Sense Multiple Access With Collision Detection</td>
</tr>
<tr>
<td>Deep RL</td>
<td>Deep Reinforcement Learning</td>
</tr>
<tr>
<td>DQN</td>
<td>Deep Q-network</td>
</tr>
<tr>
<td>GA</td>
<td>Genetic Algorithm</td>
</tr>
<tr>
<td>ILP</td>
<td>Integer Linear Program</td>
</tr>
<tr>
<td>IoV</td>
<td>Internet Of Vehicles</td>
</tr>
<tr>
<td>LLDP</td>
<td>Link Layer Discovery Protocol</td>
</tr>
<tr>
<td>MANETs</td>
<td>Mobile Ad-hoc Networks</td>
</tr>
<tr>
<td>MILP</td>
<td>Mixed-integer Linear Program</td>
</tr>
<tr>
<td>MIP</td>
<td>Mixed Integer Program</td>
</tr>
<tr>
<td>MWNs</td>
<td>Multihop Wireless Networks</td>
</tr>
<tr>
<td>NLP</td>
<td>Nonlinear Programming Problem</td>
</tr>
<tr>
<td>NP</td>
<td>Non-deterministic Polynomial-time</td>
</tr>
<tr>
<td>RL</td>
<td>Reinforcement Learning</td>
</tr>
<tr>
<td>RSUs</td>
<td>Road Side Units</td>
</tr>
<tr>
<td>Acronym</td>
<td>Definition</td>
</tr>
<tr>
<td>--------</td>
<td>------------</td>
</tr>
<tr>
<td>SDMWN</td>
<td>Software-defined Multihop Wireless Networking</td>
</tr>
<tr>
<td>SDN</td>
<td>Software-defined Networking</td>
</tr>
<tr>
<td>TDCCCD</td>
<td>Topology Discovery Using Inter-controller And Controller-device Communications</td>
</tr>
<tr>
<td>TDCD</td>
<td>Topology Discovery Using Controller-device Communications</td>
</tr>
<tr>
<td>TDMA</td>
<td>Time-division Multiple Access</td>
</tr>
<tr>
<td>V2I</td>
<td>Vehicle-to-infrastructure</td>
</tr>
<tr>
<td>V2V</td>
<td>Vehicle To Vehicle</td>
</tr>
<tr>
<td>VANETs</td>
<td>Vehicular Ad-hoc Networks</td>
</tr>
<tr>
<td>WANs</td>
<td>Wide Area Networks</td>
</tr>
<tr>
<td>WMNs</td>
<td>Wireless Mesh Networks</td>
</tr>
<tr>
<td>WSNs</td>
<td>Wireless Sensor Networks</td>
</tr>
</tbody>
</table>
# List of Symbols

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>Action space</td>
</tr>
<tr>
<td>$R$</td>
<td>Reward</td>
</tr>
<tr>
<td>$S$</td>
<td>State space</td>
</tr>
<tr>
<td>$Cost^k_{TD}$</td>
<td>The cost of topology discovery running by a controller placed on device $k$ on all network devices in the network calculated using Equation (3.2) (TDCD model)</td>
</tr>
<tr>
<td>$Cost_{TD}$</td>
<td>The total cost of topology discovery running by all controllers on their own assigned network devices and among all controllers in the network calculated using Equation (3.12) (TDCCCD model)</td>
</tr>
<tr>
<td>$C'_{u,v}$</td>
<td>The required bandwidth of link $(u, v)$ to exchange control packets</td>
</tr>
<tr>
<td>$C_{RM}$</td>
<td>Replay memory capacity</td>
</tr>
<tr>
<td>$C_{u,v}$</td>
<td>The capacity of link $(u, v)$ to exchange control packets</td>
</tr>
<tr>
<td>$\varepsilon_{\text{decay}}$</td>
<td>Epsilon decay rate</td>
</tr>
<tr>
<td>$\varepsilon_{\text{min}}$</td>
<td>Minimum epsilon rate</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>Exploration rate</td>
</tr>
<tr>
<td>$f_{a,b}^{u,v}$</td>
<td>Binary Decision Variable: The value equals one if and only if link $(u, v)$ is used to provide communications between device $a$ and device $b$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Discount factor</td>
</tr>
<tr>
<td>$L^{\text{Int}}_{u,v}$</td>
<td>A set of links in the interference range of link $(u, v)$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Learning rate</td>
</tr>
<tr>
<td>$\text{minbatch}_{\text{size}}$</td>
<td>Minbatch size</td>
</tr>
<tr>
<td>$\text{minC}_{\text{RM}}$</td>
<td>Minimum replay memory size</td>
</tr>
<tr>
<td>$\text{neighbor}_{i,j}$</td>
<td>The $j^{th}$ neighbor of network device $i$</td>
</tr>
<tr>
<td>$\text{neighbor}[i]$</td>
<td>A set of neighbors of network device $i$ in the network</td>
</tr>
<tr>
<td>$N_{\text{hop}}_{m,n}$</td>
<td>The number of hops in the shortest path between device $m$ and device $n$</td>
</tr>
<tr>
<td>$N$</td>
<td>A given number of controllers</td>
</tr>
</tbody>
</table>
$P_c$  Crossover rate in the proposed GA approach  
$P_m$  Mutation rate in the proposed GA approach  
$PopSize$  The number of chromosomes in the population in the proposed GA approach  
$R_{Flow-Rq}$  The arrival rate of new flows in each device that triggers a flow request message toward the assigned controller  
$R_{TD}$  The rate of running topology discovery by each controller  
$V$  The set of wireless network devices  
$x_{k,i}$  Binary Decision Variable: The value equals one if and only if device $i$ is assigned to a controller placed on device $k$  
$y_k$  Binary Decision Variable: The value equals one if and only if there is a controller placed on device $k$
Chapter 1

Introduction

1.1 Background

In wireless networks, devices can communicate directly if they are within their respective transmission range. Otherwise, to extend the network coverage, two modes are defined: infrastructure-based and infrastructure-less. In the infrastructure-based mode, an infrastructure such as a base station or an access point coordinates the communications among devices, provides wireless access and enables communications through typically a fixed network core. Cellular networks use this mode to extend the coverage of the network. While, in the latter mode, devices must be able to communicate with each other without the help of any infrastructure. To achieve this goal, devices forward packets to a destination through intermediate devices in a multihop manner. Therefore, in such networks, devices can be a host or a relay [1].

Multihop wireless networks (MWNs) are infrastructure-less networks, in which devices can be stationary or mobile, and can communicate wirelessly with each other using one or more intermediate devices [2–4]. Such networks are self-configuring and self-organizing: devices are responsible for organizing the network and delivering messages in a distributed manner.

Since MWNs work without the need of any infrastructure, they have several advantages, including rapid deployment, flexibility, low cost, etc. We can distinguish four categories of MWNs as follows:

- Mobile ad-hoc networks (MANETs): A MANET is a MWN that consists of a number of heterogeneous mobile wireless devices such as smartphones, laptops, vehicles,
sensors [5]. These mobile devices forward packets using multihop communications.

- Vehicular ad-hoc networks (VANETs): A VANET is a subgroup of MANETs, in which mobile devices are vehicles with high mobility. In addition to the communications among vehicles, i.e., vehicle to vehicle (V2V), VANETs provide vehicle-to-infrastructure (V2I) on the roads. In these networks, V2V and V2I communications help to disseminate the latest information related to traffic jams such as speed, the vehicle directions, the traffic density, and accidents through the network [6, 7]. Distributing such information aids in managing the traffic, decreasing the number of accidents and saving people's lives. In addition, VANETs can provide connectivity to the Internet for drivers and passengers via the infrastructure component.

- Wireless sensor networks (WSNs): A WSN consists of a set of sensor devices that work together to sense the environment, process and forward data in a multihop manner to a base station or sink [8]. Further analyses of data will be performed at the sink or a server connected to the WSN through the sink. A WSN usually does not have any infrastructure and all devices communicate with each other via intermediate devices. In WSNs, sensor devices can be mobile or stationary.

- Wireless mesh networks (WMNs): In a WMN, devices consist of mesh clients and mesh routers. Mesh clients are wireless devices such as laptops, smartphones, which can join or leave the network at any time. Mesh routers provide connectivity for the mesh clients and help to forward packets in a multihop manner without any fixed/wired infrastructure [9]. In such networks, mesh clients can be stationary or mobile nodes, while mesh routers have, typically, at most minimal mobility.

In addition to several challenges that all wireless networks face, including unreliable wireless medium, interference, hidden and exposed terminals [10], MWNs have the following common challenges [1, 11-13]:

- Self-organizing and distributed management: MWNs are self-organized, and the management of the network is distributed. Therefore, devices are responsible for establishing the connectivity with each other. In addition, devices usually have only a partial, local view of the network topology. As a result, in these networks, without any central control management, most likely, it will be challenging to make globally optimal decisions to provide coordination and connectivity among devices.
• Device mobility: In MWNs, since devices can be mobile and move in an arbitrary fashion, the network topology may change dynamically. Therefore, these dynamic changes may lead to frequent disconnections and reconnections among the components of the network. Consequently, providing connectivity and communications among the devices while adjusting the network to the dynamic topology changes is quite challenging.

• Energy: In MWNs, devices consume energy for receiving, transmitting and processing data; however, they have limited battery power, and it is difficult to recharge or replace the battery. In addition, if a device loses its battery power, it will become unusable in the network, which affects the lifetime of the network. Moreover, without any central control, it is difficult to manage the energy consumption of the network to achieve the network-wide objectives.

• Quality of service (QoS): In addition to the unreliable wireless medium that leads to difficulties in terms of providing QoS, such as delivery guarantees, delay, there are some other factors in MWNs that affect the QoS. Because of the device mobility, the network topology will change frequently; as a result, compared to a static network, the network status will change dynamically. Therefore, it is challenging to offer QoS guarantees (hard or soft) based on the network status.

1.2 Problem Statement and Research Objective

Software-defined networking (SDN) [14] is a promising architecture that provides simplicity in network devices and flexibility in network management by separating the control plane and the data plane of the network. Decoupling the control plane and the data plane of the network simplifies the network devices, which now act only as simple forwarding elements. The control plane, removed from the forwarding devices, resides in a logically centralized controller, which is responsible for managing and controlling the network. An SDN controller is capable to program network devices based on the requirements of the application plane on the top of the controller. The application plane specifies the policies and the requirements of the network. Therefore, an SDN controller can bring programmability and flexibility to communication networks, and simplify network management and enable network innovations [1, 15]. In an SDN architecture, the controller and network devices can exchange control packets using a southbound application-programming interface (API) such as OpenFlow [16] to update the global view of the network and receive forwarding paths.
SDN was first proposed for wired networks. However, various studies have been conducted to apply SDN to wireless networks. These works demonstrate the potential of SDN to overcome the existing challenges of some wireless networks, including resource allocation, interference management, heterogeneous management, scalability [17, 18].

SDN has the potential to address the MWN-specific challenges outlined earlier [19, 20]. Software-defined multihop wireless networking (SDMWN) applies the SDN concept to MWNs, in which a logically centralized controller collects information related to links and wireless network devices from the network periodically and creates a global view. This global view of SDMWN consists of the connectivity graph among devices. In the traditionally distributed management of MWNs, network devices are responsible for establishing the connectivity with each other using local, partial views of the network. While, in SDMWN, using the periodic updates of the network state, the controller is able to program the network devices, which is useful to optimize routing decisions globally and adjust to dynamic topology changes faster.

The main idea of SDN is to control the entire network by a logically centralized controller that has a global view of the network. This logically centralized controller may be implemented as a physically centralized or distributed control plane. In a physically centralized control plane, only one controller manages and controls the entire network and data plane devices. On the other hand, in a physically distributed control plane, the network is divided into different domains and several distributed controllers are responsible for managing the domains. Each type of the control plane architecture has its own pros and cons.

Since implementing a physically centralized control plane is simple and satisfies the idea of SDN, which is a logically centralized control management, in most studies on SDMWN, a centralized control plane architecture is proposed to show the benefits of applying SDN to MWNs [21–29]. However, this scheme has some drawbacks such as introducing a single point of failure, scaling limitations, energy depletion issue [30].

A distributed control plane is a solution to overcome the challenges of a single controller. Chapter 2 reviews the state-of-the-art in applying SDN with a distributed control plane to MWNs. Although a distributed control plane addresses reliability, scalability, energy depletion issues [31], it raises some new challenges that should be addressed.

One of the challenges in proposing a distributed control plane is how to achieve the global view of the network and synchronize different views to provide inter-domain communications among devices. Depending on the type of inter-controller communications to achieve the global view of the network, in Chapter 2, the related work in SDMWN with a distributed control plane is categorized into two different groups: Flat and Hierarchi-
cal [30–32]. In a flat architecture, all controllers exchange their local views of the network to provide inter-domain communications and achieve a global view of the network that can be stored in all or some of the controllers. On the other hand, in a hierarchical model, the network is managed by different levels of controllers, in which each local controller exchanges its local view only with the upper layer controllers. Moreover, at the highest level, a main controller collects all information about the network to obtain the global view of the network and manage the network in a centralized way.

In SDMWN, in addition to the data packets exchanged among network devices, control packets need to be exchanged periodically among a controller and network devices (controller-device) to discover the network topology, exchange configurations and set up flow tables. Moreover, in a multi-controller environment, a number of control packets is required to be exchanged periodically among controllers (inter-controller) to synchronize and integrate different network views and obtain a global view of the network to provide inter-domain communications. Depending on the network characteristics, proposing a distributed control plane can be costly in terms of control overhead injected to the network. In this thesis, the terms control overhead in the network and the network cost are used interchangeably.

In some SDMWNs, such as WSNs, because of their unique characteristics, there is only one interface to forward both control and data traffic. Therefore, in such networks, the capacity of links is used to exchange both data and control traffic. Moreover, because of the wireless nature of links, interference can influence the available capacity of links [33,34]. Therefore, achieving the global view of the network using unreliable and shared wireless communications, and the capacity limit of links can be quite challenging. In SDMWN, minimizing the control packets flowing over links improves scalability and reduces energy consumption and packet losses, which also has an impact on the reliability of the control plane. However, most studies in SDMWN do not provide any details about the existing challenges to achieve the global view of the network.

Another challenge in a distributed control plane is to determine the number of controllers, controller placements and controller assignments to network devices. Heller et al. [35] refer to the mentioned challenges in a distributed control plane as the controller placement problem (CPP) that is a non-deterministic polynomial-time (NP)-hard problem. The main objective of the CPP is to find the number of controllers, their placements and controller assignments to network devices with the aim of considering different metrics and objectives. Although various studies have been conducted so far to solve the CPP in wired networks, only a few studies consider the impact of solving the CPP in wireless networks on network performance [36–40]. This problem is introduced as the wireless CPP in [41], in which communication links among controllers and network devices are wireless. Address-
ing these challenges, especially in SDMWN with shared and unreliable communications, has a significant impact on network performance metrics such as latency, cost, reliability and control overhead.

Solving the CPP in a network has a direct effect on the control overhead injected to the network [39]. Chapter 2 reviews the related work in solving the CPP in wired networks while minimizing the generated control overhead in the network [42–46]. Moreover, Chapter 2 presents an overview of the literature in the wireless CPP [36–41]. However, to the best of our knowledge, in the related work in solving the CPP, all communications use the shortest paths, but the capacity of links and the impact of interference on the capacity of links, which is crucial in SDMWN is not considered. Therefore, solving the CPP with the objective of minimizing the network cost, considering the unique characteristics of SDMWN and the capacity limit of links is still an open research opportunity.

In addition, to the best of our knowledge, there are still the following research opportunities in solving the CPP in SDMWN. It is still an open research area to solve the CPP in SDMWN with mobile and stationary devices to minimize the network cost and demonstrate the effect of device mobility on solving the CPP. Moreover, in this case, considering the capacity limit of links and the impact of interference on the capacity of links to select routes among devices and controllers can be future research directions in solving the CPP in SDMWN.

1.3 Contributions

In this thesis, the objective is to solve the CPP in SDMWN while minimizing the network cost, i.e., the number of required control packets to be exchanged in the control plane. These control packets consist of controller-device control packets to discover the network topology, exchange configurations and set up flow tables in network devices as well as inter-controller control packets to synchronize different network views and obtain a global view of the network. In this thesis, it is assumed that the SDMWN architecture is SDN-based ad hoc networks which are infrastructure-less networks.

The novelty of this work is considering the characteristics of SDMWN, the capacity of wireless links and the effect of interference on the capacity of links to find the number of controllers, controller placements and controller assignments in the network.

In Chapter 3, the CPP in SDMWN is formulated into two different optimization problems, topology discovery using controller-device communications (TDCD) and topology discovery using inter-controller and controller-device communications (TDCCCD), to find
the placements of a given number of controllers and their assignments to the network devices. The objective of the proposed optimization problems is to minimize the number of required control packets to be exchanged in SDMWN with an in-band architecture. In an in-band architecture, the control and data traffic share the same band or channel. In addition, to solve the problems, the capacity limit of links to exchange control packets and the effect of interference on the capacity of links is considered to find controller placements and controller assignments. Both proposed optimization problems are nonlinear problems which are computationally intensive. Therefore, in Chapter 3, the CPP is only investigated with a small network to show the impact of the number of controllers, the arrival rate of new traffic flows, and the capacity of links on control overhead, the average number of controller-device hops and the average number of inter-controller hops.

In Chapter 4, due to the high computational complexity of the proposed optimization models, the TDCCCD model, which reduces the network cost compared to the TDCD model, is selected and simplified to a linear problem, in which the capacity of links is not considered as a constraint in the network. Then, a genetic algorithm (GA) [47, 48] based approach to solve the simplified TDCCCD model is proposed to minimize the network cost and find the controller placements and assignments in SDMWN. The results demonstrate that, the proposed approach is able to find solutions faster compared to the TDCCCD model and it reduces the network cost compared to the random placement approach.

In Chapter 5, the existing heuristic algorithms in the literature [40, 43–45] are extended to find near-optimal solutions for the simplified TDCCCD model. The results shows that, using ranking degrees assigned to the possible controller placements, including the average distance to other devices as a degree or the connectivity degree of each placement, the extended heuristic algorithms are able to achieve the optimal solutions in small-scale networks in terms of the generated control overhead and the number of controllers selected in the network. Moreover, these algorithms are able to reduce the control overhead in large-scale networks and find near-optimal solutions in large-scale networks faster.

Finally, in Chapter 6, the potential of using deep reinforcement learning (Deep RL) [49–52] in solving the CPP in SDMWN is investigated. Deep RL becomes a popular approach to the different problem ares, which uses the neural networks to train an agent to make predictions for the new data sets. Therefore, in Chapter 6, a deep Q-network (DQN) algorithm is proposed to the simplified TDCCCD model. In addition, a preliminary mobility scenario is considered to show the impact of using Deep RL in SDMWN with devices with minimal mobility. The results demonstrate that the proposed DQN algorithm can find optimal and near-optimal solutions in a small-scale networks for the CPP in SDMWN. Moreover, the results show the effect of increasing number of devices in the network on the computational time of running the proposed DQN algorithm.
1.4 Organization

The organization of this research proposal is as follows. Chapter 2, first overviews the state-of-the-art in applying a distributed control plane to SDMWN and existing challenges. After that, Chapter 2 reviews the related work in solving the CPP in wired networks to minimize the network cost and the related work in solving the CPP in wireless network while considering different objectives and constraints. Also, the existing heuristic algorithms and the proposed approaches based on RL and Deep RL in the literature are presented in Chapter 2. Then, Chapter 2 summarizes the existing challenges and open research areas. Chapter 3 introduces two different optimization models to minimize the network cost while considering the characteristics of SDMWN and related constraints. To solve the problem in a reasonable time, Chapter 4 presents a GA-based heuristic approach to the CPP in SDMWN and compares the achieved results with the optimal solutions. Chapter 5 extends the proposed heuristic approaches in the related work and compares the results of the algorithms with the optimal solutions. Chapter 6 investigates using Deep RL in solving the CPP in SDMWN and finally, Chapter 7 concludes the thesis and outlines the future research directions.

1.5 Publications

To date, the following papers are a part of the contributions of the thesis.

- **Published Journal Papers**

- **Submitted Journal Paper**
• **Accepted Conference Paper**

• **Submitted Conference Paper**
Chapter 2

Literature Review

2.1 Introduction

As mentioned earlier in Chapter 1, SDN has the potential to address the MWN-specific challenges in terms of network management, device mobility, energy consumption, quality of service, etc. Several studies have demonstrated that applying SDN to MWNs can be beneficial [18,53–55]. In [1], various studies are reviewed that show the benefits of applying SDN to MWNs, in which the control logic of wireless devices is logically centralized in an SDN controller that programs the whole network.

In most studies on SDMWN, a physically centralized controller manages the network to satisfy the requirements of different types of applications such as routing [21,22], scheduling [23,24], task allocation [25,26], load-balancing [27,28] and congestion control [29]. In this context, the results show that these approaches, compared to the traditional distributed management, can be beneficial in terms of network performance metrics, including packet delivery ratio and delay [1].

However, in addition to the aforementioned drawbacks of a centralized control plane in Chapter 1, such as a single point of failure, scaling limitations [30], applying a physically centralized control plane to SDMWN has some disadvantages. In this case, due to the mobility of network devices and scalability issue, direct connections among network devices and a centralized controller are not reasonable or practical for some scenarios, e.g., constrained resources for network devices. Therefore, network devices need to communicate with a controller in a multihop manner using an unreliable and shared wireless medium to update the general view of the network and receive flow rules. Consequently, network
devices may face higher latency, especially devices farther away from the controller, which leads to higher flow setup time [56].

In general, in an SDN architecture with a distributed control plane, the network is divided into different domains and the management of the domains is distributed among multiple controllers. Therefore, controllers are able to control devices that are closer geographically which helps to reduce latency and reacts to topology changes faster. Moreover, in case of a controller failure, other controllers can manage the network. Section 2.2 reviews various studies on applying a distributed control plane to SDMWN.

Although proposing a distributed control plane in an SDN architecture can be beneficial to overcome the shortcomings of a centralized control plane, it raises several new challenges. One of the challenges is how to achieve the global view of the network. In an SDN architecture with a distributed control plane, two types of control packets are required to be exchanged in the network: controller-device and inter-controller control packets. Controller-device control packets are exchanged to update the states of SDN-based devices and the network view in each controller, and exchange the configurations and set up flow rules in network devices. Moreover, in such networks, inter-controller communications inject control packets to the network to synchronize the network views of different controllers, which aims to provide inter-domain communications among network devices [57]. In some types of SDMWN such as WSNs, because of their characteristics, there is only one interface to exchange both data and control traffic. Therefore, the capacity of wireless links is shared between data and control traffic. In this case, when we limit the available capacity of links to exchange control packets in the network, providing controller-device and inter-controller communications to achieve the global view of the network using wireless mediums can be quite challenging and should be addressed. Moreover, proposing a distributed control plane can be costly in terms of control overhead injected to the network compared to a centralized control plane.

In Chapter 3, two different approaches are considered to discover the network topology and achieve the global view of the network in all controllers in SDMWN. Then, the generated control overhead in both models are compared in case of having centralized and distributed control planes.

Another challenge of a distributed control plane is determining the number of controllers, their locations and how to assign controllers to network devices that impact on network performance metrics such as latency, cost, reliability [58]. Heller et al. in [35] refer to the mentioned challenges of a distributed control plane as the controller placement problem (CPP) which is a NP-hard problem. Several studies have been conducted so far to consider various metrics and multiple objectives such as minimizing latency among
controllers and network devices [35], load-balancing among controllers [59], minimizing the cost of the control plane deployment [60,61], improving reliability [62,63], to solve the CPP in wired networks. However, only a few studies consider the impact of solving the CPP in wireless networks on network performance, considering the characteristics of wireless communications. Therefore, it is important to address the challenges raised by proposing a distributed control plane in SDMWN based on its unique characteristics and constraints.

As stated in Section 1.3, the objective of the thesis is to solve the CPP in SDMWN while minimizing the number of required control packets to be exchanged to discover the states of network devices and the network topology, and to exchange configurations and set up flow rules in network devices. Moreover, the capacity of links and the impact of interference is considered on solving the CPP in SDMWN. In addition, the average number of hops in controller-device and inter-controller communications are investigated to evaluate the performance of the proposed models.

Therefore, this chapter first reviews various studies on SDMWN with a distributed control plane and their shortcomings in Section 2.2. Section 2.3 describes the related work in solving the CPP in wired networks to minimize the control overhead in the network and discusses the existing challenges. Section 2.4 presents an overview of the related work in solving the CPP in wireless networks. Various heuristic algorithms have been proposed to solve the CPP in large-scale networks. Section 2.5 reviews the related work on the heuristic algorithms proposed to solve the CPP. Recently, deep learning and reinforcement learning concepts [49–52] have received a significant attention to solve various optimization problems. A few studies use these concepts to find the optimal solutions in solving the CPP in wired network, which are presented in Section 2.6. Finally, Section 2.7 summarizes the existing challenges and open research areas in solving the CPP in SDMWN to minimize control overhead.

2.2 Distributed Control Plane in SDMWN

We can categorize the state-of-the-art in applying a distributed control plane to SDMWN into two different architectures depending on the type of inter-controller communications to achieve the global view of the network: Flat (horizontal) and Hierarchical. However, most studies do not provide any details about inter-controller communications using the wireless mediums to achieve the global view of the network in SDMWN.
2.2.1 Flat Architecture

In a flat architecture, each controller manages a domain. To provide inter-domain communications in the data plane, all controllers exchange information about the local/partial view of the network to obtain the global view. This global view can be stored in all or some controllers [30–32]. HyperFlow [64] and Onix [65] have a flat control plane, in which all controllers are at the same level and communicate together to achieve the global view. A few studies on SDMWN consider a flat distributed control plane to show the advantages of using multiple controllers in SDMWN [66–68].

In SDCSN [66], dSDiVN [67] and RDSNET [68], the network is divided into different domains. In each domain, a device is defined as a head of the domain to host a controller. dSDiVN uses the clustering algorithm defined in [69, 70] to place local mobile controllers, which helps to decrease latency among vehicles and their assigned controllers. In RDSNET, the network is clustered into three domains statically. In SDCSN, its authors do not provide any details about the number of controllers as well as simulation results to show the advantages of the proposed architecture.

SDCSN and RDSNET define gateway devices in each domain to provide inter-domain communications. On the other hand, in dSDiVN, to obtain a global view of the network, all local controllers exchange their local views of the network. In RDSNET, there is an assumption that each controller installs the required flow rules on its own controlled network devices at the beginning of the simulation to avoid overhead and latency.

dSDiVN and RDSNET provide a recovery approach, in which backup controllers are used in case of failure. In dSDiVN, each local controller has a list of backup controllers and then selects the best backup controller to exchange its own view of the network periodically. Moreover, each controller informs the vehicles in its own domain about the selected backup controller via installing flow rules. Therefore, in case of failure, vehicles send flow requests to the selected backup controller which now acts as the main controller in that domain. In RDSNET, to manage a controller failure, a backup controller is considered to act as a main controller in case of failure. However, in dSDiVN and RDSNET, the authors do not discuss how to select backup controllers.

dSDiVN is simulated in NS3 [71] and the results show that when a controller failure happens, after a short time of dropping packets, packet delivery will resume compared to a non-SDN architecture. However, it shows that increasing the number of hops between a controller and vehicles leads to higher flow setup time which requires to choose the best controller for each vehicle. In RDSNET, simulated in OMNeT++ [72], the results demonstrate that, compared to a centralized architecture with increasing the number of
devices, network devices experience higher total number of received messages and lower latency.

In general, in a flat architecture, since the network is controlled in a distributed manner, it is challenging to achieve a level of consistency among controllers to share their local views. In addition, it is challenging to provide inter-domain communications, especially with increasing the number of network devices and controllers. Therefore, it may lead to wasting network resources to update the network views in different controllers [73].

2.2.2 Hierarchical Architecture

In a hierarchical architecture, the control plane consists of different levels of controllers. In general, at the lowest layer, there are some local controllers, each controlling a domain and exchanging information with the upper layer controllers. In addition, at the highest level, there is a main controller referred to as a root that has a global view of the network via exchanging information with local controllers periodically [30–32]. Kandoo [74] and [75] have hierarchical control plane architectures, in which there are different layers of controllers. In these architectures, in case of inter-domain communications, local controllers communicate with the upper layer controllers, which helps to reduce the interaction load among controllers in different levels.

In [76–78], a two-layer hierarchical control plane for SDMWN is considered, in which the network is divided into different domains and each domain is managed by a local controller that exchanges the local/partial view of the network with a root controller in the higher layer. In [76], the architecture is simulated in Cooja [79] and local controllers are placed on devices defined as a head of each domain. The results demonstrate that, compared to a centralized control plane, the proposed architecture reduces the time to initiate communications with the controller and installing flow rules. In DeVANET [77], a root controller is responsible for controlling the network, handling failures in local controllers and inter-domain communications. DeVANET, simulated in OMNeT++ [72] and the SUMO simulator [80], improves network performance compared to a non-SDN architecture in terms of scalability. However, in [76,77], the authors do not discuss the clustering methods and how to determine the number of controllers in the lower layer. The D-SDWMN architecture is proposed in [78], in which, in addition to a root controller, there are two backup controllers in the higher layer. The root controller, based on the received information from the network, selects some network devices to place local controllers. Then local controllers request their neighbors to join their own domains. Moreover, in case of any changes in the network, new network devices can join a controller or a controller can go down. In [78], the
authors do not provide any simulation results or any details about selecting the network devices for placing controllers.

In general, in a hierarchical architecture, compared to a flat architecture, interaction load among controllers in different levels is reduced. However, with increasing the number of devices in the network, there are some challenges that should be considered, including determining the number of layers and defining the responsibilities of the controllers in different layers.

2.2.3 Proposed Control Plane Architectures in this Thesis

Chapter 3 presents two approaches to discovery the network topology and achieving the global view of the network in SDMWN. Both approaches consider that each controller obtains the global view of the network. To achieve the global view of the network in the first approach, each controller runs topology discovery on all network devices. While, the second approach uses a flat architecture, in which each controller has a local view of its own domain. Therefore, to achieve the global view of the network in the second approach, all controllers communicate together in an interval of time. Using inter-controller communications in the second approach, the global view of the network is stored in all controllers.

Both proposed approaches in Chapter 3 inject more control overhead to the network to discover the network topology, compared to a hierarchical architecture with a root controller. However, in a hierarchical architecture, controllers need to communicate with the upper layer controllers to provide inter-domain communications. Therefore, when increasing the number of devices in the network, these communications generate more control overhead to set up flow tables and exchange the required configurations.

2.3 Controller Placement Problem in Wired Networks

As mentioned earlier, control overhead injected to a network with an SDN architecture consists of control packets exchanged among network devices and their assigned controllers to discover the states of network devices and network topology, and to exchange configurations and set up flow rules in devices. In addition, control packets are required to be exchanged among controllers to synchronize different network views and achieve the global view of the network.
In this thesis, the terms the generated control overhead, which consists of the total number of required control packets to be exchanged in the network and the network cost are used interchangeably.

Determining the number of controllers, controller placements and controller assignments has an impact on the number of control packets exchanged in the network [39]. This section reviews some papers that propose an approach to solving the CPP in wired networks while minimizing control overhead in the network.

One of the first papers that takes minimizing control overhead in solving the CPP into consideration is [42]. In [42], its authors propose a dynamic approach to adjust the number of controllers and placements in large-scale wide area networks (WANs), which aims to minimize flow setup time as well as control overhead in the network. In the proposed approach in [42], the control overhead consists of the number of control packets exchanged to provide controller-device and inter-controller communications (flat architecture), considering the number of hops in the shortest paths among controllers and devices. In [42], the proposed approach is called dynamic since in addition to the controller-device and inter-controller communication costs, the authors consider a cost of controller-device reassignment so that a heuristic algorithm checks the traffic patterns in the network periodically and decides whether to change the controller assignments or not. In the proposed model in [42], the traffic patterns result in different rates of flow requests in devices. The authors in [42] formulate the problem as an integer linear program (ILP), which is minimizing a weighted sum of the mentioned control overheads in the network. Moreover, they define the delay among network devices and their assigned controllers as a constraint with an upper bound. To solve the problem, two heuristic algorithms are proposed: a greedy knapsack (DCP-GK) and a simulated annealing-based approach (DCP-SA). The results demonstrate that DCP-SA and DCP-GK are able to achieve a tradeoff between minimizing flow setup time as a constraint and minimizing the control overhead as an objective function compared to having one controller in the network that increases flow setup time or placing a controller on each device that leads to higher control overhead. Moreover, the results show that DCP-SA outperforms DCP-GK in terms of flow setup time and control overhead in the network; however, DCP-GK is faster. In [42], the authors do not provide any performance evaluations to compare the results obtained from the heuristic algorithms and the proposed optimization model. Moreover, in the proposed optimization model, the cost of reassignment is defined as the cost of communications among devices and new controllers using the shortest paths. However, the controller reassignment has an impact on the total control overhead in the network that should be considered in calculating the network cost.

The objective of [43] is to find the optimal controller placements and a minimum spanning tree among controllers to provide inter-controller communications with the objec-
tive of minimizing the control overhead in large-scale WANs, considering the number of hops. In [43], control overhead consists of the amount of data (byte/second) generated by controller-device communications to collect information from devices and to set up flow rules as well as the amount of data generated by synchronization among controllers (flat architecture) using the minimum spanning tree. Moreover, the authors define the delay among devices and controllers as a constraint. The problem is formulated as a mixed integer program (MIP) and a greedy heuristic approach is proposed to find controller placements and a minimum spanning tree among controllers. The results show that the proposed greedy heuristic approach is able to find solutions in a reasonable time while the deviation between the optimal solutions obtained using the CPLEX [81] and the solutions from the proposed approach is less than 3 percent. In addition, the results demonstrate the impact of the cost of topology discovery, flow rule setup and synchronization among controllers on the number of controllers. In the network with higher cost of topology discovery and flow rule setup, more controllers are needed, while when increasing the cost of synchronization among controllers, fewer controllers are selected.

Yao et al. in [44] propose a model to find the best controller placements in both centralized and distributed control plane architectures while minimizing the total cost of flow rule setup in the network. In [44], the cost of flow rule setup is considered only as the cost of routing (control overhead) from all devices to their assigned controllers along the shortest path, considering the number of neighbors as the degree of a device. Then, all devices with degrees higher than the average degree of all devices are selected to be possible controller placements. The authors, first, propose an optimization model to solve the problem in a centralized control plane architecture and find a controller placement. After that, to solve the problem in a distributed control plane environment, the authors estimate the number of domains by dividing the total degrees of all devices to the capacity of the controllers that defines the maximum allowed flow requests to be assigned to a controller in the network. Then, using the aforementioned optimization problem, the best controller placement in each domain is selected. Moreover, in a distributed control plane, an algorithm is proposed to check the load of different controllers periodically and in case of controller overloading, one of the neighbor controllers is selected to manage the devices, which aims to satisfy the load of controllers as a constraint and minimize the cost of flow rule setup as the objective. To demonstrate the performance of the proposed model, the authors only provide results in a centralized control plane architecture to show that the proposed model is able to find a controller placement with lower cost of flow rule setup compared to the model proposed in [35]. In addition, the results demonstrate that in case of controller overloading, the proposed algorithm is able to reduce the load of the controller and assign the devices to another controller. However, the authors do not provide
any results to show the performance of the proposed model in a distributed control plane architecture.

A measurement-aware distributed controller placement problem (MDCP) is proposed in [45]. The objective of MDCP is to find the number of controllers and controller placements while minimizing control overhead injected to the network by inter-controller communications to synchronize different network views and controller-device communications to set up flow rules, considering the number of hops in the shortest path communications. In MDCP, the problem is formulated as a quadratic integer programming problem. To solve the proposed optimization problem, the authors propose two different heuristic algorithms: a discrete approximation algorithm (DAA) and a connectivity ranking algorithm (CRA). The DAA, first, transforms the optimization problem to a facility location problem [82] and estimates the synchronization cost for each controller placement. Then, the approximation algorithm finds the number of controllers and controller placements in the network. On the other hand, in CRA, a ranking algorithm is proposed based on the connectivity degrees of the possible placements (number of neighbors). The possible controller placements with higher ranking degree (higher number of neighbors) are added to the set of controllers in the network until adding a new placement increases the current control overhead in the network. The results demonstrate that CRA and DAA are able to solve the problem in small-size networks, less than 30 devices, with slightly higher control overhead compared to the optimal results. Moreover, these two algorithms are able to solve the problem with less control overhead compared to a random controller placement model. Compared to DAA, CRA is faster and selects fewer controllers in the network. In addition, the results show that the number of controllers selected in a network depends on the network size and the network topology. In [45], the authors do not consider the control overhead generated by controller-device communications to discover the network topology.

In [46], the authors consider three different objectives to solve the CPP, including minimizing control overhead among network devices and their assigned controllers (SCA), minimizing control overhead among controllers (CCA) (flat architecture) as well as minimizing delay in these communications, and finding a fair optimal tradeoff between CCA and SCA overhead (FTCS) in large-scale networks. The authors use linear programming to formulate the problem and use a bargaining game [83] to find a pareto optimal solution between CCA and SCA control overheads. Moreover, they define two upper bounds for the tolerated CCA control overhead defined as a constraint in the first mentioned objective and the tolerated SCA control overhead defined as a constraint in the second mentioned objective. In addition, they consider load-balancing among controllers to find the solution. The results demonstrate that FTCS can achieve better results in terms of total control overhead compared to SCA and CCA. Moreover, the evaluation results in [46] show that increas-
ing the number of devices generates higher control overhead in the network. However, in [46], the authors do not provide any results to show the impact of the proposed model on delay, load-balancing among controllers and the number of controllers in the network. Moreover, the authors do not consider the control overhead generated by controller-device communications to discover the network topology.

**Summary:** In the reviewed related work in this section [42–46], control overhead injected to the network is only considered as control overhead generated by devices to send the flow rule requests toward their assigned controllers along shortest paths. However, in an SDN architecture, control overhead exchanged to set up configurations and flow rules in network devices consist of control packets sent by devices to controllers as well as control packets sent by controllers to devices, considering the rate of flow rule requests in devices. In this case, a device and its assigned controllers do not necessarily use the same path in both directions to communicate with each other and these communications may use different paths with different number of hops. In all reviewed studies, controller-device and inter-controller communications use the shortest paths to communicate, while communications along the shortest paths may not be practical in real networks because of the limited capacity of links and traffic management.

### 2.4 Controller Placement Problem in Wireless Networks

In the studies reviewed in Section 2.3, controllers and network devices are connected via wired links. Abdel-Rahman et al. [41] introduce the wireless CPP, in which communication links among controllers and network devices are wireless. For wireless networks, in addition to the metrics addressed in the wired CPP, the characteristics of unreliable and shared wireless mediums should be considered in solving the wireless CPP [36], which plays a critical role on connectivity among controllers and network devices.

The objective of the proposed model in [41] is to minimize the number of controllers and find optimal controller placements and assignments in a network. In [41], first, the authors formulate the controller placement and assignment problem in wired networks as two different models, a mixed-integer linear program (MILP) and an ILP. The difference between these two models is in considering different constraints: average response time and per-link response time of a controller for its assigned network devices, respectively. In these models, network devices use time-division multiple access (TDMA) slots to access the network and communicate with the controller. Then, the authors extend the model to
a wireless network as a chance-constrained stochastic program (CCSP) with considering wireless communications, in which the total delay consists of the network access delay for the devices, transmission delay, propagation delay and the queuing delay at the controller. The results demonstrate that the first model is able to reduce the number of selected controllers in the network compared to the model proposed in [84]. Moreover, the results show the impact of the defined upper bound value for the average response time of the controller and the value of the defined time slots in communications among devices and their assigned controllers on the number of selected controllers. In this case, when increasing the value of the upper bound value, fewer controllers are required in the network, while when increasing the value of the time slots, more controllers will be selected in the network.

Dvir et al. in [37] and [36] propose multi-objective optimization problems to solve the wireless CPP, in which inter-controller communications and communications among controllers and access points are wireless. In [37] and [36], the objective is to cluster the network into a given number of domains using the K-median algorithm [85] and find controller placements. Then, the access points in each domain are assigned to a selected controller using one-hop wireless links. In both papers, network devices use carrier sense multiple access (CSMA) to access the network. In [37], the wireless CPP is formulated so that the objective is to minimize the weighted sum of propagation latency and link failure probability while considering throughput and a new metric called transparency as constraints. Transparency is defined as the latency in the data plane, which is caused by interference added by the proposed control plane. In [37], the results show that the number of selected controllers has an impact on the transparency in the network. Dvir et al. extend this work in [36], in which transparency is considered as a part of the objective function and the capacity of controllers is defined as a constraint. When increasing the number of devices in the network, the K-median algorithm is slow to solve the CPP. Therefore, in [36], the authors propose an extension algorithm called EWCPP. By checking a combination of all controller placements, the EWCPP is able to find the number of controllers to minimize the objective function and satisfy the constraints. Moreover, they propose a simulated annealing-based approach called SAWCPP to solve the problem. The evaluation results show that EWCPP is able to find fewer controllers compared to SAWCPP but it takes longer to find the results using EWCPP. Moreover, the results demonstrate that with increasing the number of devices in the network, link failure probability and latency obtained from both algorithms increase.

The objective of [38] is to solve the wireless CPP in a VANET with a two-layer hierarchical control plane to minimize latency and the number of controllers in the network. The authors in [38] formulate the problem as a modified facility location problem [86] to find some road side units (RSUs) to host local controllers, while higher layer controllers
are distributed on the Internet. In this case, vehicles are able to communicate with local controllers directly or via other RSUs in the network. Therefore, communications among controllers and network devices consists of vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications via IEEE 802.11p standard, and RSU-to-RSU communications via Ethernet. They consider different metrics including the locations of RSUs, the load of controllers, the locations of vehicles, to minimize latency that consists of transmission latency, queuing latency, contention latency, processing and propagation latency on a single hop. In this case, V2V and V2I use carrier sense multiple access with collision avoidance (CSMA/CA) and RSU-to-RSU communications uses carrier sense multiple access with collision detection (CSMA/CD) to calculate the contention delay. The authors use the CPLEX [81] to solve the problem and NS3 [71] to measure network performance. The results show that, compared to a random placement, using the proposed approach in [38], the network experiences lower latency. Moreover, the proposed approach improves network performance in terms of latency and packet delivery ratio compared to OLSR [87] and AODV [88]. However, the authors do not provide any results to show the impact of the proposed approach on the number of controllers in the network.

Qin et al. in [39], propose an optimization model to find controller placements in wireless edge networks while achieving a tradeoff between minimizing delay among devices and their assigned controllers, and minimizing control overhead (Mbps) in the network. Control overhead in [39] consists of controller-device communications to set up flow rules, and inter-domain communications to discover the network topology along the shortest paths. The authors formulate the problem as a nonlinear programming problem and then they simplify the model to a MIP problem. To calculate the control overhead injected to the network by synchronization among controllers, the authors use two different models: leader-less, in which all controllers communicate together to achieve the global view of the network and leader-based, in which all controllers only exchange information with a lead controller. To find controller placements in large-scale networks, the authors in [39] propose a randomized greedy algorithm. The results show that the proposed algorithm is able to find near-optimal solutions and outperforms the improved version of the MDCP algorithm proposed in [45] in terms of minimizing delay and control overhead. However, in [39], the authors do not consider the controller-device communication control overhead to discover the network topology and the characteristics of wireless medium to solve the problem.

In [40], a greedy algorithm is proposed to find a given number of controller placements in wireless mesh networks (WMNs). In the proposed algorithm in [40], each possible controller placement is ranked based on the average delay to all other devices and a value that shows the average number of links in the interference range of the links between
the possible controller placement and devices in the network. The placement with the smallest ranking will be selected to place a controller. The evaluation results show that the proposed algorithm is able to minimize the average and maximum delay among devices and their assigned controllers in WMNs. The objective of the paper is to find only the controller placements and the process of the controller assignment is not defined in the proposed algorithm. Moreover, the authors do not provide any evaluation results to show the impact of interference on the selected controller placement.

In [89], a mobility-aware approach is proposed to solve the CPP in VANETs dynamically while minimizing the flow setup delay. The authors formulated the problem as an ILP and they proposed a simulated annealing-based algorithm to find near-optimal solutions for the problem. Based on the mobility prediction of vehicles in the network, the proposed approach selects RSUs dynamically to place a controller while minimizing the inter-controller communication overhead to synchronize different views in the control plane. The results show that the proposed approach is able to reduce the flow setup delay as well as the exchanged control overhead among controllers to synchronize the views.

**Summary:** The objective of most studies in solving the CPP in both wired and wireless networks is to minimize propagation delay among controllers and devices that is proportional to the distance among them. Minimizing the distance among controllers and devices as well as among controllers has an impact on the reliability of the control plane, especially in wireless networks with shared and unreliable communications.

To the best of our knowledge, in the related work of solving the CPP in wireless networks, to minimize propagation delay, the capacity of wireless links and the impact of interference on the capacity of links are not considered when determining the number of controllers and controller placements.

### 2.5 Heuristic Approaches to the CPP

Based on the reviewed studies on solving the CPP in wired and wireless networks in the literature, formulating the CPP as an optimization problem helps to find optimal controller placements and assignments in the network while satisfying different objectives and constraints. However, when increasing the number of devices and possible controller placements in the network, it takes a long time to solve the optimization problem and find the optimal solutions. Therefore, various heuristic algorithms have been proposed to solve the CPP and find near-optimal solutions in a reasonable time while considering different objectives in large-scale networks [58]. To propose a heuristic algorithm to solve
the CPP, depending on the objectives of the model, different metrics can be considered to find controller placements in the network [90], such as the number of hops or delay among controllers and their assigned devices [43], delay among controllers [57], the capacity of controllers [42] and link capacities.

As mentioned earlier, in Chapter 3, the objective is to formulate the CPP in SDMWN as an optimization problem while minimizing the control overhead in the network, referred to as the network cost, considering the characteristics of SDMWN. However, the results demonstrate that when increasing the number of devices in the network, due to the high computational complexity, the solving time increases significantly. Therefore, to solve the problem in a reasonable time, the objective of Chapter 4 and Chapter 5 is to extend some of the proposed heuristic algorithms in the literature to minimize control overhead in SDMWN and satisfy different constraints. Therefore, this section discusses the heuristic algorithms proposed in the literature of the CPP in wired networks that aim to minimize the control overhead based on different metrics of the possible controller placements and their shortcomings to find the number of controllers, controller placements and controller assignments in the network. In addition, several research efforts have been conducted on GA-based heuristic approaches to the CPP with different objectives and constraints to show the benefits of the GA in solving the CPP that will be reviewed in this section.

Capacity of the Possible Controller Placements

DCP-GK [42] is a heuristic algorithm based on the greedy knapsack approach to minimize the control overhead and flow setup time in large-scale WANs. In DCP-GK, to find the controller placements, in each iteration, a placement is selected with the minimum sum of route costs to all other devices and with the delay less than a pre-configured threshold to all other devices. Then, the greedy knapsack algorithm assigns the selected controller to the devices that are not assigned to any controllers in the network. The controller assignment is based on the capacity of the controllers defined as the knapsack, the rate of flow requests in devices is defined as the weight of each item and the cost of the paths among the selected controller and devices is defined as the profit. In this algorithm, to find the number of controllers, the algorithm stops when all devices are assigned to controllers or there is not a possible controller placement in the network to satisfy the constraints. DCP-GK solves the CPP in a static network. DCP-SA is proposed to reassign controllers to devices dynamically based on network traffic. The input of the DCP-SA is a feasible controller assignment that satisfies the capacity of controllers and the delay constraint provided by an algorithm. Then, DCP-SA, using a simulated annealing approach, improves the controller assignments. In both DCP-GK and DCP-SA, the controller assignment is based on the
capacity of the controllers.

In most studies, the proposed heuristic algorithms use the capacity of controllers to find the best controller placements and assignments [42,91]. However, nowadays, the capacity of controllers does not have a vital role in solving the CPP since SDN controllers such as ONOS [92] and OpenDaylight [93] are able to support millions of requests within a reasonable response time.

Delay Among the Possible Controller Placements and Devices

In the proposed greedy algorithm in [43], to find the number of controllers in the network, in the first iteration, all possible controller placements are selected. After that, in each iteration, one of the placements is selected and reduced from the set of controllers. The devices assigned to the reduced controller will be assigned to the closest controllers while satisfying the controller-device delay constraint. Finally, the algorithm stops if reducing a new controller increases the network cost. In this paper, the authors do not provide any details about how to select a controller in each iteration to be reduced from the set of controllers.

Ranking Degrees of the Possible Controller Placements

As mentioned earlier in [40,45] and [44], controller placements are selected using a ranking algorithm. In [40], the ranking algorithm is based on the average delay of the possible placement to all other devices in the network and the average number of links in the interference range of the communication links to other devices. Then, the possible placement with the smallest degree is selected. However, the proposed algorithm does not find the number of controllers in the network. On the other hand, in [45] and [44], based on the connectivity degree of possible controller placements to other devices in the network, in each iteration, a controller placement with a higher degree is selected to be added to the set of controllers. In [45], these iterations continue until adding a new controller increases the network cost. In [44], the number of controllers are estimated by dividing the total degrees of all devices to the capacity of a controller. The capacity of a controller defines the maximum number of allowed flow requests to be assigned to a controller in the network.

GA-Based Heuristic Approaches to the CPP

Different studies propose a GA-based heuristic approach to the CPP to find near-optimal solutions for large-scale wired networks. In [94], the objective is to partition the network
into a given number of domains and find a controller placement in each domain while minimizing propagation latency in controller-device and inter-controller communications. Moreover, the authors consider load-balancing among controllers in their approach. To achieve this goal, a multi-objective GA-based approach is proposed to determine the domains in the network. Then, another algorithm finds a controller placement in each domain that has the minimum distance from the possible placement to other devices in the domain.

The results demonstrate that although increasing the number of controllers in a network can reduce the objective function outcomes, after a certain value, placing more controllers may lead to a higher value of the objective function in the network. In addition, the results show that a higher number of iterations in the GA algorithm does not necessarily result in reducing the value of the objective function.

Sanner et al. in [95] propose a GA algorithm that aims to find near-optimal solutions for the CPP and partitions the network into different domains while maximizing the average connectivity among devices and controllers in each domain and considering load-balancing among controllers. In the proposed GA algorithm, to demonstrate the number of domains in the network, a random number is selected among a predefined interval. The results show that the proposed GA algorithm is able to find optimal solutions for small-scale networks and the Pareto front for both objectives in medium-size networks. In addition, based on the results, the computational time of the proposed algorithm varies depending on the network topology.

In [96], the authors present a multi-objective GA algorithm to the CPP to minimize propagation latency in controller-device and inter-controller communications in the network considering load-balancing among controllers. They also propose a particle swarm optimization (PSO) [97] function to improve the Pareto front for multiple objectives. The results demonstrate that the proposed algorithm is able to outperform in terms of propagation latency and load-balancing among controllers in the network compared to the Matlab GA solver.

A multi-objective GA algorithm is proposed in [98] with the objective of finding near-optimal solutions for the CPP to minimize inter-controller latency and the number of controllers as well as providing load-balancing among controllers in the network. To achieve this goal, the proposed algorithm minimizes the number of domains in the network approximately using different operations. Using fitness sharing for the defined objectives, the proposed algorithm is able to minimize the number of controllers and inter-controller propagation latency approximately and manage load imbalance among controllers.

Mohanty et al. in [99], propose a GA algorithm to find near-optimal solutions for the CPP that aims to minimize the network cost that includes the cost of placing controllers
in the network and the cost of controller-device communications considering the distance among them and the rate of forwarded requests to each controller. To evaluate the proposed algorithm, a random placement algorithm and a simulated annealing [100] based algorithm are considered. The results demonstrate that the proposed GA algorithm achieves lower network cost and execution time when placing different number of controllers in the network compared to other two algorithms.

A GA-based approach is presented in [101] to the CPP to approximately minimize controller-device propagation latency while providing load-balancing among controllers. Using a gradient descent [102] based algorithm for the fitness evaluation of the solutions, the proposed GA algorithm is able to outperform in terms of latency and load balancing among controllers compared to a random placement, the K-median algorithm and a greedy algorithm based on the capacity constraint of the controllers in both small-scale and large-scale networks.

In [103], a traffic-aware controller placement GA algorithm is proposed, in which the objective is to find controller placements and assignments dynamically considering load balancing among controllers and minimizing propagation latency. The authors in [103] consider traffic variation that may result in load imbalance among controllers. The results show that the proposed GA algorithm is able to dynamically manage and reduce load imbalance among controllers.

The objective of [104] is to find controller placement and assignment while considering controller-device propagation latency, the number of hops among devices and their assigned controllers, and link utilization for the exchanged control traffic. In [104], a GA algorithm is proposed to find near-optimal solutions for the proposed CPP. Moreover, a new algorithm called analytic hierarchy process (AHP) is proposed to consider the three defined objectives in the controller assignment. Using the proposed algorithms, the results demonstrate that the network experiences lower propagation latency and higher link utilization.

Huang et al. [105] introduce a clustering based GA approach to the CPP to partition the network into different domains and place controllers in each domain. They also propose a gradient descent scheduling algorithm to balance loads among different domains in the network. Considering different request arrival rates to the controllers, the results show that the proposed approach is able to reduce the response time compared to the k-center algorithm in large-scale networks. Moreover, compared to a capacity-based scheduling algorithm, the proposed approach increases control plane utilization.

**Summary:** In chapter 4, the GA-based approaches to the CPP in the literature are adapted to the proposed optimization problem in SDMWN to find near-optimal solutions and investigate the impact of the proposed algorithm in a network with different number
of network devices. After that, Chapter 5, presents the extended heuristic algorithms in this section [40, 43–45] to find near-optimal solutions for the proposed model in SDMWN.

### 2.6 Deep RL Approaches to the CPP

Recently, a few research works have been conducted to apply reinforcement learning and deep reinforcement learning to the CPP. In reinforcement learning (RL), using feedbacks from the environment and a Q-table, a learner called the agent is responsible for learning how to map the situations in the environment to the actions to maximize the achieved reward [51]. In [106], a heuristic algorithm based on learning automaton is proposed to find the number of controllers and controller placements in wired networks to minimize latency among controllers and devices. In this paper, using random actions and evaluations of the actions, the learning automaton learns to select optimal actions in the network. The results show that, the proposed approach is able to reduce latency compared to the K-means clustering algorithm [107].

Deep reinforcement learning (Deep RL) [49–52] is able to overcome the existing challenges of RL, including low-dimensional of situation-action representations in Q-table based on the advantages of artificial neural networks. Using the training data, Deep RL is able to learn and make predictions for the new data sets which is beneficial in dynamic environments. In [108], the authors propose a multi-objective optimization problem (D4CPP) to solve the CPP in wired networks that aims to minimize latency among the devices and their assigned controllers and balance the load of the control plane among controllers. D4CPP, also, considers data flow fluctuations in the network. Therefore, to solve the problem in a dynamic environment, a deep Q-network based approach (DQN) [109] based on the data flow fluctuations is proposed. D4CPP consists of two sub-systems such that in the first sub-system, the objective is to find the placements of a given number of controllers. While, the second sub-system dynamically adjusts the controller assignments to network devices. The results demonstrate that, compared to K-means algorithm [107], D4CPP is able to improve the network performance in terms of latency among the devices and controllers and load-balancing among controllers.

Yuan et al. in [110] propose a dynamic controller assignment in internet of vehicles (IoV) with a hierarchical control plane using multi-agent Deep RL to minimize control delay and control traffic load among vehicles and their assigned controllers. In the proposed approach, multi-agent deep deterministic policy gradient (MADDPG) [111], the controller placements are fixed and act as agents in the algorithm to dynamically adjust controller
assignments. The results show that the proposed approach is able to reduce and packet loss in the network.

**Summary:** Based on the reviewed studies, applying Deep RL to the CPP can be beneficial in a network in terms of network performance metrics including minimizing delay and control traffic, and load-balancing in a network. In Chapter 6, the objective is to investigate the impact of proposing a Deep RL to the proposed CPP in this thesis for SDMWN on network performance.

### 2.7 Open Research Areas

Based on the related work reviewed in this chapter, to solve the wireless CPP in SDMWN, there are several open research opportunities that need to be taken into consideration in the future work to improve the performance of SDMWN while minimizing the network cost.

To calculate the network cost in SDMWN, we need to consider the control overhead generated by communications among controllers and devices in both directions to exchange configurations and set up flow rules, and to discover the network topology as well as the control overhead generated by inter-controller communications to synchronize network views. However, in [39, 45, 46], the control overhead generated by controller-device communications to discover the network topology is not considered as the network cost. Therefore, in Chapter 3, two different approaches are considered to calculate the network cost in SDMWN, in which the network cost consists of the cost of controller-device communications to set up flow rules and exchange configurations, and to discover the network states and topology as well as the network cost to provide inter-controller communications.

In addition, in the related work reviewed in this chapter, the existing research efforts assume that all communications use the shortest paths and they do not consider the capacity of links and the impact of limiting the capacity of links to exchange the control overhead on solving the CPP. In Chapter 3, the CPP in SDMWN is formulated to find a given number of controller placements in the network and assign controllers to devices while minimizing the network cost as an objective and satisfying the capacity of links as a constraint. Moreover, in the proposed models in Chapter 3, the objective is to minimize the number of hops among devices and their assigned controllers, and among controllers with the aim of satisfying the capacity of links. Minimizing the number of hops in the wireless communication has a significant effect on minimizing the latency and improving reliability in the control plane.
Further, determining the number of controllers has a direct impact on the network cost. In Chapter 3, the results demonstrate that depending on the characteristics of the network such as the cost of flow rule set up, the capacity of links, etc., in some cases, having a centralized control plane compared to a distributed control plane can be costly. Therefore, selecting the right number of controllers to minimize the network cost and satisfy constraints needs to be investigated in the future work. This is also a part of the proposed heuristic approaches presented in Chapter 5.

Finally, the related work demonstrate that, Deep RL approaches are able to be beneficial in solving the CPP in SDMWN. Investigating the impact of using a Deep RL approach on solving the CPP in SDMWN to minimize the network cost will be presented in Chapter 6.
Chapter 3

Optimization Models for SDMWN Controller Placement Problem

3.1 Introduction

One of the costs of applying an SDN architecture to a network is the need for exchanging controller-device and inter-controller control packets to discover the network states and topology, exchange configurations and update flow rules in SDN forwarding devices [56].

Although applying an out-of-band SDN architecture to MWNs reduces collisions and delay in the network, some types of networks, such as WSNs, because of their characteristics, only have a single interface to forward both control and data messages. Therefore, in such networks, the capacity of links is used to exchange both data and control traffic [1]. Moreover, because of the wireless nature of links, interference can influence the available capacity of links [33, 34]. In this case, the available capacity of a link is used by the traffic flowing over each link as well as the traffic flowing over the links in the interference range of that link.

In an SDN architecture with multiple controllers, in addition to controller-device communications, controllers need to communicate with each other to discover the network topology and synchronize the network views of different controllers. Therefore, it is important to consider the capacity of links as a constraint in solving the CPP in SDMWN to exchange control traffic [112].

Hence, in solving the CPP in SDMWN, we should consider the capacity of links as well as the impact of interference on the capacity of links to find the right number of controllers
and the best controller placements and assignments. However, to the best of our knowledge, in the related work in solving the CPP, all communications use the shortest paths and the existing research efforts have not considered the impact of limiting the capacity of links to exchange the control overhead and the effect of interference on the capacity of links, which is crucial in SDMWN.

In this chapter, the objective is to find the optimal placements of \( N \) controllers in SDMWN with an in-band architecture and assign controllers to network devices to minimize the network cost. The network cost is considered as the total number of control packets exchanged in the network per second.

Two different optimization models are proposed to solve the problem. In both models, the capacity of links and the impact of interference on the capacity of links are considered as a constraint to place controllers in the network and assign them to network devices while minimizing the network cost. The two models use two different approaches to discover the network topology and demonstrate the impact of topology discovery process on the network cost. In both models, each controller obtains the global view of the network. In the first model, called topology discovery using controller-device communications (TDCD), each controller runs topology discovery on all network devices to achieve the global view of the network. Therefore, it is not required to provide inter-controller communications to discover the network topology. On the other hand, in the second model, called topology discovery using inter-controller and controller-device communications (TDCCCD), based on a flat control plane architecture, each controller runs topology discovery on its own assigned devices. Therefore, to obtain the global view of the network in the second model, all controllers communicate with each other (flat architecture) in an interval of time.

The rest of this chapter is organized as follows. Section 3.2 introduces the system model used in both optimization models to solve the CPP in SDMWN. Sections 3.3 and 3.4 present the two proposed optimization models. Section 3.5 shows the evaluation of the proposed models. Finally, Section 3.6 concludes the chapter.

### 3.2 Model Formulation

An SDMWN is modeled as a directed graph \( G = (V, E) \), where \( V \) represents the set of wireless network devices and \( E \) is the set of links between each pair of devices such that link \((u, v)\) and link \((v, u)\) are the members of \( E \) if and only if device \( u \) is within the transmission range of device \( v \) \((R_T)\). The protocol model formulated in [34] is used to find a set of links in the interference range of link \((u, v)\). In the proposed models in the thesis, finding the
set of links in the interference range of link \((u, v)\) helps to calculate the total number of control packets flowing over link \((u, v)\) per second. The total packets consist of the total packets flowing over the link \((u, v)\) as well as the total packets flowing over the set of links in the interference range of link \((u, v)\). Section 3.3 and Section 3.4 describe the proposed models in more detail.

Since in MWNs wireless network devices are responsible for organizing the network and delivering messages, all wireless network devices are considered as SDN forwarding devices, communicating in a multihop manner, controlled by an SDN controller. Moreover, both models assume that all network devices are candidate locations to host a controller. In addition, the network architecture of both models is in-band, in which network devices use a single interface to forward data traffic as well as exchanging control traffic with a controller placed on a device and exchanging control traffic among controllers.

In both proposed models in this thesis, it is assumed that, the information required to be exchanged in the control plane in each part of the models can fit into a single control packet that can be an OpenFlow packet [16].

### 3.2.1 Notations

The notations used in the proposed models are listed in Table 3.1.

### 3.2.2 Model Outputs

The outputs of both proposed optimization models in this chapter are listed as follows while considering the capacity of links as a constraint to place controllers in the network and assign them to network devices.

- Optimal placements of \(N\) controllers in an SDMWN
- Controller assignments to network devices
- Route selections among controllers and network devices as well as among controllers
- Optimal cost of placing \(N\) controllers in the network
Table 3.1: Notations used in Chapter 3

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_k$</td>
<td>Output (binary decision variable): The value equals one if and only if there is a controller placed on device $k$</td>
</tr>
<tr>
<td>$x_{k,i}$</td>
<td>Output (binary decision variable): The value equals one if and only if device $i$ is assigned to a controller placed on device $k$</td>
</tr>
<tr>
<td>$f_{a,b}^{u,v}$</td>
<td>Output (binary decision variable): The value equals one if and only if link $(u,v)$ is used to provide communication between device $a$ and device $b$</td>
</tr>
<tr>
<td>$Cost^k_{TD}$</td>
<td>The total cost of topology discovery (control packets/second) running by a controller placed on device $k$ on all network devices in the network calculated using (3.2) (TDCD model)</td>
</tr>
<tr>
<td>$Cost_{TD}$</td>
<td>The total cost of topology discovery (control packets/second) running by all controllers on their own assigned network devices and among all controllers in the network calculated using (3.12) (TDCCCD model)</td>
</tr>
<tr>
<td>$R_{TD}$</td>
<td>The rate of running topology discovery by each controller (1/second)</td>
</tr>
<tr>
<td>$R_{Flow,Rq}$</td>
<td>The arrival rate of new flows in each device that triggers a flow request message toward the assigned controller (1/second)</td>
</tr>
<tr>
<td>$N$</td>
<td>A given number of controllers</td>
</tr>
<tr>
<td>$neighbor[i]$</td>
<td>A set of neighbors of network device $i$ in the network</td>
</tr>
<tr>
<td>$neighbor_{i,j}$</td>
<td>The $j^{th}$ neighbor of network device $i$</td>
</tr>
<tr>
<td>$C_{u,v}$</td>
<td>The capacity of link $(u,v)$ to exchange control packets (control packets/second)</td>
</tr>
<tr>
<td>$C'_{u,v}$</td>
<td>The required bandwidth of link $(u,v)$ to exchange control packets (control packets/second)</td>
</tr>
<tr>
<td>$L_{Int}^{u,v}$</td>
<td>A set of links in the interference range of link $(u,v)$</td>
</tr>
</tbody>
</table>
3.3 TDCD Optimization Model

In the TDCD model, the CPP in SDMWN is formulated so that each controller runs topology discovery on the whole network to obtain the global view. In this case, it is not required to provide inter-controller communications. Therefore, the network cost in this model consists of the total number of controller-device control packets exchanged per second to discover the network states and topology by each controller as well as the total number of controller-device control packets exchanged per second to exchange configurations and set up flow tables.

3.3.1 Objective Function

The problem is formulated as a nonlinear programming problem (NLP) and Equation (3.1) shows the objective function that aims to minimize the total network cost.

\[
\text{Min} \left( \sum_{k=1}^{V} \text{Cost}_{TD}^k y_k + \sum_{k=1}^{V} \sum_{i=1, i \neq k}^{V} \left( R_{Flow,Rq} x_{k,i} \sum_{\forall (u,v) \in E} f_{u,v}^k + \sum_{\forall (u,v) \in E} f_{u,v}^i \right) \right) \quad (3.1)
\]

subject to:

\((3.3), (3.4), (3.5), (3.6), (3.7), (3.10)\)

The first part of Equation (3.1) \(\sum_{k=1}^{V} \text{Cost}_{TD}^k y_k\) calculates the total cost of topology discovery (control packets/second). The cost of topology discovery of each possible controller placement is calculated using Equation (3.2). The second part of Equation (3.1) calculates the cost of controller-device communications using the best routes among controllers and their assigned network devices to exchange configurations and set up flow rules (control packets/second). In the case here, the model counts control packet transmissions over each hop along their routes. In other words, number of hops in transmitting a control packet has a direct effect on the network cost.

(A) Cost of Topology Discovery
To discover the network topology, after connecting the forwarding devices to the controller, the controller constructs and sends probes periodically to all connected devices containing their device IDs. For example, OpenDaylight [93], the most popular open source SDN controller, sends out the probes every 5 seconds \( R_{TD} = 0.2 \) (control packets/second). Usually, these probes contain link layer discovery protocol (LLDP) [113] messages sent to the forwarding devices as PacketOut messages [114].

Therefore, this model calculates the total number of control packets generated by the controllers per second and count the number of hops \( \sum_{(u,v) \in E} f_{u,v}^{k,i} + \sum_{j=1,j \neq k}^{\text{neighbor}[i]} \sum_{(u,v) \in E} f_{u,v}^{\text{neighbor}_{i,j,k}} \) in the best route between a controller placed on device k and each device (the first part of Equation (3.2)). The objective function in Equation (3.1) selects the best routes with minimum number of hops among each controller and all devices while satisfying the defined constraints in Section 3.3.2.

When the assigned devices receive the probes from the controller containing its device ID, they flood the probes. This cost is inevitable for any controller placement, resulting in a packet being transmitted over a wireless link to discover that link. Therefore, we do not need to model this cost explicitly in solving the CPP.

After flooding the probes, their neighbors send the received probes to the sending controller as PacketIn messages that helps the controller to construct the view of the links. Using these packets, each controller is able to find the links among devices. The second part of Equation (3.2) calculates this cost, in which the neighbors of each device send back the probes as PacketIn messages to the controller using the best route between the controller and each neighbor \( \sum_{j=1,j \neq k}^{\text{neighbor}[i]} \sum_{(u,v) \in E} f_{u,v}^{\text{neighbor}_{i,j,k}} \).

(B) **Cost of Exchanging Configurations and Setting Up Flow Tables**

When a forwarding device receives a flow that does not match any flow entries in its own flow table, basically, the device encapsulates the incoming packet or the header of the packet and forwards that as a PacketIn message to its assigned controller. After receiving the PacketIn message by the controller, the controller establishes flow rules using FlowMod messages forwarded to the devices [115].

\[
Cost^k_{TD} = R_{TD} \left[ \sum_{i=1,i \neq k}^{V} \left( \sum_{(u,v) \in E} f_{u,v}^{k,i} \right) + \sum_{j=1,j \neq k}^{\text{neighbor}[i]} \sum_{(u,v) \in E} f_{u,v}^{\text{neighbor}_{i,j,k}} \right] \tag{3.2}
\]
The second part of Equation (3.1) calculates the total number of control packets exchanged among controllers and their own assigned network devices to exchange configurations and set up flow tables, counting the number of hops in both directions among controllers and devices \((\sum_{(u,v) \in E} f^{k,i}_{u,v} + \sum_{(v,u) \in E} f^{i,k}_{u,v})\), and the arrival rate of flow requests in the controllers. To simplify the model, it is assumed that the arrival rate of flow requests for all devices are the same and constant. This model considers the cost of communications in both directions between a pair of device and its assigned controller, compared to the models proposed in [42–46] that only consider the cost of sending flow requests from the devices to their controllers using the shortest paths.

### 3.3.2 Constraints

The objective function defined in Equation (3.1) is subject to the following constraints. The constraint defined in Equation (3.3) avoids assigning a device to a controller that is not placed in the network.

\[
x_{k,i} \leq y_k, \quad \forall i, k \in V
\]  

(3.3)

The constraint defined in Equation (3.4) ensures that each device is assigned to exactly one controller.

\[
\sum_{k=1}^{|V|} x_{k,i} = 1, \quad \forall i \in V
\]  

(3.4)

Equation (3.5) ensures that there is a given number of controllers in the network.

\[
\sum_{k=1}^{|V|} y_k = N
\]  

(3.5)

Equation (3.6) defines the control flow conservation constraint, in which the constraint ensures that the total control flow entering each device in the network equals to the total control flow leaving the device except for the source and destination of a flow.

\[
\sum_{(u,v) \in E} f^{a,b}_{u,v} - \sum_{(v,w) \in E} f^{a,b}_{v,w} = \begin{cases} 
1, & \text{if } v = b \\
-1, & \text{if } v = a \\
0, & \text{otherwise}
\end{cases}, \quad \forall a, b \in V
\]  

(3.6)
In both proposed optimization models in this thesis, it is assumed that, only a fraction of the link capacity is assigned to exchange control packets. The constraint defined in Equation (3.7) ensures that each link can handle the total number of control packets flowing over the link per second.

In the TDCD model, the total number of control packets flowing over link \((u, v)\) in the network consists of the total number of control packets exchanged per second to discover the network topology, exchange configurations and set up flow tables \((C'_u,v)\) as well as the total number of control packets flowing over links in the interference range of link \((u, v)\) per second \((C'_{u',v'})\).

\[
C'_u,v + \sum_{\forall(u',v') \in L_{u,v}} C'_{u',v'} \leq C(u,v), \ \forall(u, v) \in E \quad (3.7)
\]

Equation (3.8) calculates the total number of control packets flowing over a link per second to discover the network topology (the first line) and to provide controller-device communications to exchange configurations and set up flow tables (the second line).

\[
C'_{u,v} = \sum_{k=1}^{|V|} (R_{TD} \left[ \sum_{i=1,i \neq k}^{|V|} (f^k,i_{u,v} + \sum_{j=1,j \neq k}^{|\text{neighbor}[i]|} f^{\text{neighbor}_i,j,k}_{u,v}) \right] ) y_k + \sum_{k=1}^{|V|} \sum_{i=1,i \neq k}^{|V|} (R_{Flow,Rq} x_{k,i} [f^k,i_{u,v} + f^i,k_{u,v}]), \ \forall(u, v) \in E \quad (3.8)
\]

As mentioned earlier, the model presented in [34] is adopted to find a set of links in the interference range of a link. Equation (3.9) shows the total number of control packets flowing over a link in the interference range of link \((u, v)\) per second. These packets consist of the control packets flowing over the link to discover the network topology, exchange configurations and set up flow tables.

\[
C'_{u',v'} = \sum_{k=1}^{|V|} (R_{TD} \left[ \sum_{i=1,i \neq k}^{|V|} (f^{k,i}_{u',v'} + \sum_{j=1,j \neq k}^{|\text{neighbor}[i]|} f^{\text{neighbor}_i,j,k}_{u',v'}) \right] ) y_k + \sum_{k=1}^{|V|} \sum_{i=1,i \neq k}^{|V|} (R_{Flow,Rq} x_{k,i} [f^k,i_{u',v'} + f^i,k_{u',v'}]), \ \forall(u', v') \in L_{u,v} \quad (3.9)
\]

Integrality constraints are presented in Equation (3.10).

37
In the TDCD model introduced in Section 3.3, each controller is responsible for running topology discovery to obtain the global view of the network. Therefore, modeling the inter-controller communications to discover the network topology is not considered in Section 3.3.

In this section, the objective is to modify the topology discovery process in the TDCD model so that each controller runs topology discovery only on its own assigned devices. Therefore, in the new optimization model called TDCCCD, to obtain the global view of the network on each controller, all controllers need to communicate together in an interval of time (flat architecture). In TDCCCD, the cost of topology discovery (control packets/second) consists of the cost of running topology discovery by each controller on its own assigned devices and the cost of communications among controllers to obtain the global view of the network.

### 3.4 TDCCCD Optimization Model

In the TDCD model introduced in Section 3.3, each controller is responsible for running topology discovery to obtain the global view of the network. Therefore, modeling the inter-controller communications to discover the network topology is not considered in Section 3.3.

In this section, the objective is to modify the topology discovery process in the TDCD model so that each controller runs topology discovery only on its own assigned devices. Therefore, in the new optimization model called TDCCCD, to obtain the global view of the network on each controller, all controllers need to communicate together in an interval of time (flat architecture). In TDCCCD, the cost of topology discovery (control packets/second) consists of the cost of running topology discovery by each controller on its own assigned devices and the cost of communications among controllers to obtain the global view of the network.

#### 3.4.1 Objective Function

The new problem is formulated as an NLP problem and Equation (3.11) shows the objective function to minimize the total network cost (control packets/second). The only difference between the TDCD and the TDCCCD models is in achieving the global view of the network in each controller as shown in the first parts of Equation (3.1) and Equation (3.11).

\[
\text{Min}(\text{Cost}_{TD} + \sum_{k=1}^{\lvert V \rvert} \sum_{i=1, i \neq k}^{\lvert V \rvert} (R_{Flow,Rq} x_{k,i} [\sum_{(u,v) \in E} f_{u,v}^{k,i} + \sum_{(u,v) \in E} f_{i,u}^{i,k}] ) )
\]

subject to:

\[(3.3), (3.4), (3.5), (3.6), (3.7), (3.10)\]

The first part of Equation (3.11) (\text{Cost}_{TD}) shows the total cost of topology discovery in the network calculated using Equation (3.12). The second part of Equation (3.11) calculates
the cost of controller-device communications using the best routes among controllers and their assigned network devices to exchange configurations and set up flow rules as discussed earlier in Section 3.3.

(A) Cost of Topology Discovery

In the TDCCCD model, each device communicates only with its own assigned controller and each controller discovers a partial view of the network including its own assigned devices. Therefore, controllers communicate together to obtain the global view of the network.

\[
Cost_{TD} = R_{TD} \sum_{k=1}^{V} \sum_{i=1, i \neq k}^{V} \left( \sum_{(u,v) \in E} f_{i,j}^{k} \right) + \\
\sum_{m=1}^{V} \sum_{j=1, j \neq k, m}^{V} \sum_{(u,v) \in E} f_{i,j}^{\text{neighbor}, j,m} x_{m, \text{neighbor}, j} x_{k,i} \right] + \\
R_{TD} \sum_{k=1}^{V} \sum_{p=1, p \neq k}^{V} \left( y_{k} y_{p} \sum_{(u,v) \in E} f_{u,v}^{k,p} \right) \] \tag{3.12}

The first and the second lines of \(Cost_{TD}\) shown in Equation (3.12) calculate the total number of control packets exchanged among controllers and devices per second to discover the network topology. The third line of Equation (3.12) calculates the number of control packets exchanged among controllers per second to obtain the global view of the network.

To discover the network topology in the TDCCCD model compared to the TDCD model, each controller constructs and sends out probes periodically to its own assigned devices. Therefore, since each device only communicates with its assigned controller to discover the network topology, it only sends out the probes received from its own controller to all of its neighbors. In this case, compared to the TDCD model, in which each device receives probes from all controllers in the network and needs to send them out to all of its port, the TDCCCD model generates fewer packets in the control plane. In addition, in the TDCCCD model, after sending out the probes by devices to all of their ports, their neighbors send the received probes to their own controllers. Using these packets, each controller is able to find the links among its own devices and to devices outside of its partial view.
In addition, controllers need to communicate in an interval of time to achieve the
global view of the network, considering the number of hops between each pair of con-
trollers \( \sum_{v(u,v)\in E} f^{k,p}_{u,v} \). The TDCCCD model considers the same interval of time for
both running topology discovery by each controller and inter-controller communications.

(B) Cost of Exchanging Configurations and Setting Up Flow Tables

As discussed earlier in Section 3.3.1, in addition to the cost of topology discovery, we need
to consider the cost of controller-device communications to exchange configurations and set
up flow tables. The second part of Equation (3.11) calculates the total number of control
packets exchanged among controllers and their own assigned network devices to set up flow
rules and exchange configurations, counting the number of hops in both directions among
controllers and devices, and the arrival rate of flow requests in the controllers.

3.4.2 Constraints

The objective function presented in Equation (3.11) is subject to the defined constraints
in Section 3.3.2. However, since the TDCCCD model uses a different way to provide a
global view of the network on all controllers in the network, the total number of control
packets flowing over links to discover the topology are calculated differently. Therefore,
Equation (3.8) and Equation (3.9) are modified to Equation (3.13) and Equation (3.14),
respectively.

Equation (3.13) calculates the total number of control packets flowing over link \((u,v)\)
per second to discover the network topology (the first and the second lines) and to provide
controller-device communications to exchange configurations and set up flow rules (the
third line).

\[
\begin{align*}
C'_{u,v} &= R_{TD} \sum_{k=1}^{V} \sum_{i=1, i \neq k}^{V} \left( f^{k,i}_{u,v} + \sum_{m=1}^{V} \sum_{j=1, j \neq k}^{V} f^{neighbor\_i,\_m}_{u,v} x_{m,neighbor\_i,\_j} \right) x_{i,k} + \\
R_{TD} \sum_{k=1}^{V} \sum_{p=1, p \neq k}^{V} (y_{k} y_{p} f^{k,p}_{u,v}) + \\
\sum_{k=1}^{V} \sum_{i=1, i \neq k}^{V} (R_{Flow,Rq} x_{k,\_i,\_u} [f^{k,i}_{u,v} + f^{i,k}_{u,v}]), \quad \forall (u,v) \in E
\end{align*}
\]
Equation (3.14) shows the total number of control packets flowing over a link in the interference range of link \((u, v)\) per second.

\[
C_{u',v'} = R_{TD} \left[ \sum_{k=1}^{V} \sum_{i=1,i \neq k}^{V} (f_{u',v'}^{k,i} + \sum_{m=1}^{V} \sum_{j=1,j \neq k}^{V} f_{u',v'}^{\text{neighbor}_i,j,m} x_{m,\text{neighbor}_i,j}) x_{i,k} \right] + \\
R_{TD} \left[ \sum_{k=1}^{V} \sum_{p=1,p \neq k}^{V} (y_k y_p f_{u',v'}^{k,p}) \right] + \\
\sum_{k=1}^{V} \sum_{i=1,i \neq k}^{V} (R_{Flow} R_{q} x_{k,i} [f_{u',v'}^{k,i} + f_{u',v'}^{i,k}], \forall (u', v') \in L_{u,v}^{\text{Int}})
\]

(3.14)

### 3.5 Model Results and Analysis

This thesis uses a mathematical programming language (AMPL) [116, 117] to implement the proposed optimization problems and the Baron solver 19.7.13 [118] which aims to find the optimal solutions globally for nonlinear optimization problems. NEOS server [119–121] is used to run the proposed models in AMPL running on an Intel Xeon E5-2698 @ 2.3GHz, 192GB RAM and 300G SAS drives setup in RAID5.

**Case Study**

Because of the high computational complexity of the proposed models in this chapter, a small network with 6 wireless network devices is considered as shown in Fig. 3.1 to illustrate the performance of the proposed optimization models. Here is an example of the links in the interference range of link \((1, 2)\) in the topology shown in Fig. 3.1: \(L_{1,2}^{\text{Int}} = \{(1, 3), (2, 4), (3, 4), (4, 2), (4, 5), (5, 3), (6, 4), (2, 1), (3, 1), (3, 5), (4, 3), (4, 6), (5, 4)\}\). Therefore, the total number of packets flowing over link \((1, 2)\) per second consists of the total number of packets flowing over the link per second and the total number of packets flowing over the links in \(L_{1,2}^{\text{Int}}\) per second.

The objective of the proposed optimization models is to place \(N\) controllers in this network, which aims to minimize the cost of the control plane defined in Equation (3.1) and Equation (3.11), and satisfy the defined constraints in Section 3.3.2 and Section 3.4.2 for the TDCD model and the TDCCCD model, respectively. In this evaluation, it is
assumed that each device receives a new flow every 2 seconds ($R_{Flow,Rq} = 0.5 \, (1/second)$) and each controller runs topology discovery every 5 seconds ($R_{TD} = 0.2 \, (1/second)$), which is adopted from OpenDaylight [93].

### 3.5.1 The Impact of the Number of Controllers On the Network Cost

Table 3.2 shows that, using the TDCD and the TDCCCD models, in the topology shown in Fig. 3.1, if $N = 1$, device 4 is the optimal controller placement to minimize the number of control packets exchanged in the network per second and satisfy the defined constraints. As shown in Table 3.2, in both proposed models, the cost of running topology discovery when the controller is placed on device 4 is 3.2 ($control \, packets/second$) and the cost of exchanging configurations and setting up flow rules in all devices assigned to this controller is 6 ($control \, packets/second$). Therefore, Table 3.2 presents the total network cost achieved from both proposed models that shows the minimum number of control packets exchanged per second in this topology when $N = 1$ to discover the network topology and set up flow rules in all devices.

To find the optimal network cost achieved from both models for the topology shown in Fig. 3.1 when placing different number of controllers, we need to run the proposed optimization models for different values of $N$ from 1 (placing a controller in one of the devices) to 6 (placing a controller on each device). As mentioned earlier, the total network cost consists of the total cost of topology discovery and the total cost of controller-device communications to exchange configurations and set up flow tables in devices. Table 3.3
Table 3.2: Optimal cost of placing one controller in both TDCD model and TDCCCD model (control packets/second)

<table>
<thead>
<tr>
<th>Controller placement</th>
<th>Cost of topology discovery</th>
<th>Cost of flow rule setup</th>
<th>Total network cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>device 4</td>
<td>3.2</td>
<td>6</td>
<td>9.2</td>
</tr>
</tbody>
</table>

and Table 3.4 demonstrate the optimal total network cost achieved from the TDCD model and the TDCCCD model, respectively when placing different number of controllers in the network.

Table 3.3: Optimal cost of placing $N$ controllers in TDCD model (control packets/second)

<table>
<thead>
<tr>
<th>$N$</th>
<th>Controller Placement</th>
<th>Cost of topology discovery</th>
<th>Cost of flow rule setup</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3.2</td>
<td>6</td>
<td>9.2</td>
</tr>
<tr>
<td>2</td>
<td>3, 4</td>
<td>7.4</td>
<td>4</td>
<td>11.4</td>
</tr>
<tr>
<td>3</td>
<td>3, 4, 5</td>
<td>11.6</td>
<td>3</td>
<td>14.6</td>
</tr>
<tr>
<td>4</td>
<td>2, 3, 4, 5</td>
<td>17.2</td>
<td>2</td>
<td>19.2</td>
</tr>
<tr>
<td>5</td>
<td>2, 3, 4, 5, 6</td>
<td>23.2</td>
<td>1</td>
<td>24.2</td>
</tr>
<tr>
<td>6</td>
<td>1, 2, 3, 4, 5, 6</td>
<td>29.6</td>
<td>0</td>
<td>29.6</td>
</tr>
</tbody>
</table>

As shown in Table 3.3, when the optimization problem proposed in the TDCD model is solved to place one controller in the network ($N = 1$), the minimum network cost is achieved and the optimal controller placement is device 4. On the other hand, as presented in Table 3.4, using the TDCCCD model, the minimum network cost is achieved when three controllers ($N = 3$) are placed in the network. In this case, the optimal controller placements are devices 3, 4 and 5.

Moreover, as demonstrated in Table 3.3, using the TDCD model, when we solve the problem to place six controllers, the network experiences the highest cost of the control plane defined in Equation (3.1). On the other hand, as shown in Table 3.4, the highest network cost calculated using Equation (3.11) in the TDCCCD model is achieved when
Table 3.4: Optimal cost of placing $N$ controllers in TDCCCD model (control packets/second)

<table>
<thead>
<tr>
<th>$N$</th>
<th>Controller Placement</th>
<th>Cost of topology discovery</th>
<th>Cost of flow rule setup</th>
<th>Total cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3.2</td>
<td>6</td>
<td>9.2</td>
</tr>
<tr>
<td>2</td>
<td>3, 4</td>
<td>2</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3, 4, 5</td>
<td>2.2</td>
<td>3</td>
<td>5.2</td>
</tr>
<tr>
<td>4</td>
<td>2, 3, 4, 5</td>
<td>3.6</td>
<td>2</td>
<td>5.6</td>
</tr>
<tr>
<td>5</td>
<td>2, 3, 4, 5, 6</td>
<td>5.8</td>
<td>1</td>
<td>6.8</td>
</tr>
<tr>
<td>6</td>
<td>1, 2, 3, 4, 5, 6</td>
<td>9.2</td>
<td>0</td>
<td>9.2</td>
</tr>
</tbody>
</table>

placing one or six controllers in the network. Therefore, in each proposed model, finding the right number of controllers in a network, depending on the network topology, has a direct effect on the cost of the network.

Fig. 3.2 shows the total cost of topology discovery achieved from the TDCCCD model, which consists of the total cost of controller-device communications to discover the network state and topology, and the total cost of inter-controller communications to integrate views from each domain and obtain the global view of the network. As demonstrated in this figure, as the number of controllers increases in the network, the cost of controller-device communications decreases, while the network faces higher cost of inter-controller communications to integrate different network views. Although placing three controllers in the network results in the minimum total network cost as shown in Fig. 3.3, the optimal cost of topology discovery is obtained when we place two controllers in the network as demonstrated in Fig. 3.2. Therefore, a solution with the optimal cost of topology discovery does not necessarily result in the overall optimal network cost.

Table 3.5 shows that the TDCCCD model generates fewer control packets per second compared to the TDCD model. As mentioned earlier, the difference between these two models is in achieving the global view of the network in controllers. Therefore, TDCD generates more number of control packets to discover the network topology. Consequently, it results in more control packets flowing over links and the links in the interference range of each link as shown in equation (3.9).
Fig. 3.2: The cost of topology discovery in the network achieved from the TDCCCD model (control packets/second)

Fig. 3.3: The total network cost achieved from the TDCCCD model (control packets/second)
Table 3.5: Optimal network cost (control packets/second) in TDCD vs. TDCCCD

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDCD</td>
<td>9.2</td>
<td>11.4</td>
<td>14.6</td>
<td>19.2</td>
<td>24.2</td>
<td>29.6</td>
</tr>
<tr>
<td>TDCCCD</td>
<td>9.2</td>
<td>6</td>
<td>5.2</td>
<td>5.6</td>
<td>6.8</td>
<td>9.2</td>
</tr>
</tbody>
</table>

Depending on the network topology, placing different numbers of controllers may result in the same network cost. For example, using the TDCCCD model, in the topology as shown in Fig. 3.4a, in case of placing 2 or 3 controllers in the network \( N = 2 \) or \( N = 3 \), the total network cost is 2.8 (control packets/second). Moreover, depending on the topology, it is possible to have more than one placement to be selected as the optimal placement. For example, in the topology as shown in Fig. 3.4b, when \( N = 1 \), both device 3 and device 4 can be selected as the optimal placement by the solver.

![Fig. 3.4: Impact of number of controllers and controller placements on the network cost](image)

3.5.2 The Impact of \( R_{Flow,Rq} \) On the Network Cost

Fig. 3.5 and Fig. 3.6 demonstrate that using the TDCCCD model and the TDCD model, if we increase the arrival rate of new flows in each device \( R_{Flow,Rq} \) in the defined scenario, since the number of control packets exchanged per second to set up new flow rules increases, the optimal solution to minimize the total cost of network is to place one controller on each device \( N = 6 \). Therefore, in case of increasing the value of \( R_{Flow,Rq} \), placing more controllers in the network decreases the cost of communications among devices and controllers and the total network cost.
Fig. 3.5: The impact of increasing the value of $R_{Flow,Rq}$ in the TDCCCD model on the network cost

Fig. 3.6: The impact of increasing the value of $R_{Flow,Rq}$ in the TDCD model on the network cost
3.5.3 The Impact of Link Capacity On the Number of Controllers

The results show that, using the proposed nonlinear optimization models: TDCD and TDCCCD, the average control overhead flowing over links is almost around the calculated network cost. To show the impact of the capacity of links assigned to exchange control packets on the number of controllers in the network using the TDCCCD model while considering different values of arrival rate of flow requests, first, we limit the assigned capacity of all links to 6 (control packets/second). In this case, the solver is not able to solve the problem for \( N = 1, 5 \text{ and } 6 \) while satisfying the capacity constraint defined in Equation (3.7). In this scenario, depending on the network requirements, only two (total cost = 6 (control packets/second)), three (total cost = 5.2 (control packets/second)) or four controllers (total cost = 5.6 (control packets/second)) can be placed on the network while satisfying the capacity constraint.

On the other hand, as shown in Fig. 3.5, if we assume that \( R_{\text{Flow,Rq}} = 4 \) (1/second), with an increase of the number of controllers in the network, the total network cost decreases such that placing six controllers has the minimum cost (10 (control packets/second)). In this case, e.g., if we limit the assigned capacity of all links to 10 (control packets/second) to exchange control packets, the solver is not able to solve the problem for \( N < 6 \) while satisfying the capacity constraint defined in Equation (3.7). In other words, the problem can be solved only for placing six controllers in the network to satisfy the capacity constraint. Therefore, depending on the capacity of links and the value of \( R_{\text{Flow,Rq}} \), the number of possible controllers to be placed in the network varies. In addition, in the proposed model, communications among controllers and devices as well as among controllers may not use the shortest paths to ensure the capacity limit of links. Therefore, a solution that would enforce or only consider communications along the shortest path would either violate the capacity constraint or not be a feasible solution. Moreover, the results show that for each solution, the average control overhead owing over wireless links in the network is almost around the network cost.

Finally, results show that when \( N = 1 \), placing a controller randomly in one of the possible controller placements in the topology shown in Fig. 3.1 instead of solving the optimization problem does not minimize the network cost. We compare the optimal network cost when placing one controller in the topology shown in Fig. 3.1 with the average cost of placing a controller in one of the six possible controller placements each time in the network. Each device \( k \) in Fig. 3.1 can be a candidate to be placed a controller \((y_k = 1)\). The optimal cost of placing one controller in the network is 9.2 (control packets/second), while the average cost of placing a controller in one of the possible placements is 12.6 (control packets/second). The results demonstrate that the proposed model indeed finds a
Table 3.6: Average number of controller-device hops and average number of inter-controller hops in the topology as shown in Fig. 3.1 using the TDCCCD model

<table>
<thead>
<tr>
<th>Number of controllers $(N)$</th>
<th>Average number of controller-device hops</th>
<th>Average number of inter-controller hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0.5</td>
<td>1.16</td>
</tr>
<tr>
<td>5</td>
<td>0.2</td>
<td>1.4</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1.53</td>
</tr>
</tbody>
</table>

Minimizing the average number controller-device hops as well as the average number of inter-controller hops in the network has a direct effect on the reliability of SDMWN, in which devices communicate in a multihop manner using shared and wireless medium, and on propagation delay which is the objective of most studies in wired and wireless CPPs. Using the TDCCCD model that generates fewer control overhead, Table 3.6 demonstrates the impact of increasing the number of controllers in the topology as shown in Fig. 3.1 on the average number controller-device and inter-controller hops. As presented in Table 3.6, with placing more controllers in the network, the network experiences higher average number of inter-controller hops while the average number of controller-device hops reduces. The results show that placing three controllers in the topology as shown in Fig. 3.1, i.e., the optimal number of controllers to minimize the network cost, see Fig. 3.3, is able to reduce both average number of hops in device-controller communications as well as average number of hops in inter-controller communications.
3.5.5 The Impact of Increasing the Number of Devices On the Execution time of the optimization problem

The results demonstrate that when we increase the number of devices in the network, it takes a long time to solve the proposed optimization problem and find the minimum number of controllers that aims to minimize the network cost. For example, it takes 0.5 (seconds) to solve the problem in a network with 3 devices, while the solver needs 153.7 (minutes) to solve the problem in a network with 6 devices. Further, as shown in Section 3.5.1, the total cost for different solutions may be close and there may be more than one optimal solution for a specific network topology and a given set of constraint. Therefore, investigating the heuristic algorithms to the proposed optimization model helps to find the near-optimal solutions in large-scale networks.

3.6 Summary

In this Chapter, two different optimization models, TDCD and TDCCCD, are proposed to solve the CPP in SDMWN while network devices are stationary. In the TDCD model, each controller communicates with all network devices to achieve the global view of the network. In the TDCCCD model, in addition to the controller-device communications to achieve the local view of the network by each controller, using the inter-controller communications, the global view of the network is stored in all controllers in the network. The results obtained from the optimization models demonstrate the impact of different factors such as the number of controllers, the capacity of wireless links and the arrival rate of new flows in devices on the generated control overhead as well as on the average number of controller-device hops and the average number of inter-controller hops in the network. Moreover, the proposed optimization solutions are able to minimize the control overhead compared to a random placement. Further, the results also show that the TDCCCD model compared to the TDCD model generates fewer number of control packets in the network.

Although formulating the CPP as an optimization problem helps to find the optimal number of controllers and determine controller placements in a network, with increasing the number of devices and controllers in the network, solving the optimization problem is computationally intensive. For example, for a network with 20 devices, even after three days, the solver on NEOS server is not able to solve the problem to place two controllers in the network. In addition, the solver may not be able to allocate sufficient memory to solve the problem. Due to the high computational complexity of the optimization models, only a small network is demonstrated for evaluation in this chapter. Therefore, in Chapter
and Chapter 5, the objective is to extend the proposed heuristic approaches described in the related work to find solutions in large-scale networks in a reasonable time compared to the time takes to solve the optimization problems. After that, Chapter 6 presents a Deep RL approach to solve the proposed CPP in SDMWN.
Chapter 4

A Genetic Algorithm for SDMWN Controller Placement Problem

4.1 Background

As discussed earlier, the objective of this thesis is to solve the CPP in SDMWN with the objective of minimizing the network cost, considering the characteristics of SDMWN. In Chapter 3, two different optimization models are proposed, the TDCD and TDCCCD models, to find a given number of controller placements and assign the controllers to the network devices in SDMWN, which aims to minimize the network cost (control packets/second). As a result, the TDCCCD model is also able to decrease the average number of controller-device hops and the average number of inter-controller hops.

The only difference between the TDCD and the TDCCCD models is in achieving the global view of the network in each controller. The results demonstrate that those two proposed optimization models are able to solve the problem while minimizing the network cost and satisfying the multiple defined constraints. In addition, the results show the effect of different metrics, including the number of controllers, the arrival rate of new traffic flows, and the capacity of links and the effect of interference on the capacity of links on the network cost. Based on the results, the TDCCCD model is able to achieve a lower network cost compared to the TDCD model.

However, since both TDCD and TDCCCD optimization models are nonlinear problems, the computational complexity is very high. As a result, although solving the optimization models finds the optimal solution in the network, when increasing the number of devices or
the number of controllers in the network, it takes a long time to obtain the optimal results. For example, even after three days, the solver, running on an Intel Core i7 CPU (3.20 GHz) and 16.0 GB RAM, is still not able to find the optimal solution in a network with 20 devices when placing two controllers. In the TDCD and TDCCCD optimization models, it is assumed that all devices can be candidates to host a controller. Therefore, to find the optimal solution, all combinations of selecting \( N \) placements from \( V \) devices are searched to find a set of controllers to minimize the network cost and satisfy the defined constraints, including the capacity of links and assigning each device to exactly one controller in the network. Moreover, to find the minimum number of controllers required to minimize the network cost, it is needed to solve the optimization models for different number of controllers from placing only one controller in the network to placing one controller on each device in the network. Consequently, due to the high computational complexity of the proposed optimization models, it is only able to investigate a small network as shown in Chapter 3, which is considered an impractical solution for SDMWN.

Various heuristic algorithms have been proposed to solve the CPP and find near-optimal solutions in a reasonable time with the aim of considering different objectives in large-scale networks [122]. The goal of this chapter is to propose a genetic algorithm (GA) based heuristic approach to solve the TDCCCD model and find a near-optimal solution in large-scale networks. The TDCCCD model is selected to be solved since it generates fewer number of control packets compared to the TDCD model as shown in Section 3.5. To evaluate the performance of the proposed GA algorithm compared to the solution based on the TDCCCD model, in this chapter, the TDCCCD model is simplified from a nonlinear to a linear problem such that all communications use the shortest paths. Section ?? describes the simplified model in more detail.

A GA [47, 48] is a meta-heuristic algorithm inspired by the natural selection theory to the optimization problems. Several research efforts have been conducted on the GA based approaches to the CPP with the aim of different objectives and constraints to show the benefits of the GA in solving the CPP reviewed in Section 2.5 [94–96,98,99,103–105]. Based on those references, GA based approaches are beneficial to find solutions that improve network performance in terms of delay among controllers and network devices and load balancing among controllers. However, existing GA-based approaches in the literature are proposed for wired networks. Therefore, in this chapter, the objective is to propose a GA-based approach and adapt the algorithm to find near-optimal solutions for the proposed CPP in SDMWN considering the characteristics of SDMWN.

The results demonstrate that the GA-based heuristic can derive the same optimal solution for a small network with much less computational overhead, and can solve larger networks in a short period of time, making it feasible for non-trivial network sizes.
In this chapter, the simplified TDCCCD model is presented in Section 4.2. Section 4.3 introduces a GA based approach to the simplified TDCCCD model. Section 4.4 evaluates the proposed heuristic algorithms and compares the obtained results with the optimal solutions in networks with different topologies and different number of devices. Finally, Section 4.5 summarizes the chapter.

4.2 Simplified TDCCCD Model

In this section, we present the simplified model of the TDCCCD with an aim to convert the CPP from a nonlinear problem to a linear problem. In the proposed linear problem, all communications among devices and controllers as well as among controllers are changed to use the shortest paths. In the TDCCCD model, the third defined binary decision variable \((f_{a,b}^{u,v})\) shows that if link \((u,v)\) is used to provide communications between device \(a\) and device \(b\) and makes the model non-linear. In the simplified TDCCCD model, this variable will be replaced with a constant value that shows the number of hops in the shortest path communication.

4.2.1 Notations

In this chapter, the notations listed in Table 4.1 are used in defining the simplified TDCCCD model.

4.2.2 Model Outputs

In the simplified TDCCCD model, since controllers and devices communicate using the shortest paths without considering the capacity of links, the outputs are the optimal placements of \(N\) controllers and controller assignments to network devices in SDMWN, which are defined as follows in the objective function.

- \(y_k\): The value equals one if and only if there is a controller placed on device \(k\).
- \(x_{k,i}\): The value equals one if and only if device \(i\) is assigned to the controller placed on device \(k\).
- Optimal cost of placing \(N\) controllers in the network.
Table 4.1: Notations used in Chapter 4 to define the simplified TDCCCD model

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y_k$</td>
<td>Output (binary decision variable): The value equals one if and only if there is a controller placed on device $k$</td>
</tr>
<tr>
<td>$x_{k,i}$</td>
<td>Output (binary decision variable): The value equals one if and only if device $i$ is assigned to a controller placed on device $k$</td>
</tr>
<tr>
<td>$\text{Cost}_{TD}$</td>
<td>The total cost of topology discovery ($control\ packets/second$) running by all controllers on their own assigned network devices and among all controllers in the network calculated using (4.2)</td>
</tr>
<tr>
<td>$R_{TD}$</td>
<td>The rate of running topology discovery by each controller ($1/second$)</td>
</tr>
<tr>
<td>$R_{Flow,Rq}$</td>
<td>The arrival rate of new flows in each device that triggers a flow request message toward the assigned controller ($1/second$)</td>
</tr>
<tr>
<td>$N$</td>
<td>A given number of controllers</td>
</tr>
<tr>
<td>$\text{neighbor}[i]$</td>
<td>A set of neighbors of network device $i$ in the network</td>
</tr>
<tr>
<td>$\text{neighbor}_{i,j}$</td>
<td>The $j^{th}$ neighbor of network device $i$</td>
</tr>
<tr>
<td>$N_{hop_{m,n}}$</td>
<td>The number of hops in the shortest path between device $m$ and device $n$</td>
</tr>
</tbody>
</table>
4.2.3 Objective Function

In the objective function presented in Equation (4.1), $Nhop_{m,n}$ is defined as the number of hops in the shortest path between device $m$ and device $n$ instead of defining $f_{m,n}^{u,v}$ introduced in Equation (3.11) that finds available links between device $m$ and device $n$ in Chapter 3.

$$\min\left(Cost_{TD} + \sum_{k=1}^{|V|} \sum_{i=1}^{|V|} (R_{Flow,Rq} \ x_{k,i} [Nhop_{k,i} + Nhop_{i,k}]) \right)$$ (4.1)

subject to:

$$(4.3), (4.4), (4.5), (4.6)$$

The first part of Equation (4.1) ($Cost_{TD}$) shows the total cost of topology discovery in the network calculated using Equation (4.2).

$$Cost_{TD} = R_{TD} \left[ \sum_{k=1}^{|V|} \sum_{i=1}^{|V|} (Nhop_{k,i} + \right.$$ 

$$\left. \sum_{m=1}^{|V|} \sum_{j=1,j\neq k,m}^{\text{neighbor}[j]} Nhop_{\text{neighbor}_{i,j,m}} x_{m,\text{neighbor}_{i,j}} x_{k,i} \right] +$$

$$R_{TD} \left[ \sum_{k=1}^{|V|} \sum_{p=1,p\neq k}^{V} (y_{k,p} \ Nhop_{k,p}) \right]$$ (4.2)

4.2.4 Constraints

The objective function presented in Equation (4.1) is subject to the following constraints defined in Section 3.3.2.

$$x_{k,i} \leq y_{k}, \ \forall i, k \in V$$ (4.3)

$$\sum_{k=1}^{|V|} x_{k,i} = 1, \ \forall i \in V$$ (4.4)
Algorithm 4.1 demonstrates the process of finding the optimal solution using the simplified TDCCCD model. In this algorithm, all combinations of selecting $N$ placements from $V$ devices are searched to find a set of controllers to minimize the network cost and satisfy the defined constraints. Further, algorithm 4.2 shows the cost function that calculates the total network cost of the possible solutions using Equation (4.1) that consists of the cost of topology discovery and the cost of exchanging configurations and setting up flow tables for a possible solution.

Algorithm 4.1 The simplified TDCCCD Algorithm

Input:

$G = (V, E), N, Constraints(e.g., capacity of links)$

1: $MinCost \leftarrow \infty$
2: for each solution $k$ in the set of possible solutions do
3: Assign the set of the controllers in $k$ to network devices
4: $\text{Cost}_k = \text{CostFunc}(G, k)$
5: if ($\text{Cost}_k < MinCost$) and ($\text{ConstraintCheck}$) then
6: $MinCost = \text{Cost}_k$
7: end if
8: end for
9: return $N$ Controller Placements, Controller Assignments, $MinCost$

4.3 Proposed GA-based Approach to the Simplified TDCCCD Model

In general, a GA-based approach starts with a given population, generated randomly. Each member of the population, called a chromosome or individual, represents a possible solution for the defined problem. In the proposed approach in this chapter, each possible solution
Algorithm 4.2 Cost Function

Function CostFunc \((G, k)\) :

1: Calculate \(Cost^k_{TD}\) \(\triangleright\) Cost of topology discovery in the network for the solution \(k\)
2: Calculate \(Cost^k_{Setup}\) \(\triangleright\) Cost of setting up flow tables and exchanging configurations
3: for the solution \(k\)
4: \(Cost_k = Cost^k_{TD} + Cost^k_{Setup}\)
5: return \(Cost_k\)

selects \(N\) devices as candidate controller placements, where \(N\) is the number of controllers in the network. In addition, in a GA algorithm, each chromosome consists of a set of genes and in this chapter, each gene represents a device number as a possible controller placement as shown in Fig. 4.1. These devices can be represented as a value from 1 to \(V\) \((a to f \in [1, |V|])\) such that \(V\) is the set of network devices. For example, Fig. 4.2 shows a feasible solution based on the objective function introduced in Equation (4.1) and constraints defined in Section 4.2.4 for placing 5 controllers in a network with 10 devices. As shown in this figure, the chromosome consists of 5 genes, i.e., the number of controllers, and the value of each gene can be from 1 to 10 since we assume that each device hosts only one controller.

Fig. 4.1: Chromosome description in the GA-based approach to the CPP in SDMWN

Fig. 4.2: A chromosome in SDMWN with 10 devices and 5 controllers

Algorithm 4.3 demonstrates the process of the proposed approach. As shown in the
algorithm, the inputs of the algorithm consist of the network graph \((G = (V, E))\), the number of controllers \((N)\), the capacity of all links in the network \((C_{(u,v)}, \forall(u,v))\), population size \((PopSize)\), GA operator probabilities \((P_c, P_m)\) and the number of generations in the network \((genNum)\).

As shown in Algorithm 4.3 (lines 3 to 5), after generating the first population randomly, the next step is to calculate the fitness of each chromosome in the population. The fitness function in the proposed approach is the cost function defined in Equation (4.1) and Algorithm 4.2. The cost of each chromosome in the population shows its quality to be selected as a solution. In Section 4.4, the impact of capacity of links on the fitness function and the feasibility of a solution are described in more detail.
Algorithm 4.3 A GA based approach to the simplified TDCCCD model

**Input:**

\[ G = (V, E), N, C(u, v) \forall (u, v), PopSize, P_c, P_m, genNum, \]

1: \( \text{Num} \leftarrow 0 \)
2: \( \text{Pop} \leftarrow \text{Generate } PopSize \text{ possible solutions (population) randomly} \)
3: \( \text{for Chromosome } k \text{ in Pop do} \)
4: \( \text{Calculate } \text{CostFunc}(G, k) \)
5: \( \text{end for} \)
6: \( \text{while } (\text{Num} < \text{genNum}) \text{ do} \)
7: \( \text{Apply selection operator and select two parents (} w_1 \text{ and } w_2 \text{) from Pop} \)
8: \( \text{ch}_1 \leftarrow w_1 \)
9: \( \text{ch}_2 \leftarrow w_2 \)
10: \( \text{Apply crossover operator to } \text{ch}_1 \text{ and } \text{ch}_2 \text{ with a probability of } P_c \)
11: \( \text{Apply mutation operator to } \text{ch}_1 \text{ and } \text{ch}_2 \text{ with a probability of } P_m \)
12: \( \text{ConstraintCheck (} \text{ch}_1 \text{)} \)
13: \( \text{CostFunc}(G, \text{ch}_1) \)
14: \( \text{ConstraintCheck (} \text{ch}_2 \text{)} \)
15: \( \text{CostFunc}(G, \text{ch}_2) \)
16: \( \text{if } \text{ch}_1 \text{ and } \text{ch}_2 \text{ satisfy the defined constraints then} \)
17: \( \text{Pop} \leftarrow \text{Replace } w_1 \text{ and } w_2 \text{ with } \text{ch}_1 \text{ and } \text{ch}_2 \text{ if the parents have a higher network cost} \)
18: \( \text{end if} \)
19: \( \text{Num}++ \)
20: \( \text{end while} \)
21: \( \text{return } \text{Number of Controllers, Controller Placements, Controller Assignments, MinCost} \)

Then, as shown in lines 6 to 20 in Algorithm 4.3, the rest of the algorithm will be iterative for a given number of times (\( genNum \)) which is also called the number of generations. In this part of the algorithm, the following operators are used to generate the next population in the algorithm as follows.

- **Selection operator:** In each iteration, the algorithm needs to select at least two parents to generate children that can be a part of the next generation. There are different types of selection operators in the GA: roulette wheel selection, tournament selection, random stochastic selection, etc. [47]. In this chapter, a roulette wheel
selection is used based on [99], in which a probability is assigned to each chromosome that is proportional to its fitness value. As a result, chromosomes with better fitness (less network cost) in the proposed approach have more chance to be selected as parents in each iteration.

- **Crossover operator**: To generate the possible next population (generation) in the GA, after selecting parents, one or two children are produced as shown in lines 8 and 9 in Algorithm 4.3. The first child is a copy of parent one and the second child is a copy of parent two. A crossover operator is applied with a probability to change the children as a mix of two selected parents. A crossover operator can be one-point, two-point or multi-point [47]. In the proposed approach, a one-point crossover is selected based on [96,104], in which for each child, with a uniform probability, one value is selected randomly from 0 to $|N| - 1$. After that, for each child, all genes with the index number less or equals to the randomly generated value, are unchanged genes from the parent and the rest of genes are replaced with those of from the other parent. Fig. 4.3 shows the crossover operator used in the proposed approach to produce two children. As depicted in this figure, the crossover point for the child one is index number 4. Therefore, all genes with index number 0 to 4 will be unchanged and the rest of genes will be replaced with gens from parent two.

- **Mutation operator** After changing the children with a crossover operator, a mutation operator is applied with a probability to change one or more genes to a randomly selected device that does not equal to other genes in the chromosome. In the proposed approach, for each child, an index number with a value from 0 to $|N| - 1$ is selected randomly to change its gene. The number is selected randomly such that the selected device is not currently assigned to host a controller in the possible solution.

In Algorithm 4.3, we assume that each device will be assigned to the closest controller. As shown in lines 12 and 14, after generating the children in each generation, the algorithm ensures that each child satisfies the constraint defined in Section 4.2.4, including the capacity constraint of links, i.e., as described in more detail in Section 4.4. If the child satisfies the defined constraints, it will be replaced with one of the parents that has a higher network cost.
4.4 Experiments, Results and Analysis

To evaluate the performance of the proposed GA algorithm presented in Section 4.3, the algorithm is run for the topology shown in Fig. 3.1 when there are 2, 3 and 4 controllers in the network (N = 2, 3 and 4) running on an Intel Core i7 CPU (1.8 GHz) and 8.0 GB RAM. The possible number of solutions to select 2, 3 and 4 controllers in this topology with 6 devices are 15, 20 and 15, respectively. The number of solutions is based on the combination of selecting $N$ controller placements from $V$ devices. Therefore, it is assumed that the number of individuals selected randomly in the population is 10 ($PopSize = 10$). Moreover, based on [101, 105], $p_c$ and $p_m$ are initialized as 1 and 0.1, respectively.

Assuming 5 generations to run the algorithm, the results show that the proposed algorithm is able to find the optimal solution when there are 2, 3 and 4 controllers in the network. In addition, the proposed algorithm is run on the topology 10 times. Compared to the execution time to solve the optimization problem presented in Section 4.2, it takes 7.5 ms, 7.3 ms and 6.5 ms to select 2, 3 and 4 controller placements and assignments in the network, respectively. Therefore, the proposed GA algorithm is significantly faster to find the solutions in the topology as shown in Fig. 3.1.

As discussed earlier in Chapter 3, for each possible solution, the average control overhead flowing over wireless links is around the network cost. If we limit the capacity of links to a certain value as an input of Algorithm 4.3, and if the network cost of each possible solution is higher than the capacity of links, the solution is not feasible and violates the capacity constraint. In this case, as mentioned in Chapter 3, based on the capacity of links,
finding the number of controllers that satisfies the capacity constraint can be achieved by the proposed model.

To evaluate the proposed GA algorithm in a network with a larger number of devices, the algorithm is run for the topology shown in Fig. 4.4, with the number of controllers ranging from 2 to 8. The same values for $p_c$ and $p_m$ are used as mentioned earlier to run the algorithm on this topology. It is assumed that the population size is 20 ($PopSize = 20$) that equals twice the number of genes (controllers) in the chromosome. The algorithm is run for different numbers of generations, ranging from 100 to 500.

Fig. 4.4: An SDMWN with 10 wireless network devices

Fig. 4.5 shows the average total network cost achieved for 25 runs for different numbers of generations with 95% confidence interval when placing different numbers of controllers in the topology in Fig. 4.4. Moreover, in comparison, this figure also shows the average total network cost for 25 runs when a random placement algorithm selects different numbers of controllers in this topology. As these results show and based on the calculated 95% confidence intervals, the differences in the results for different numbers of generations are not statistically significant. Moreover, even for a relatively low number of generations, the proposed GA algorithm outperforms the random placement algorithm in terms of the average total network cost.

Fig. 4.6 demonstrates the average number of controller-device hops and the average number of inter-controller hops when placing different numbers of controllers in the topology in Fig. 4.4. As shown in Fig. 4.6, placing more controllers in the network decreases the device-controller distances a lot while the controller-controller distance only increases slightly. Moreover, compared to a random placement algorithm, the proposed GA algorithm is able to reduce the average number of inter-controller hops in the network.
Fig. 4.5: Average total network cost in the topology in Fig. 4.4 for the proposed GA algorithm with different number of generations vs. random placement algorithm with 95% confidence interval.

Fig. 4.6: Average number of controller-device hops vs. average number of inter-controller hops in the topology as shown in Fig. 4.4.
Fig. 4.7 plots the execution time of running the proposed GA algorithm when placing different numbers of controllers and running the algorithm for 100, 300 and 500 numbers of generations with 95% confidence interval for our topology. As shown in this figure, as expected, increasing the number of generations results in higher execution times and the increase is faster than linear. In addition, based on the results presented in this figure, for a given number of generations, increasing the number of controllers to be placed leads to an increase in execution time, as the number of controllers determines the size of a chromosome and hence the execution time for chromosome operations is affected. But it increases at a very lower rate when the number of controllers is smaller.

The average execution time of running the proposed GA algorithm on networks with different numbers of devices up to 200 is shown in Fig. 4.8. As expected and similar to Fig. 4.7, increasing the number of generations results in higher execution time. In addition, as illustrated in this figure, compared to the proposed optimization model, the proposed GA algorithm can provide answers to the CPP much faster and is therefore applicable to much larger networks. That is in particular true when we can limit ourselves to a relatively low number of generations, which, as shown in Fig. 4.5, provides already relatively good placement results/network costs. Moreover, as the network scales up further, the execution time seems to grow faster than linear. More work would be needed to explore the scalability.
Fig. 4.8: Average execution time (seconds) when increasing the number of devices for different numbers of generations when $N = 2$

limits of the proposed approach in this case.

### 4.5 Summary

In this chapter, a GA-based approach is proposed for the simplified TDCCCD model with an aim to find the placement of multiple controllers in SDMWN and the assignment of controllers to network devices. The proposed GA-based approach was compared to the optimization model in SDMWN. The main objective in both approaches is to minimize the total control packets required to be exchanged in the network to discover the network state and topology, and to exchange configurations and set up flow tables in network devices.

The experimental results show that the proposed GA-based approach could generate near-optimal solution for larger-scale SDMWN in a reasonable time. A number of experiments have been performed to evaluate the GA-based approach with various values and network sizes, up to 200 nodes. The results provided insights in terms of choosing the values for different parameters for the CPP.
Chapter 5

Heuristic Algorithms for SDMWN Controller Placement Problem

5.1 Introduction

As presented in Chapter 4, the TDCCCD model is simplified to a linear problem, in which the capacity of links is not considered as a constraint to exchange control packets in the network. Therefore, all communications in the control plane use the shortest paths in the network. After that, in Chapter 4, a GA based heuristic algorithm is proposed to find near-optimal solutions for the problem. The proposed GA algorithm is able to find the placements of a given number of controllers and assign the controllers to devices in network with different number of devices, up to 200 devices.

In this chapter, the existing heuristic algorithms reviewed in Section 2.5 [40, 43–45] are extended to solve the TDCCCD model and find near-optimal solutions in large-scale networks. As mentioned in Chapter 4, the TDCCCD model is selected to be solved since it generates fewer number of control packets compared to the TDCD model as shown in Section 3.5. The results demonstrate that using ranking degrees assigned to the possible controller placements, including the average distance to other devices as a degree or the connectivity degree of each placement, the extended heuristic algorithms are able to achieve the optimal solution in small-scale networks in terms of the generated control overhead and the number of controllers selected in the network. As a result, using the extended heuristic algorithms, the average number of hops among devices and their assigned controllers as well as among controllers will be reduced. Moreover, these algorithms are able to lower the control overhead in large-scale networks and select fewer number of controllers compared
to an extended algorithm that solve the CPP in SDMWN based on a randomly selected controller placement approach.

In this chapter, Section 5.2 shows the proposed extension of the existing heuristic algorithms [40, 43-45] to the simplified optimization model. Section 5.3 evaluates the proposed heuristic algorithms and compares the obtained results with the optimal solutions in networks with different topologies and different number of devices. Finally, Section 5.4 summarizes the chapter.

5.2 Proposed Heuristic Approaches for the TDCCCD Model

This section presents an extension of the existing heuristic algorithms in the related work reviewed in Section 2.5 to find the near-optimal solutions for the proposed optimization problem defined in Section 4.2. As discussed in Section 2.5, the existing heuristic algorithms to the CPP, with the aim of different objectives and satisfying defined constraints, use different metrics such as the capacity of controllers or the characteristics of the possible controller placements to reduce the high computational complexity of the optimization problems and find near-optimal solutions in a reasonable time.

In this section, the idea of the existing heuristic algorithms introduced in [40, 43-45] is adapted to solve the simplified TDCCCD model while minimizing the network cost used in the thesis, as defined in Equation (4.1) and satisfying the defined constraints. This network cost consists of the cost of controller-device and inter-controller communications to discover the network topology as well as the cost of controller-device communications to set up flow tables and exchange configurations.

The existing heuristic algorithms in [44] and [45] select controller placements based on the connectivity degrees of devices. The connectivity degree of each device shows the number of neighbors connected to the device directly. However, the heuristic algorithm in [44] does not find the number of controllers and the network cost considered is only the controller-device communications to set up flow rules. On the other hand, the authors in [45] do not consider the control overhead generated by controller-device communications to discover the network topology. The generated control overhead in [45] consist of the control overhead injected to the network by inter-controller communications to synchronize different network views and the controller-device communications to set up flow rules.

Algorithm 5.1 adapted from the existing heuristic algorithms presented in [44] and [45] to find near-optimal solutions for the simplified TDCCCD model. The input of Algo-
Algorithm 5.1 is the network graph and the defined constraints. As a part of this algorithm, \( P \) is calculated which contains a list of devices based on their connectivity degrees in descending order. In this algorithm, the possible placement with the higher connectivity degree is selected (more neighbors) in each iteration and this placement is put into \( k \), the set of possible placements. Then, in each iteration, the algorithm finds the nearest controller in \( k \) to be assigned to each device and the network cost is calculated using Equation (4.1). If the calculated network cost increases, the placement is removed from \( k \). Unlike [44], Algorithm 5.1 finds the number of controllers, i.e., the iteration continues until adding a new controller increases the network cost. The outputs of the Algorithm 5.1 includes the number of controllers, the placements of the controllers, controller assignments to network devices and the minimum achieved cost.

**Algorithm 5.1** Connectivity-degree based heuristic algorithm adapted from [44] and [45]

**Input:**

\[ G = (V, E), \text{Constraints} \]

1. \( P[] \leftarrow \text{DescendingSorted}(V, \text{ConnectivityDegree}(V)) \)
2. \( \text{MinCost} \leftarrow \infty \)
3. \( i \leftarrow 0 \)
4. **do**
5. \( \text{Add placement } P[i] \text{ to the set of possible placements } k \)
6. \( \text{Assign the set of the controllers in } k \text{ to network devices} \)
7. \( \text{Cost}_k = \text{CostFunc}(G, k) \)
8. 
9. **if** \( \text{(Cost}_k > \text{MinCost}) \) **then**
10. \( \text{Remove } p[i] \text{ from the set of possible placements } k \)
11. **end if**
12. \( i++ \)
13. **while** \( \text{(Cost}_k < \text{MinCost}) \)
14. **return** Number of Controllers, Controller Placements, Controller Assignments, Min-Cost

The authors in [40] propose the average delay as a degree that is proportional to the distance among devices. The main idea of the approach is to calculate the average distance among the possible placements to all other devices in the network as the ranking degree. The algorithm presented in [40] does not find the number of controllers and it only finds the controller placements in the network. Hence, this thesis proposes an extension of this approach to the simplified TDCCCD model.
Algorithm 5.2 is the modification of [40] such that the input of the algorithm is the network graph and the defined constraints. As shown in line (1) of the algorithm, $P$ is calculated which is the list of devices based on their average distance degrees to other devices in ascending order. To find the number of controllers, in each iteration, a placement with the lowest distance degree is selected ($P[i]$) and added to the set of controllers ($k$). Then, each device is assigned to the nearest controller and the network cost is calculated using Equation (4.1). If adding the new controller decreases the network cost, another iterations continues; otherwise, the new controller is removed from the set of controllers and the iteration stops. The outputs of Algorithm 5.2 are controller placements and assignments, and the minimum cost of SDMWN are returned as results.

Algorithm 5.2 Distance-degree based heuristic algorithm adapted from [40]

**Input:**

$G = (V, E)$, Constraints

1. $P[] \leftarrow \text{AscendingSorted}(V, \text{AvgDistanceDegree}(V))$
2. $\text{MinCost} \leftarrow \infty$
3. $i \leftarrow 0$
4. do
5. Add placement $P[i]$ to the set of possible placements $k$
6. Assign the set of the controllers in $k$ to network devices
7. $\text{Cost}_K = \text{CostFunc}(G, k)$
8. if $(\text{Cost}_k > \text{MinCost})$
9. Remove $p[i]$ from the set of possible placements $k$
10. end if
11. $i++$
12. while $(\text{Cost}_k < \text{MinCost})$
13. return Number of Controllers, Controller Placements, Controller Assignments, Min-Cost

Finally, to investigate the effect of the ranking degrees to select the controllers compared to a random placement, the heuristic algorithm presented in [43] is extended such that in the simplified TDCCC model, in the first iteration, all devices are added to the set of possible controller placements. Then, in each iteration, one of the placements is removed randomly from the set of controllers. All devices assigned to the removed controller placement are assigned to the closest controller in the set of controllers. If removing the
new controller reduces the network cost, the iteration continues; otherwise, the controller is not removed from the set of the controllers and the algorithm stops. Algorithm 5.3 shows the extension of the proposed algorithm in [43] which removes the possible placements randomly from the set of controllers in each iteration. Algorithm 5.3 finds the number of controllers, controller placements and assignments as well as the minimum cost.

**Algorithm 5.3** Randomly-selected heuristic algorithm adapted from [43]

**Input:**

\[ G = (V, E), N, \text{Constraints} \]

1: \( \text{MinCost} \leftarrow \infty \)
2: Add all placements in \( V \) to the set of possible controller placements \( k \)
3: Assign the set of the controllers \( k \) to network devices
4: do
5: Select a placement \( i \) randomly from \( k \)
6: Remove \( i \) from the set of possible placement \( k \)
7: Assign the set of the controllers in \( k \) to network devices
8: \( \text{Cost}_k = \text{CostFunc}(G, k) \)
9: if \( \text{Cost}_k > \text{MinCost} \)
10: Add \( i \) to the set of possible placements \( k \)
11: end if
12: while \( \text{Cost}_k < \text{MinCost} \)
13: return Number of Controllers, Controller Placements, Controller Assignments, Min-Cost

The computational complexity of the presented algorithms, i.e., Algorithms 5.1, 5.2 and 5.3 is less than solving the simplified TDCCCD model that needs to search all possible combinations of selecting \( N \) placements from \( V \) devices, which is a NP-hard problem [35]. However, the reviewed algorithms [40, 43–45] and their extensions, i.e., Algorithms 5.1, 5.2 and 5.3, do not consider the capacity of links as a constraint and the communications in the control plane use the shortest paths. Section 5.3 investigates the impact of the limited capacity of links to exchange control packets on the obtained number of controllers in the network.
5.3 Results and Analysis

As mentioned earlier, the proposed optimization models in Chapter 3 are nonlinear problems and due to the high computational complexity of the optimization models, only a small network is investigated. Therefore, to evaluate the performance of the three proposed algorithms in Section 4.2, including the connectivity-degree based, the distance-degree based and the randomly-selected compared to the optimal solution, the simplified TDCCCD model presented in Section 4.2 is used. A number of experiments have been performed for evaluation. Randomly generated topologies with different numbers of devices are used and the proposed algorithms are run on each topology 10 times to calculate the average network cost. Moreover, AMPL (a mathematical programming language) [117] is used running on the NEOS server, i.e., as described in Section 4.4 and the solver is CPLEX [81] to implement the simplified TDCCCD model. In addition, a system with the following configurations is used to run the proposed heuristic algorithms: an Intel Core i7 CPU (1.8 GHz) and 8.0 GB RAM. To evaluate the optimization problem and the proposed heuristic algorithms, it is assumed that $R_{FlowRq}$ is 0.5 (1/second) and $R_{TD}$ is 0.2 (1/second).

Fig. 5.1 and Fig. 5.2 demonstrate two randomly generated SDMWNs with 10 wireless network devices. In these figures, the devices with the red circle are the optimal controller placements with respect to network cost. Algorithm 5.1 and Algorithm 5.2 are able to find the optimal number of controllers, which is 5 controllers in Topology 1 and the optimal controller placements, which are devices 1, 2, 3, 4 and 9, while minimizing the network cost (13.8 (control packets/second)). In addition, Algorithm 5.1 and Algorithm 5.2 are able to find the optimal number of controllers and controller placements in Topology 2, which is 6 controllers placed on devices 1, 2, 3, 4, 7 and 10, while minimizing the network cost (13.2 (control packets/second)). On the other hand, Algorithm 5.3 produces a higher network cost for both topologies: 15.0 (control packets/second) is the minimum network cost for the topology as shown in Fig. 5.1 and 15.6 (control packets/second) is the minimum network cost for the topology as shown in Fig. 5.2.

Table 6.5 demonstrates the average minimum cost achieved from each of the three extended algorithms and the linear optimization problem in randomly generated topologies with different numbers of devices. As shown in this table, for small-scale networks, Algorithm 5.1 based on the connectivity degree and Algorithm 5.2 based on the distance degree are able to find the same solution as the optimal solution obtained from solving the optimization problem for 10 or 20 devices. The results generated from Algorithm 5.1 and Algorithm 5.2 are mostly close to that of the optimal solution, whereas Algorithm 5.3 generally has a higher cost. In addition, Fig. 5.3 shows the number of selected controllers in SDMWN in the optimal solution and the results of the proposed heuristic algorithms

72
Fig. 5.1: Topology 1- an SDMWN with 10 wireless network devices

Fig. 5.2: Topology 2- an SDMWN with 10 wireless network devices
when increasing the number of devices in the network. As demonstrated in Table 6.5, when increasing the number of devices in the network, the network cost achieved from the optimal solution and the proposed heuristic algorithms are different that shows the impact of selecting the right number of controllers and controller placements on the network cost.

Table 5.1: Minimum cost obtained from the proposed algorithms and the optimal solution from the simplified TDCCCD model (control packets/second)

<table>
<thead>
<tr>
<th>Number of devices</th>
<th>Optimal Solution</th>
<th>Algorithm 5.1</th>
<th>Algorithm 5.2</th>
<th>Algorithm 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>13.8</td>
<td>13.8</td>
<td>13.8</td>
<td>15</td>
</tr>
<tr>
<td>20</td>
<td>42.6</td>
<td>42.6</td>
<td>42.6</td>
<td>54</td>
</tr>
<tr>
<td>30</td>
<td>79.2</td>
<td>79.6</td>
<td>80.4</td>
<td>88</td>
</tr>
<tr>
<td>40</td>
<td>126</td>
<td>149.8</td>
<td>131.4</td>
<td>143.8</td>
</tr>
<tr>
<td>50</td>
<td>186.4</td>
<td>199.4</td>
<td>194.8</td>
<td>223.8</td>
</tr>
<tr>
<td>60</td>
<td>219.8</td>
<td>274</td>
<td>252</td>
<td>286.6</td>
</tr>
</tbody>
</table>

Table 5.2 shows the minimum cost achieved from the proposed heuristic algorithms in large-scale networks up to 500 network devices. Due to the high computational complexity of the simplified TDCCCD model, the optimization problem is solved for only 10 to 60 devices in the network. As depicted in Table 6.5 and Table 5.2, both Algorithm 5.1 and Algorithm 5.2 generate lower cost consistently compared to Algorithm 5.3. Algorithm 5.1 is based on the average connectivity degree. Intuitively, if a device has high connectivity degree, it has a higher chance to be selected as a controller. In this case, the selected controller can directly communicate with more network devices without going over multiple hops, which is more effective in the network cost. Algorithm 5.2 is based on the average distance degree. A device with lower average distance degree is more likely to be selected as a controller. Similar to Algorithm 5.1, in Algorithm 5.2, the selected devices as controllers are likely to be able to directly communicate with network devices; hence, multi-hop communications between a controller and network devices would be reduced. On the other hand, Algorithm 5.3 is based on random placement, which may result in lower cost effective placements.

In the reviewed algorithms [40, 43–45] and their extensions, i.e., Algorithms 5.1, 5.2
Fig. 5.3: The number of devices versus the number of controllers selected in the presented algorithms and the TDCCCD model

and 5.3, communications in the control plane use the shortest paths. However, the results from Chapter 3 show that using the proposed nonlinear optimization models, the average control overhead flowing over links is almost around the network cost achieved by Algorithms 5.1, 5.2 and 5.3. For instance, in topology 1 as shown in Fig. 5.1, the average control overhead flowing over links is around 13.8 (control packets/second), which is the minimum cost obtained from running the simplified TDCCCD model. If we consider the capacity constraint of links and limit the capacity of all links to a certain value, using the network cost achieved from the proposed heuristic algorithms and the generated placement results, we are able to find the number of controllers that satisfies the capacity constraint. In this case, when placing a given number of controllers, if the heuristic algorithms or the simplified TDCCCD model result with higher cost than the limited capacity, the solution then is not able to satisfy the capacity constraint. For example, when placing 5 controllers in the topology as shown in Fig. 5.1, the minimum cost achieved is 13.8 (control packets/second). In this scenario, if we limit the capacity of all links to 12 (control packets/second) in the network, there is no feasible solution to place 5 controllers in this network while satisfying the capacity constraint.

In addition, Table 5.3 demonstrates the computational time of solving the simplified TDCCCD model and finding the near-optimal solution using the proposed heuristic algo-
Table 5.2: Minimum cost obtained from the proposed heuristic algorithms in large-scale networks (control packets/second)

<table>
<thead>
<tr>
<th>Number of devices</th>
<th>Algorithm 5.1</th>
<th>Algorithm 5.2</th>
<th>Algorithm 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>284.6</td>
<td>337</td>
<td>368.6</td>
</tr>
<tr>
<td>80</td>
<td>420</td>
<td>408.8</td>
<td>460.2</td>
</tr>
<tr>
<td>90</td>
<td>557</td>
<td>481.8</td>
<td>629.8</td>
</tr>
<tr>
<td>100</td>
<td>763.4</td>
<td>539</td>
<td>801.6</td>
</tr>
<tr>
<td>150</td>
<td>1091.4</td>
<td>1098.6</td>
<td>1604.7</td>
</tr>
<tr>
<td>200</td>
<td>1723.6</td>
<td>1930</td>
<td>2204.5</td>
</tr>
<tr>
<td>300</td>
<td>4035.6</td>
<td>4076.2</td>
<td>4303.4</td>
</tr>
<tr>
<td>400</td>
<td>6424.8</td>
<td>7111.4</td>
<td>7308.9</td>
</tr>
<tr>
<td>500</td>
<td>10122.6</td>
<td>10675</td>
<td>12029.4</td>
</tr>
</tbody>
</table>

As shown in Table 5.3, when the number of devices in the network increases, it takes a long time to solve the simplified TDCCCD model and find the minimum number of controllers that aims to minimize the network cost. Algorithms 5.1, 5.2 and 5.3 are able to find a near-optimal solution faster than the simplified TDCCCD model. Table 5.4 depicts that when increasing the number of devices in the large-scale network (70 to 500 devices), the time saving for Algorithm 5.1 is significant compared to the other two algorithms. The main reason for this is that Algorithm 5.1 calculates the connectivity degree of the possible placements which can be completed faster than calculating the average distance degree values from the possible placements to other devices in the network used in Algorithm 5.2. Also, Algorithm 5.3 needs to evaluate a large number of possible solutions to stop the iterations. Table 5.4 shows the running time of the proposed heuristic algorithms when there are up to 500 devices in the network. This table demonstrates that, the algorithms are able to find near-optimal solutions in large-scale networks very efficiently.
Table 5.3: Execution time of the simplified TDCCCD model vs. execution time of the proposed heuristic algorithms (second)

<table>
<thead>
<tr>
<th>Number of devices</th>
<th>Optimal Solution</th>
<th>Algorithm 5.1</th>
<th>Algorithm 5.2</th>
<th>Algorithm 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.4484</td>
<td>0.0086</td>
<td>0.0120</td>
<td>0.0042</td>
</tr>
<tr>
<td>20</td>
<td>21.1262</td>
<td>0.0109</td>
<td>0.0163</td>
<td>0.0105</td>
</tr>
<tr>
<td>30</td>
<td>95.5204</td>
<td>0.0133</td>
<td>0.0220</td>
<td>0.0171</td>
</tr>
<tr>
<td>40</td>
<td>300.3674</td>
<td>0.0154</td>
<td>0.0272</td>
<td>0.0226</td>
</tr>
<tr>
<td>50</td>
<td>1062.1368</td>
<td>0.0163</td>
<td>0.0413</td>
<td>0.0348</td>
</tr>
<tr>
<td>60</td>
<td>1889.29</td>
<td>0.0172</td>
<td>0.0419</td>
<td>0.0382</td>
</tr>
</tbody>
</table>

Table 5.4: Execution time of the proposed heuristic algorithms in large-scale networks (second)

<table>
<thead>
<tr>
<th>Number of devices</th>
<th>Algorithm 5.1</th>
<th>Algorithm 5.2</th>
<th>Algorithm 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>0.01754</td>
<td>0.04554</td>
<td>0.05268</td>
</tr>
<tr>
<td>80</td>
<td>0.01845</td>
<td>0.05377</td>
<td>0.07014</td>
</tr>
<tr>
<td>90</td>
<td>0.01954</td>
<td>0.06315</td>
<td>0.09617</td>
</tr>
<tr>
<td>100</td>
<td>0.02313</td>
<td>0.07222</td>
<td>0.11397</td>
</tr>
<tr>
<td>150</td>
<td>0.03531</td>
<td>0.18085</td>
<td>0.19192</td>
</tr>
<tr>
<td>200</td>
<td>0.06372</td>
<td>0.48571</td>
<td>0.39283</td>
</tr>
<tr>
<td>300</td>
<td>0.13912</td>
<td>0.94347</td>
<td>1.03180</td>
</tr>
<tr>
<td>400</td>
<td>0.17023</td>
<td>1.44209</td>
<td>1.64672</td>
</tr>
<tr>
<td>500</td>
<td>0.28923</td>
<td>3.39799</td>
<td>3.02576</td>
</tr>
</tbody>
</table>
Fig. 5.4 demonstrate the network cost, the average number of hops between devices and their assigned controllers (CD) and among controllers (CC) in SDMWN for the optimal solution and the proposed heuristic algorithms, respectively, in small-scale networks. The number of selected controllers of optimal solution and the three proposed algorithms are shown in Fig. 5.3. As demonstrated in Fig. 5.4, although Algorithm 5.3 achieves the lower average number of hops among devices and their assigned controllers compared to other approaches, the number of selected controllers and the controller placements and assignments using Algorithm 5.3 result in higher average number of hops among controllers and network cost. In addition, the results show that in some cases, e.g., 50 network devices, although Algorithm 5.1 and Algorithm 5.2 achieve the lower average number of hops among controllers compared to the optimal solution, the controller placements and assignments in these two algorithms result in higher network cost and average number of hops among devices and their assigned controllers as demonstrated in Fig. 5.4.

Fig. 5.5 shows the number of selected controllers in Algorithms 5.1, 5.2 and 5.3. In addition, as presented in Fig. 5.5, in the randomly generated topologies with more than 150 devices, Algorithm 5.1 and 5.3 are able to achieve lower average number of hops between devices and their assigned controllers. Moreover, as shown in Table 5.2, when increasing the number of devices in the network, Algorithm 5.1 is able to achieve lower network cost compared to other two algorithms, while Algorithm 5.3 obtains higher network cost. Fig. 5.6 and Fig. 5.7 demonstrate the average number of hops between devices and their assigned controllers and the average number of hops among controllers, respectively when increasing the number of devices in the network. As shown in Fig. 5.5, although in some cases, Algorithm 5.3 finds fewer number of controllers in SDMWN, the average number of hops among controllers is higher compared to Algorithms 5.1 and 5.2 as shown in Fig. 5.7.

<table>
<thead>
<tr>
<th>Number of devices</th>
<th>Optimal Solution</th>
<th>Algorithm 5.1</th>
<th>Algorithm 5.2</th>
<th>Algorithm 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>13.8</td>
<td>1.0</td>
<td>1.0</td>
<td>13.8</td>
</tr>
<tr>
<td>20</td>
<td>42.6</td>
<td>1.0</td>
<td>1.0</td>
<td>42.6</td>
</tr>
<tr>
<td>30</td>
<td>79.6</td>
<td>1.0</td>
<td>1.2</td>
<td>79.6</td>
</tr>
<tr>
<td>40</td>
<td>126.0</td>
<td>1.0</td>
<td>1.38</td>
<td>149.8</td>
</tr>
<tr>
<td>50</td>
<td>186.4</td>
<td>1.0</td>
<td>1.14</td>
<td>199.4</td>
</tr>
<tr>
<td>60</td>
<td>219.8</td>
<td>1.0</td>
<td>1.91</td>
<td>274.0</td>
</tr>
</tbody>
</table>

Fig. 5.4: Optimal solution vs. the proposed heuristic algorithms
Fig. 5.5: The number of devices versus the number of controllers selected in the presented algorithms in large-scale network

Fig. 5.6: The number of devices versus the average number of hops between devices and their assigned controllers in the presented algorithms in large-scale network
Fig. 5.7: The number of devices versus the average number of hops between controllers in the presented algorithms in large-scale network

5.4 Summary

In this chapter, the heuristic algorithms presented in the related work [40, 43–45] are extended to the CPP while minimizing the control overhead or the network cost to find near-optimal solution for the simplified TDCCCD model.

The results obtained from the three proposed heuristic algorithms and the linear optimization model show that, using a ranking algorithm based on the connectivity degree of the possible placements (Algorithm 5.1) or the degree of average distance of the placements to other devices (Algorithm 5.2), two proposed heuristic algorithms are able to find optimal solutions in small-scale networks. Moreover, when increasing the number of devices, these two proposed heuristic algorithms are able to select the better number of controllers and controller placements, which results in lower network cost and average number of hops among controllers compared to the third proposed algorithm (Algorithm 5.3). Algorithm 5.3 selects the possible placements randomly. Using the results obtained from the proposed heuristic algorithms, we are able to find the number of controllers that satisfy the capacity of links constraint defined in the proposed optimization model. Moreover, the results showed that the computational times for all three heuristic algorithms are significantly lower than that of the optimal solution, which make them more practical. Among those three heuristic algorithms, Algorithm 5.1 consistently requires less computational time which is based on calculating connective degree values of the possible
placements. In addition, the proposed heuristic algorithms in this chapter are faster than the proposed GA-based heuristic algorithm in Chapter 4 and these algorithms are also able to find the number of controllers compared the proposed GA algorithm that requires to run for different number of controllers.
Chapter 6

Using Deep Reinforcement Learning for SDMWN Controller Placement Problem

6.1 Introduction

As presented in Chapter 4 and Chapter 5, different heuristic algorithms are proposed to find near optimal solutions for the simplified TDCCCD model presented in Section 4.2, including the connectivity-degree based, the distance-degree based, the randomly-selected and the GA based approaches.

Recently, deep reinforcement learning (Deep RL) \cite{49, 50, 52} has gained increasing attention to apply to different types of problems using the combination of deep learning \cite{123} and reinforcement learning (RL) \cite{51}. Deep learning, known as deep neural network, is a subset of machine learning inspired by the structure of the human’s brain. In RL, using observations from the environment and a Q-table, an agent maps the situations in the environment to the actions to maximize the achieved reward. Deep RL is able to overcome the existing challenges of RL, including the low-dimensional of status-action representations in Q-table based on the advantages of neural networks. In this case, using the training data, Deep RL is able to learn and make predictions for the new data sets that can be beneficial in dynamic environments and the environments with a high-dimensional status-action.

Based on the related work reviewed in Section 2.6, applying Deep RL have the potential to find the controller placements and assignments in a network with a dynamic
environment. The objective of this chapter is to investigate the potential of applying Deep RL to solve the proposed CPP in SDMWN with the objective of minimizing the network cost. Therefore, to find the placements of $N$ controllers and assign the controllers to the network devices, based on [108], a deep Q-network (DQN) [124] is proposed that is a value-function-based Deep RL algorithm introduced to solve the ATTARI games. DQN is able to overcome the aforementioned challenges in RL in an environment with a high-dimensional status-action space and to approximate the Q-value function for all possible actions using deep learning. Section 6.2 describes a DQN algorithm in more detail.

The rest of this chapter is organized as follows. Section 6.2 introduces the proposed Deep RL to the proposed simplified TDCCCD model presented in Section 4.2. The evaluation of the proposed Deep RL is demonstrated in Section 6.3 and Section 6.4 summarizes the chapter.

### 6.2 Adaptation of Deep RL for the proposed CPP

To solve a problem using a Deep Q-network (DQN) algorithm [124], we need to describe different parameters and definitions which are listed as follows [49,50,52].

- **Environment:** The main part of a problem that needs to be defined in the DQN algorithm is the environment that generates the input data. The DQN algorithm is adapted in this chapter to the CPP in SDMWN; the environment is an SDMWN with a number of wireless network devices.

- **State Space ($S$):** It represents the set of possible situations in the environment. Equation (6.1) shows the set of states in the proposed problem in this chapter that consists of the state of all network devices in a network with $V$ devices. As shown in Equation (6.2) and Equation (6.3), the state of device $i$ demonstrates if this device is assigned as a controller to other devices in the network ($x_{i,j}$ is defined in Table 4.1).

\[
S = \{(s_1, s_2, ..., s_{|V|})\} \quad (6.1)
\]

\[
s_i = \{w_i^1, w_i^2, ..., w_i^{|V|}\} \quad (6.2)
\]

\[
w_i^j = \begin{cases} 
1, & \text{if } x_{i,j} = 1 \\
0, & \text{otherwise}, 
\end{cases} \quad \forall i, j \in V, \text{ s.t. } \sum_{i=1}^{|V|} x_{i,j} = 1 \quad (6.3)
\]
\( \text{Action Space (A):} \) It represents the possible actions that can be taken in a defined environment. Using the current observation and DQN, a sequence of actions will be applied that can change the state of the environment. Equation (6.4) shows the action space in an environment that consists of the actions applied at various particular times. As shown in Equation (6.5) and Equation (6.6), in the proposed algorithm in this chapter, a possible action in a particular time demonstrates the network devices that will be selected as controllers. In this case, the action changes the current state of the environment and the selected controllers will be assigned to the devices in the network (\( y_j \) is defined in Table 4.1).

\[
A = \{a_t, a_{t+1}, a_{t+2}...\} \tag{6.4}
\]

\[
a_t = \{b^1_t, b^2_t, ..., b^{|V|}_t\} \tag{6.5}
\]

\[
b^j_t = \begin{cases} 
1, & \text{if } y_j = 1 \\
0, & \text{otherwise, } \forall j \in V, \ s.t. \ \sum_{j=1}^{|V|} y_j = N 
\end{cases} \tag{6.6}
\]

\( \text{Agent:} \) Another essential part of DQN is an agent that is responsible for making decisions and taking a sequence of actions in the environment. Based on [108], in the proposed DQN in this chapter, it is assumed that the agents are devices.

\( \text{Reward (R):} \) When the agent takes an action, the environment obtains a reward. \( R \) shows the set of rewards obtained using DQN during the running time. In the proposed DQN, the reward is the network cost of the possible solution demonstrated by the action and the new states.

\( \text{Transition:} \) In DQN, when an agent takes an action \( a_t \) at the particular time \( t \) in the environment, a transition from state \( s_t \) to state \( s_{t+1} \) happens in the environment that obtains reward \( r_t \).

\( \text{Q-function:} \) Using the current state in the environment, DQN approximates the action-value function referred to as a Q-function that shows the reward of all possible actions using the concept of neural networks.

\( \text{Replay Memory:} \) In DQN, during the running time, the transitions will be saved in a replay memory. After that, by sampling random stored transitions called mini-batch, a DQN is able to train the network.
**Exploration and Exploitation:** In DQN, an agent is able to obtain experience using exploration and exploitation strategies. Using the exploration strategy, the agent is able to explore and learn the environment by taking random actions and achieving the rewards. While in the exploitation strategy, using the current knowledge, DQN maximizes the reward. In DQN, a parameter called epsilon (ε) is the exploration rate that shows the probability of exploring the environment by the agent. During the running time of DQN, this exploration rate will be decayed using a defined decay rate to increase the probability of exploiting strategy.

**Q-network and Target Q-network:** In DQN, two deep neural networks are created such that Q-network retrieves the Q-function as shown in Equation (6.8) and target Q-network calculates the target value as shown in Equation (6.7) using the training process. Target Q-network and its weights (θ') will be updated by Q-network (θ = θ) in an interval of time to make the learning process stable. In Equation (6.7), \( r(s, a, s') \) shows the obtained reward at state \( s \) when an action \( a \) is selected that changes the state to \( s' \). In addition, \( \gamma \in [0,1] \) is the discount factor that demonstrates the importance of the future rewards for the agent such that decreasing the rate will reduce the effect of future rewards on the target value. In this equation, \( \alpha \in (0,1] \) shows the learning rate such that the larger learning rate value demonstrates the stronger effect of new data for updating the Q-function as shown in Equation (6.8).

\[
\text{target}_Q \leftarrow r(s, a, s') + \gamma \max_a Q(s', a'; \theta) \tag{6.7}
\]

\[
Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[\text{target}_Q] \tag{6.8}
\]

**Policy:** In DQN, a policy shows how the agent takes actions to maximize the reward. In the proposed DQN in this chapter, the objective is to minimize the network cost in the network calculated using Equation (4.1).

Algorithm 6.1 shows the process of the proposed DQN algorithm for the simplified TDCCCD model. After initializing the variables, Q-network and target Q-network, in each episode in DQN, using the initial observation from the environment and the exploration rate (ε), the agent decides to take an action. Using the value of ε, the agent is able to take a random action from the action space or predict an action using the maximum Q-function value based on the current knowledge. Then, the selected action will be executed in the environment and the next state and the reward will be obtained. As demonstrated in
line (12) of algorithm 6.1, executing action \( a \) also returns a value for \( done \) which is the condition of the \( while \) loop in line (7). The value of \( done \) can be returned as \( True \) if the algorithm reaches a predefined goal reward or if the \( while \) loop runs for a predefined number of iterations. The new transition will be stored in the relay memory as shown in line (13).

At the next step, if there are enough samples in the replay memory, the training phase starts, including sampling random minibatch from the memory and calculating the target Q-function \( (\text{target}_Q) \) and updating Q-function \( (\text{predicted}_Q) \). Then, the gradient descent is calculated to find the loss. As mentioned before, the target Q-network is updated in an interval of time. In the next step, the new state will be the current state in the environment. As shown in lines (22) to (24), in each episode, the value of the exploration rate \( (\varepsilon) \) will be decayed by a decay rate \( (\varepsilon_{\text{decay}}) \) but this value cannot be less than a pre-defined minimum rate \( (\varepsilon_{\text{min}}) \).
Algorithm 6.1 A DQN algorithm to the simplified TDCCCD model
Input:
\[ G = (V, E), N, N_{ep}, C_{RM}, minC_{RM}, \varepsilon, \varepsilon_{decay}, \varepsilon_{min}, \gamma, \alpha, minbatch_{size} \]

1: Initialize the replay memory capacity \( C_{RM} \)
2: Initialize Q-function with random weight \( \theta \)
3: Initialize target Q-network with weights \( \theta' = \theta \)
4: for episode 1 to \( N_{ep} \) do
5: \( done \leftarrow \) False
6: Initialize the first state in the environment \( s \)
7: while \( ! done \) do
8: \( n \leftarrow \) generate a random number
9: if \( n < \varepsilon \) then
10: Select a random action \( a_t \) from the action space \( A \)
11: else \( a = \text{argmax}_a Q(s, a; \theta) \)
12: Execute action \( a \) to observe the next state \( s' \) and reward \( r \) and return \( done \)
13: Store transition \( (s, a, r, s') \) in the replay memory
14: if \( bufferSize > minC_{RM} \) then
15: Sample random minibatch from the replay memory \( (\text{minbatch}_{size}) \)
16: Set \( targetQ = \begin{cases} r(s, a, s') & \text{if done == True} \\ r(s, a, s') + \gamma \text{max}_a Q(s', a'; \theta) & \text{otherwise} \end{cases} \)
17: Update \( Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha targetQ \)
18: Perform the gradient step on \( (targetQ - predictedQ)^2 \)
19: reset \( Q' \) every \( C \) steps by setting \( \theta' = \theta \)
20: end if
21: \( s \leftarrow s' \)
22: end while
23: if \( \varepsilon > \varepsilon_{min} \) then
24: \( \varepsilon = \varepsilon_{decay} \)
25: \( \varepsilon = \max(\varepsilon_{min}, \varepsilon) \)
26: end if
27: end for
28: return Controller Placements, \( N \) Controller Assignments, MinCost
Table 6.1: Values used for the defined parameters in the proposed DQN

<table>
<thead>
<tr>
<th>Parameter’s Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_{RM}$</td>
<td>Size of the replay memory</td>
<td>50000</td>
</tr>
<tr>
<td>$minC_{RM}$</td>
<td>Minimum size of the replay memory to start training</td>
<td>50</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>Exploration rate</td>
<td>1</td>
</tr>
<tr>
<td>$\varepsilon_{decay}$</td>
<td>Decay rate</td>
<td>0.99975</td>
</tr>
<tr>
<td>$\varepsilon_{min}$</td>
<td>Minimum epsilon value</td>
<td>0.001</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Discount factor</td>
<td>0.99</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>Learning rate</td>
<td>0.01</td>
</tr>
<tr>
<td>$minbatch_{size}$</td>
<td>minbatch size for training</td>
<td>20</td>
</tr>
</tbody>
</table>

6.3 Experiments, Results and Analysis

To evaluate the performance of the proposed DQN for the proposed CPP in SDMWN, first, the topology as shown in Fig. 3.1 is considered. The values used in this evaluation for the defined parameters in DQN are listed in Table 6.1. The proposed models in the related work [106,108,110] do not provide details about the values used for the evaluation. Therefore, in this evaluation, the values are selected from the possible value range for each parameter. To run the proposed DQN algorithm, a system with the following configurations is used: an Intel Core i7 CPU (1.8 GHz) and 8.0 GB RAM.

Table 6.2 shows the optimal results for the topology shown in Fig. 3.1 compared to the results from running the proposed DQN algorithm when $N = 2$ and $N = 3$, and $N_{ep} = 30$. As demonstrated in this table, the results are slightly different from the optimal solutions.

Fig. 6.1 demonstrates the execution time of running the proposed DQN algorithm when increasing the number of episodes. As discussed earlier in Section 6.2, there are different factors that have a direct effect on the execution of the DQN, including the number of episodes, the number of iterations for the while loop and the goal reward. Considering a constant number of iterations, Fig. 6.1 shows that increasing the number of episodes results in significantly higher execution time. Moreover when $N = 2$, the execution time is slightly lower compared to the execution time when $N = 3$. 

88
Table 6.2: Optimal results vs. the results from the proposed DQN algorithm when $N_{ep} = 30$

<table>
<thead>
<tr>
<th>$N$</th>
<th>Optimal solution</th>
<th>DQN algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost (control packs/seconds)</td>
<td>Avg. num of controller-device hops</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>5.2</td>
<td>1</td>
</tr>
</tbody>
</table>

Fig. 6.1: Execution time for running the proposed DQN algorithm for $N = 2$ and $N = 3$ when increasing the number of episodes
To evaluate the performance of the proposed DQN in a network with more number of devices, the algorithm is run for different number of devices from 10 to 60 when $N_{ep} = 30$ to avoid excessive execution time. Fig. 6.2 shows the execution time of running the algorithm, considering the same values for the parameters as shown in Table 6.1. As demonstrated in this figure, running the algorithm when there are 30 devices in the network is faster while in a network with 60 devices, it takes longer to run the proposed DQN algorithm.

![ Execution time for running the proposed DQN when increasing the number of devices in the network for $N_{ep} = 30$](image.png)

To evaluate the performance of the adapted DQN in a network with larger number of devices, a topology shown in Fig. 6.3 is considered with 10 wireless devices. Table 6.3 shows the optimal results for the topology shown in Fig. 6.3 compared to the results from running the adapted DQN algorithm presented in Algorithm 6.1 when $N = 3$ and $N_{ep} = 30$. As demonstrated in this table, there is only one difference between the selected controller placements in the optimal solutions from the simplified TDCCCD model and the solutions of the DQN such that the optimal placements are devices 1, 3 and 9, while DQN selects devices 1, 4 and 9 as controllers. Based on the results shown in Table 6.3, the optimal network cost and the cost achieved from the DQN algorithm are slightly different.

Table 6.4 presents the selected controllers in the simplified TDCCCD model and the adapted DQN. As shown in this table, the selected controllers in both solutions are different in only one controller placement for 10 and 20 devices. Table 6.5 demonstrates that based on the selected controller placements, when increasing the number of devices in the network, compared to the optimal solutions, the adapted DQN algorithm is able to reduce the
Fig. 6.3: An SDMWN with 10 wireless network devices

Table 6.3: Optimal results vs. the results from the adapted DQN algorithm when $N_{ep} = 30$ and $N=3$

<table>
<thead>
<tr>
<th>the simplified TDCCCD model presented in Section 4.2</th>
<th>DQN algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Cost (control packets/second)</td>
<td>Controller placement</td>
</tr>
<tr>
<td>Controller placement</td>
<td>Network Cost (control packets/second)</td>
</tr>
<tr>
<td>12.8</td>
<td>Devices 1, 3, 9</td>
</tr>
</tbody>
</table>
Table 6.4: The selected controller placements in the simplified TDCCCD model presented in Section 4.2 vs. the adapted DQN Algorithm when $N_{ep} = 30$ and $N = 3$

<table>
<thead>
<tr>
<th>Number of devices</th>
<th>The simplified TDCCCD model</th>
<th>DQN algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Devices 1, 3, 9</td>
<td>Devices 1, 4, 9</td>
</tr>
<tr>
<td>20</td>
<td>Devices 3, 6, 10</td>
<td>Devices 1, 3, 10</td>
</tr>
<tr>
<td>30</td>
<td>Devices 1, 8, 15</td>
<td>Devices 1, 22, 30</td>
</tr>
<tr>
<td>40</td>
<td>Devices 19, 20, 30</td>
<td>Devices 1, 5, 36</td>
</tr>
<tr>
<td>50</td>
<td>Devices 1, 21, 31</td>
<td>Devices 1, 34, 40</td>
</tr>
<tr>
<td>60</td>
<td>Devices 1, 17, 19</td>
<td>Devices 1, 2, 10</td>
</tr>
</tbody>
</table>

network cost when solving the CPP for $N=3$ and $N_{ep} = 30$. It is noted that depending on the network topology, different combinations of 3 placements may have the same network cost.

Fig. 6.4 and Fig. 6.5 demonstrate the average number of inter-controller hops and the average number of controller-device hops, respectively when increasing the number of devices in the network obtained from the simplified TDCCCD model and the adapted DQN algorithm. As shown in Fig. 6.4 and Fig. 6.5, the adapted DQN algorithm is able to find the controller placements and assign the controllers to devices such that the average number of hops among devices and their assigned controllers and the average number of hops among controllers in the network are very close to the optimal solutions from the simplified TDCCCD model. As demonstrated in these figures, for a fixed number of controllers ($N=3$), when increasing the number of devices in the network, the average number of controller-device hops increases that also results in higher network cost as shown in Table 6.5. On the other hand, these figures illustrate that finding the controller placements in SDMWN is a trade-off between the average number of inter-controller hops and the average number of controller-device hops in the network. For example, Fig. 6.4 and Fig. 6.5 demonstrate that for a network with 60 devices, the adapted DQN algorithm is able to reduce the average number of inter-controller hops compared to the optimal solution; however, in this case, the average number of controller-device hops and the network cost as shown in Table 6.5 will be higher. Moreover, the results in both figures show that the selected controller placements in both approaches are close to each other that reduce the average number of
Table 6.5: Minimum cost obtained from the simplified TDCCCD model presented in Section 4.2 vs. the results of the adapted DQN Algorithm (control packets/second) when $N_{ep} = 30$ and $N = 3$

<table>
<thead>
<tr>
<th>Number of devices</th>
<th>The simplified TDCCCD model</th>
<th>DQN algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>12.8</td>
<td>14.0</td>
</tr>
<tr>
<td>20</td>
<td>45.6</td>
<td>46.8</td>
</tr>
<tr>
<td>30</td>
<td>84.4</td>
<td>88.0</td>
</tr>
<tr>
<td>40</td>
<td>138.0</td>
<td>148.8</td>
</tr>
<tr>
<td>50</td>
<td>213.8</td>
<td>219.2</td>
</tr>
<tr>
<td>60</td>
<td>260.4</td>
<td>286.6</td>
</tr>
</tbody>
</table>

inter-controller hops in the network.

6.3.1 Running the Proposed DQN Algorithm for SDMWN with Mobile Devices

The aforementioned scenarios in this chapter only consider SDMWN with static network devices. To evaluate the performance of the proposed DQN algorithm in SDMWN with mobile devices, a scenario is considered in the topology as shown in Fig. 6.6 which is assumed not to be high mobility. The values presented in Table 6.1 are used for the parameters in this evaluation. As shown in Fig. 6.6a, at time $t_1$, the topology has 6 static network devices. In this topology, the proposed DQN algorithm is able to find device 1 and device 4 as controller placements when $N = 2$ as demonstrated with red circles in this figure. It is assumed that, after a while, at time $t_2$, device 3 moves in the network and device 3 and device 4 are disconnected as shown in Fig. 6.6b. At this time, Algorithm 6.1 selects device 1 and device 4 as controllers in the network. Finally, at time $t_3$, device 5 moves such that link (5,6) is disconnected and device 5 is in the transmission range of device 1 as shown in Fig. 6.6c. At this time, running the DQN algorithm finds the same controller placements (devices 1 and 4).

Table 6.6 shows the controller placements, minimum cost achieved and execution time
Fig. 6.4: The simplified TDCCCD model vs. the adapted DQN algorithm when $N_{ep} = 30$ and $N = 3$ in terms of the average number of inter-controller hops.

Fig. 6.5: The simplified TDCCCD model vs. the adapted DQN algorithm when $N_{ep} = 30$ and $N = 3$ in terms of the average number of controller-device hops.
of running the DQN for the aforementioned topologies as shown in Fig. 6.6. Moreover, Table 6.7 demonstrates the average number of controller-device hops and the average number of inter-controller hops in these three topologies. In the topologies shown in Fig. 6.6b and Fig. 6.6c, the proposed DQN algorithm is able to find the optimal solutions achieved using the simplified TDCCCD model. In both topologies, the optimal controller placements are device 1 and device 4 and the minimum achieved cost is 6 and 5.6 \((control\ packets/seconds)\), respectively. However, for the topology shown in Fig. 6.6a, the optimal controller placements are devices 4 and 3 and the minimum cost achieved is 6 \((control\ packets/seconds)\) as shown in Chapter 3.

Table 6.6 also presents the execution time for running the proposed DQN algorithm for different topologies when \(N_{ep} = 30\). As demonstrated in this table, the execution time for the first topology is higher compared to the topologies shown in Fig. 6.6b and Fig. 6.6c. In the topologies shown Fig. 6.6b and Fig. 6.6c, Algorithm 6.1 is able to find the goal reward (optimal cost) faster and the iterations (lines (7) to (22)) stop, while the algorithm is not able to find the optimal cost for the topology shown in Fig. 6.6a and iterations continue.
Fig. 6.6: Running the proposed DQN algorithm on SDMWN with mobile devices
Table 6.6: Controller placements, minimum cost achieved \((control \ packets/seconds)\) and execution time \((seconds)\) for the mobility scenario when \(N_{ep} = 30\)

<table>
<thead>
<tr>
<th>Topology</th>
<th>Controller placements</th>
<th>Minimum Cost</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 6.6a</td>
<td>1, 4</td>
<td>6.4</td>
<td>1633.82</td>
</tr>
<tr>
<td>Fig. 6.6b</td>
<td>1, 4</td>
<td>6</td>
<td>51.79</td>
</tr>
<tr>
<td>Fig. 6.6c</td>
<td>1, 4</td>
<td>5.6</td>
<td>39.03</td>
</tr>
</tbody>
</table>

Table 6.7: Average number of controller-device hops and average number of inter-controller hops for the mobility scenario

<table>
<thead>
<tr>
<th>Topology</th>
<th>Avg. number of controller-device hops</th>
<th>Avg. number of inter-controller hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 6.6a</td>
<td>2.0</td>
<td>2.0</td>
</tr>
<tr>
<td>Fig. 6.6b</td>
<td>2.0</td>
<td>3.0</td>
</tr>
<tr>
<td>Fig. 6.6c</td>
<td>2.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

6.4 Summary

Deep RL becomes popular in various fields and a few studies have also investigated using Deep RL for the CPP in wired networks and infrastructure-based wireless networks. However, DRL has not been researched for the CPP in SDMWN. Hence, in this chapter, to investigate the potential of using Deep RL to the CPP in SDMWN, a DQN algorithm is adapted to find near-optimal solutions for the simplified TDCCCD model presented in Chapter 4. In addition, a preliminary investigation for a simple mobility scenario is conducted. For SDMWN, which is infrastructure-less and there is no central controller, all devices have equal probability for random movement with different speeds, it is highly challenging to obtain the optimal solution periodically. Based on the results, the proposed algorithm is able to find optimal and near-optimal solutions in a small-scale network with device mobility. The impact of mobility when increasing the number of devices and having controller’s mobility is still an open research direction in solving the CPP in SDMWN.
Chapter 7

Conclusions and Future Work

7.1 Summary

In this thesis, the objective is to propose different cost-effective algorithms for the controller placement problem (CPP) in software-defined multihop wireless networking (SDMWN) to minimize the network cost in the control plane. The network cost in the proposed algorithms is considered as the total number of control packets required to be exchanged in the control plane per second. These control packets consist of the exchanged control packets between devices and their assigned controllers per second to set up flow rules and exchange configurations in the control plane as well as the exchanged control packets among all controllers per second to synchronize the network’s views and obtain the global view of the network.

The contributions of this thesis are listed as follows.

- Two different optimization models are proposed: topology discovery using controller-device communications (TDCD) and topology discovery using inter-controller and controller-device communications (TDCCCD) models to solve the CPP in SDMWN and to find the placements of $N$ controllers in SDMWN and their assignments to network devices while minimizing the network cost. The only difference between these two models is in achieving the global view of the network in controllers. To solve the CPP in SDMWN, the capacity limit of wireless links and the impact of interference on the capacity of links are considered. The results obtained from the optimization model demonstrate the impact of different factors such as the number of controllers, the capacity of wireless links and the arrival rate of new flows in
devices on the generated control overhead in the network. Moreover, the proposed optimization problems are able to minimize the control overhead compared to a random placement. In addition, the average number of hops among devices and their assigned controllers as well as the average number of hops between controllers in the network are calculated to show the performance of the proposed models.

- Due to the high computational complexity of the proposed models, the TDCCCD model that generates fewer control overhead is selected and simplified to a linear model to show the performance of the proposed optimization model when increasing the number of devices in the network.

- A genetic algorithm (GA) based heuristic approach to find near-optimal solutions for the simplified optimization model. The results demonstrate that the proposed GA algorithm is able to find optimal solutions for small-scale networks and find the near-optimal solutions for larger networks with much less computational overhead.

- Three different heuristic algorithms are proposed to the simplified TDCCCD optimization model: a connectivity-degree based, a distance-degree based and a randomly-selected. These three algorithms find the number of controllers, controller placements and assignments in the network. The results show that the connectivity-degree based and the distance-degree based algorithms are able to find optimal solutions for the small-scale networks. Moreover, the computational times for the proposed algorithms are significantly lower compared to the optimization problem for small-scale and large-scale networks.

- Proposing a deep reinforcement learning (Deep RL) to the CPP in SDMWN: the proposed approach is investigated to demonstrate the potential and the impact of using Deep RL in solving the CPP in SDMWN. Using a deep Q-network algorithm (DQN), the results show that the algorithm is able to find optimal and near-optimal solutions for small-scale networks. Moreover, the results demonstrate the effect of increasing the number of devices on the execution time of the proposed DQN algorithm. In addition, a mobility scenario is considered that shows the proposed DQN algorithm finds the optimal solutions when there is a minimal mobility in the network.

### 7.2 Possible Directions for Future Research

There are different possible directions that could be addressed in the future for solving the CPP in SDMWN. Some of these possible future research opportunities are listed as
follows.

- Formulating the proposed optimization problem in SDMWN to find the number of controllers in the control plane to minimize the network cost, the average number of controller-device hops and the average number of inter-controller hops.

- Considering multiple objectives including the energy consumption and available energy level of wireless devices to find the number of controllers, controller placements and assignments in SDMWN.

- Improving the GA-based algorithm to find the optimal number of controllers in SDMWN.

- Investigating using Deep RL to solve the CPP in SDMWN when increasing the number of devices in networks with mobile devices and its impact on the network performance.

- Investigating the impact of changing controller placements and assignments in SDMWN with mobile devices on the network cost.

- Investigating the effect of mobility in controllers in the network and how to solve the CPP, including the number of controllers, the controller placements and controller assignments dynamically to minimize the network cost.
References


108


