Online Optimization for Edge Computing under Uncertainty in Wireless Networks

Gilsoo Lee

Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering

Walid Saad, Chair
Harpreet S. Dhillon
Yaling Yang
Daniel J. Stilwell
Craig A. Woolsey

March 23, 2020
Blacksburg, Virginia

Keywords: Edge Computing, Online Optimization, Internet of Things

Copyright 2020, Gilsoo Lee
Online Optimization for Edge Computing under Uncertainty in Wireless Networks

Gilsoo Lee

ABSTRACT

Edge computing is an emerging technology that can overcome the limitations of centralized cloud computing by enabling distributed, low-latency computation at a network edge. Particularly, in edge computing, some of the cloud’s functionalities such as storage, processing, and computing are migrated to end-user devices called edge nodes so as to reduce the round-trip delay needed to reach the cloud data center. Despite the major benefits and practical applications of using edge computing, one must address many technical challenges that include edge network formation, computational task allocation, and radio resource allocation, while considering the uncertainties innate in edge nodes, such as incomplete future information on their wireless channel gains and computing capabilities. The goal of this dissertation is to develop foundational science for the deployment, performance analysis, and low-complexity optimization of edge computing under the aforementioned uncertainties. First, the problems of edge network formation and task distribution are jointly investigated while considering a hybrid edge-cloud architecture under uncertainty on the arrivals of computing tasks. In particular, a novel online framework is proposed to form an edge network, distribute the computational tasks, and update a target competitive ratio defined as the ratio between the latency achieved by the proposed online algorithm and the optimal latency. The results show that the proposed framework achieves the target competitive ratio that is affected by the wireless data rate and computing speeds of edge nodes. Next, a new notion of ephemeral edge computing is proposed in which edge computing must occur under a stringent requirement on the total computing time period available for the computing process. To maximize the number of computed tasks in ephemeral edge networks under the uncertainty on future task arrivals, a novel online framework is proposed to enable a source edge node to offload computing tasks from sensors and allocate them to neighboring edge nodes for distributed task computing, within the limited total time period. Then, edge computing is applied for mobile blockchain and online caching systems, respectively. First, a mobile blockchain framework is designed to use edge devices as mobile miners, and the performance is analyzed in terms of the probability of forking event and energy consumption. Second, an online computational caching framework is designed to minimize the edge network latency. The proposed caching framework enables each edge node to store intermediate computation results (IRs) from previous computations and download IRs from neighboring nodes under uncertainty on future computation. Subsequently, online optimization is extended to investigate other edge networking applications. In particular, the problem of online ON/OFF scheduling of self-powered small cell base stations is studied, in the presence of energy harvesting uncertainty with the goal of minimizing the operational costs that consist of energy consumption and transmission delay of a network. Such a framework can enable the self-powered base stations to be functioned as energy-efficient edge nodes. Also, the problem of radio resource allocation is studied when a base station is assisted by self-powered reconfigurable intelligent surfaces (RIS). To this end, a deep reinforcement learning approach is proposed to jointly optimize the transmit power, phase shifting, and RIS reflector’s ON/OFF states under the uncertainties on the downlink wireless channel information and the harvested energy at the RIS. Finally, the online problem of dynamic channel allocation is studied for full-duplex device-to-device (D2D) networks so that D2D users can share their data with a low communication latency when users dynamically arrive on the network. In conclusion, the analytical foundations and frameworks presented in this dissertation will provide key guidelines for effective design of edge computing in wireless networks.
Online Optimization for Edge Computing under Uncertainty in Wireless Networks

Gilsoo Lee

GENERAL AUDIENCE ABSTRACT

Smart cities will rely on an Internet of Things (IoT) system that interconnects cars, drones, sensors, home appliances, and other digital devices. Modern IoT systems are inherently designed to process real-time information such as temperature, humidity, or even car navigational data, at any time and location. A unique challenge in the design of such an IoT is the need to process large volumes of data over a wireless network that consists of heterogeneous IoT devices such as smartphones, vehicles, home access points, robots, and drones. These devices must perform local (on-device or so-called edge) processing of their data without relying on a remote cloud. This vision of a smart city seen as a mobile computing platform gives rise to the emerging concept of edge computing using which smartphones, sensors, vehicles, and drones can exchange and process data locally on their own devices. Edge computing allows overcoming the limitations of centralized cloud computation by enabling distributed, low-latency computation at the network edge.

Despite the promising opportunities of edge computing as an enabler for smart city services such as autonomous vehicles, drones, or smart homes, one must address many challenges related to managing time-varying resources such as energy and storage, in a dynamic way. For instance, managing communication, energy, and computing resources in an IoT requires handling many uncertain factors such as the intermittent availability of wireless connectivity and the fact that the devices do not know a priori what type of tasks they need to process. The goal of this dissertation is to address the fundamental challenges in edge computing under uncertainty in an IoT. In particular, this dissertation introduces novel mathematical algorithms and frameworks that exploit ideas from the fields of online optimization, machine learning, and wireless communication to enable future IoT services such as smart factories, virtual reality, and autonomous systems. In this dissertation, holistic frameworks are developed by designing, analyzing, and optimizing wireless communications systems with an emphasize on emerging IoT applications. To this end, various mathematical frameworks and efficient algorithms are proposed by drawing on tools from wireless communications, online optimization, and machine learning to yield key innovations. The results show that the developed solutions can enable an IoT to operate efficiently in presence of uncertainty stemming from time-varying dynamics such as mobility of vehicles or changes in the wireless networking environment. As such, the outcomes of this research can be used as a building block for the large deployment of smart city technologies that heavily rely on the IoT.
To my wife Pyunghwa, my child Noah, and my parents
Acknowledgments

First and foremost, I am deeply indebted to my PhD advisor, Dr. Walid Saad, for all his guidance and support throughout the course of my PhD study. His help, inspiration, and perseverance enable me to accomplish my PhD work including this dissertation. I would like to thank him for the amount of time and effort he has dedicated to mentoring my PhD work and for giving me the opportunity to benefit from his knowledge and experience. I am also thankful for his priceless advice and for his invaluable supervision that guided me to find the right path for my future career and helped me to explore new research directions that allowed me to complete this dissertation. I would like to thank the members of my PhD advisory committee, Dr. Harpreet Dhillon, Dr. Yaling Yang, Dr. Daniel Stilwell, and Dr. Craig Woolsey, for their guidance and valuable comments which have helped me to substantially improve the quality of this dissertation.

I am grateful for having the opportunity to work with great researchers during my PhD study to publish research articles. I would like to specially thank Dr. Mehdi Bennis for his help, advice, and effort to provide me with his insightful comments that improve the quality of my research papers and this dissertation.

I am also grateful to all my friends and colleagues at the NEWS Lab and Wireless@VT for their kind help. They have been great supporters. Their help inspired me to overcome difficulties I encountered during my research. I am so grateful to be surrounded by them and to be able to work with them.

The final and most sincere acknowledgment is dedicated to my family. I would like to thank my wife, Pyunghwa, and my child, Noah, for their endless love and support. I cannot find the words to express my immense gratitude to my wife. Indeed, without her support, I could have never been able to finish this dissertation. I want to express at the outset, heartiest gratitude to my parents for their unconditional love and support. Without their relentless encouragement, I would never have been able to start my PhD study. Their continuous care, love, and support made this dissertation possible.

---

This research been supported by the U.S. National Science Foundation under Grants CNS-1460333, CNS-1814477, IIS-1633363, and by the U.S. Office of Naval Research (ONR) under Grant N00014-15-1-2709.
Contents

1 Motivation, Background, and Contributions 1

1.1 Benefits and Applications of Edge Computing 1

1.1.1 An Overview on Edge Architecture 1

1.1.2 Edge Computing in the IoT Sensor Networks 1

1.1.3 Edge Computing for Vehicular Networks 1

1.1.4 Edge Computing for Smart Factories 1

1.2 Challenges and Related Studies in Edge Computing 2

1.2.1 Network Formation for Edge Computing 1

1.2.2 Computational Task Allocation 1

1.2.3 Radio Resource Management 1

1.2.4 Intelligent Transportation System Supported by Mobile Edge Devices 1

1.2.5 Smart Factory Supported by UAV-Assisted Edge Networks 1

1.2.6 Caching on Edge Networks 1

1.2.7 Edge Network with Energy Harvesting 1

1.3 Limitations of Existing Works 1

1.4 Online Optimization Frameworks for Edge Computing 1

1.5 Summary of Contributions 2

1.5.1 Distributed Edge Network Formation with Minimal Latency 2

1.5.2 Ephemeral Edge Computing in the IoT 2

1.5.3 Blockchain Systems with Wireless Mobile Miners 2

1.5.4 Computational Caching in Edge Networks 2

1.5.5 ON/OFF Scheduling of Energy Harvesting Base Stations 2

1.5.6 Energy-Efficient Networking with Reconfigurable Intelligent Surfaces 2

1.5.7 Channel Allocation for Device-to-Device Communications 2
1.6 List of Publications ................................................................. 15
1.6.1 Journal Publications .......................................................... 15
1.6.2 Conference Publications ...................................................... 16

2 Distributed Edge Network Formation with Minimal Latency 17
2.1 Background, Related Works, and Contributions ......................... 17
2.2 System Model ....................................................................... 18
2.3 Problem Formulation ............................................................. 21
2.4 Task Distribution and Network Formation Algorithms ................ 23
  2.4.1 Overview of the Proposed Optimization Framework ............... 24
  2.4.2 Edge Network Formation: Online Approach ....................... 26
  2.4.3 Task Distribution: Offline Optimization ............................. 27
  2.4.4 Performance Analysis of the Proposed Online Optimization Framework ......................................................... 28
2.5 Simulation Results and Analysis .............................................. 30
  2.5.1 Performance Evaluation of the Online Optimization Framework ........................................................................... 31
  2.5.2 Performance Evaluation of Algorithm 2.1 for a fixed $\gamma$ .... 34
  2.5.3 Optimal Network Size in an Offline Setting ....................... 35
2.6 Summary .............................................................................. 37
2.7 Appendix ............................................................................. 38
  2.7.1 Proof of Proposition 1 ........................................................ 38

3 Ephemeral Edge Computing in the IoT 39
3.1 Background, Related Works, and Contributions .......................... 39
  3.1.1 Related Works .................................................................. 39
  3.1.2 Ephemeral Edge Computing .............................................. 40
  3.1.3 Contributions .................................................................. 43
3.2 System Model and Problem Formulation ................................... 44
  3.2.1 System Model .................................................................. 44
  3.2.2 Problem Formulation ......................................................... 45
3.3 Proposed Online Task Allocation Framework ............................. 46
  3.3.1 Online Greedy Algorithm ................................................ 47
  3.3.2 Performance Analysis ........................................................ 49
3.4 Simulation Results and Analysis .............................................. 52
3.5 Conclusion ......................................................... 56
3.6 Appendix ......................................................... 57
   3.6.1 Proof of Lemma 1 ........................................... 57
   3.6.2 Proof of Lemma 2 ........................................... 58
   3.6.3 Proof of Lemma 4 ........................................... 59

4 Blockchain Systems with Wireless Mobile Miners ............... 60
   4.1 Background, Related Works, and Contributions .................. 60
   4.2 Mobile Blockchain Architecture ................................. 61
      4.2.1 Mobile Blockchain Applications .......................... 62
      4.2.2 System Model: Computation, Mobility, and Transmission .......... 62
   4.3 Average Energy Consumption Analysis ............................ 64
      4.3.1 No-forking Probability .................................... 64
      4.3.2 Average Energy Consumption ............................... 65
   4.4 Simulation Results ............................................. 65
   4.5 Conclusion ...................................................... 67
   4.6 Appendix ....................................................... 68
      4.6.1 Proof of Theorem 7 ......................................... 68

5 Computational Caching in Edge Networks ......................... 69
   5.1 Background, Related Works, and Contributions .................. 69
   5.2 System Model .................................................. 70
   5.3 Problem Formulation .......................................... 71
   5.4 Online Computational Caching .................................. 73
      5.4.1 Bandwidth allocation ...................................... 74
   5.5 Simulation Results ............................................. 75
   5.6 Summary ....................................................... 77

6 ON/OFF Scheduling of Energy Harvesting Base Stations .......... 78
   6.1 Background, Related Works, and Contributions .................. 78
   6.2 System Model .................................................. 80
      6.2.1 Network Performance ....................................... 81
      6.2.2 Power Consumption ......................................... 81
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2.3</td>
<td>Operational Expenditure of Base Stations</td>
<td>82</td>
</tr>
<tr>
<td>6.3</td>
<td>Problem Formulation</td>
<td>83</td>
</tr>
<tr>
<td>6.3.1</td>
<td>ON/OFF Scheduling as an Online Optimization Problem</td>
<td>84</td>
</tr>
<tr>
<td>6.3.2</td>
<td>Approximated Problem</td>
<td>86</td>
</tr>
<tr>
<td>6.4</td>
<td>On/Off Scheduling as an Online Ski Rental Problem</td>
<td>87</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Deterministic Online Algorithm</td>
<td>87</td>
</tr>
<tr>
<td>6.4.2</td>
<td>Randomized Online Algorithm</td>
<td>88</td>
</tr>
<tr>
<td>6.5</td>
<td>Simulation Results and Analysis</td>
<td>91</td>
</tr>
<tr>
<td>6.6</td>
<td>Summary</td>
<td>97</td>
</tr>
<tr>
<td>6.7</td>
<td>Appendix</td>
<td>97</td>
</tr>
<tr>
<td>6.7.1</td>
<td>Proof of Theorem 8</td>
<td>97</td>
</tr>
<tr>
<td>7</td>
<td>Energy-Efficient Networking with Reconfigurable Intelligent Surfaces</td>
<td>102</td>
</tr>
<tr>
<td>7.1</td>
<td>Background, Related Works, and Contributions</td>
<td>102</td>
</tr>
<tr>
<td>7.2</td>
<td>System Model</td>
<td>104</td>
</tr>
<tr>
<td>7.2.1</td>
<td>Wireless Network Model</td>
<td>104</td>
</tr>
<tr>
<td>7.2.2</td>
<td>Energy Consumption Model</td>
<td>105</td>
</tr>
<tr>
<td>7.3</td>
<td>Machine Learning Framework for Energy-Efficient Networking</td>
<td>106</td>
</tr>
<tr>
<td>7.3.1</td>
<td>Problem Formulation</td>
<td>107</td>
</tr>
<tr>
<td>7.3.2</td>
<td>Deep Reinforcement Learning Approach</td>
<td>107</td>
</tr>
<tr>
<td>7.3.3</td>
<td>Performance Analysis: Case Study</td>
<td>108</td>
</tr>
<tr>
<td>7.4</td>
<td>Simulation Results</td>
<td>109</td>
</tr>
<tr>
<td>7.5</td>
<td>Conclusion</td>
<td>112</td>
</tr>
<tr>
<td>8</td>
<td>Channel Allocation for Device-to-Device Communications</td>
<td>113</td>
</tr>
<tr>
<td>8.1</td>
<td>Background, Related Works, and Contributions</td>
<td>113</td>
</tr>
<tr>
<td>8.2</td>
<td>System Model</td>
<td>114</td>
</tr>
<tr>
<td>8.3</td>
<td>Problem Formulation</td>
<td>116</td>
</tr>
<tr>
<td>8.4</td>
<td>Online Channel Assignment: Weighted Bipartite Matching Approach</td>
<td>117</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Construction of a Weighted Bipartite Graph</td>
<td>117</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Greedy Online Algorithm: Procedure and Analysis</td>
<td>118</td>
</tr>
<tr>
<td>8.5</td>
<td>Simulation Results</td>
<td>119</td>
</tr>
<tr>
<td>8.6</td>
<td>Summary</td>
<td>122</td>
</tr>
</tbody>
</table>
9 Conclusions and Open Problems

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1 Summary</td>
<td>124</td>
</tr>
<tr>
<td>9.1.1 Edge Computing over Wireless Networks: Applications and Challenges</td>
<td>124</td>
</tr>
<tr>
<td>9.1.2 Distributed Edge Network Formation with Minimal Latency</td>
<td>124</td>
</tr>
<tr>
<td>9.1.3 Ephemeral Edge Computing for Internet of Things</td>
<td>125</td>
</tr>
<tr>
<td>9.1.4 Blockchain Systems with Wireless Mobile Miners</td>
<td>125</td>
</tr>
<tr>
<td>9.1.5 Computational Caching in Edge Networks</td>
<td>125</td>
</tr>
<tr>
<td>9.1.6 ON/OFF Scheduling of Energy Harvesting Base Stations</td>
<td>125</td>
</tr>
<tr>
<td>9.1.7 Energy-Efficient Networking with Reconfigurable Intelligent Surfaces</td>
<td>125</td>
</tr>
<tr>
<td>9.1.8 Channel Allocation for Device-to-Device Communications</td>
<td>126</td>
</tr>
<tr>
<td>9.2 Open Problems</td>
<td>126</td>
</tr>
<tr>
<td>9.2.1 Edge Communications in the IoT</td>
<td>126</td>
</tr>
<tr>
<td>9.2.2 Edge Machine Learning over Wireless Networks</td>
<td>126</td>
</tr>
<tr>
<td>9.2.3 Joint Design of Cyber-Physical System</td>
<td>127</td>
</tr>
<tr>
<td>9.2.4 Energy-Efficient Green Edge Networking for Massive IoT</td>
<td>127</td>
</tr>
</tbody>
</table>

Bibliography: 128
List of Figures

1.1 Example of distributed edge computing and its integration with cloud computing. ......................... 2

2.1 System model of the edge networking architecture and the cloud. .............................................. 18
2.2 Online optimization framework for edge network formation and task distribution. ....................... 23
2.3 Flow chart of the proposed framework for edge network formation and task distribution. .............. 25
2.4 Example of the probability $p'$ derived in Proposition 1. ......................................................... 30
2.5 Latency for different task arrival rates at the initial edge node $i$. ............................................ 31
2.6 Latency for different number of neighboring nodes. ................................................................. 32
2.7 Changes in the target competitive ratio $\gamma$ over 700 updates. ............................................... 33
2.8 Edge transmission service rate with respect to the number of neighboring nodes. ....................... 33
2.9 Task distribution with respect to the number of neighboring nodes. ........................................... 34
2.10 Latency comparison versus the target competitive ratio. ........................................................... 35
2.11 The required number of observations for different values of $\gamma$. .......................................... 36
2.12 Performance comparison of two bandwidth allocation schemes with respect to the number of neighboring nodes. ................................................................. 36
2.13 Latency for different number of neighboring edge nodes in an offline setting. ............................ 37

3.1 Ephemeral edge computing framework where the vehicles are edge devices that form a local edge computing network during a limited time period. The vehicles offload their computational tasks from sensors and process in a distributed way while moving on the same direction. ......................... 41
3.2 Ephemeral edge computing framework to offload computational tasks from sensors and allocate the offloaded tasks to neighboring edge-computing UAVs when the source edge-computing UAV is moving to the destination. ................................................................. 42
3.3 An IoT environment where the sensory IoT data is instantly requested and processed by an ephemeral edge computing network to operate real-time applications such as augmented reality and gaming. 42
3.4 Online edge computing framework to offload computational tasks and allocate the offloaded tasks to neighboring edge nodes in total edge computing period $t_{tot}$ within an ephemeral edge computing system. ................................................................. 44
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.5</td>
<td>Example of the cumulative probability distribution of $F_{K</td>
</tr>
<tr>
<td>3.6</td>
<td>The empirical competitive ratio in (3.8) with respect to the different values of $\min_i \beta_{ij}$ when $\alpha = 1$.</td>
</tr>
<tr>
<td>3.7</td>
<td>Average number of computed tasks for the different total times when $\alpha = 1$ and 100.</td>
</tr>
<tr>
<td>3.8</td>
<td>Comparison between the proposed algorithm’s result and the offline optimal solution in terms of percentage of computed tasks for different $t_{tot}$.</td>
</tr>
<tr>
<td>3.9</td>
<td>The empirical competitive ratio $D_{IP,OPT}/D_{IP}$ in (3.8) and the analytical competitive ratio $P_{LP}/D_{IP}$ in (3.13).</td>
</tr>
<tr>
<td>3.10</td>
<td>Percentage of computed tasks for different computational speeds of neighboring edge nodes and different task sizes when bandwidth is varying between 3 and 7 MHz.</td>
</tr>
<tr>
<td>3.11</td>
<td>Percentage of computed tasks for different number of neighboring edge nodes and different maximum communication distances.</td>
</tr>
<tr>
<td>3.12</td>
<td>Percentage of computed tasks for different transmit powers with respect to different computing speeds of neighboring nodes.</td>
</tr>
<tr>
<td>4.1</td>
<td>System model and timing diagram of normal and forking events.</td>
</tr>
<tr>
<td>4.2</td>
<td>Probability of a no forking event in one PoW computation.</td>
</tr>
<tr>
<td>4.3</td>
<td>Average energy consumption of an MM required to compute the PoW of a block.</td>
</tr>
<tr>
<td>4.4</td>
<td>Average energy consumption of an MM for different values of $P_{tx}$ and $\gamma_0$.</td>
</tr>
<tr>
<td>5.1</td>
<td>Example of the computing system of an edge node $i$ that consists of six neighboring edge nodes. In this example, the user request is operation $\alpha_{1,3}$, so IR $r_{1,1}$ or $r_{1,2}$ can be used as input. Since the memory of edge node $i$ is empty at the beginning, $r_{1,2}$ can be downloaded from neighboring edge node $j$. After finishing the computation, the output IR $r_{1,3}$ is stored in the memory.</td>
</tr>
<tr>
<td>5.2</td>
<td>Total latency for different memory sizes of edge node $i$.</td>
</tr>
<tr>
<td>5.3</td>
<td>Total latency for different bandwidths $B_{\text{max}}$.</td>
</tr>
<tr>
<td>5.4</td>
<td>The CDF of the empirical competitive ratio for the proposed online algorithm.</td>
</tr>
<tr>
<td>6.1</td>
<td>System model of a heterogeneous deployment with self-powered SBSs.</td>
</tr>
<tr>
<td>6.2</td>
<td>Flowchart of Algorithm 1.</td>
</tr>
<tr>
<td>6.3</td>
<td>Snapshot example of network resulting from the proposed ROA approach.</td>
</tr>
<tr>
<td>6.4</td>
<td>Total energy consumption of SBSs and delay cost per SBS for the ROA, DOA, and baselines.</td>
</tr>
<tr>
<td>6.5</td>
<td>Comparison of the total network cost when using ROA, DOA, and a baseline.</td>
</tr>
<tr>
<td>6.6</td>
<td>Comparison of the total network cost with respect to the number of UEs.</td>
</tr>
<tr>
<td>6.7</td>
<td>Empirical competitive ratio between the total cost of the ROA and the optimal cost.</td>
</tr>
<tr>
<td>6.8</td>
<td>Average ON time per SBS with respect to the transmit power of an SBS and the MBS during period $T$.</td>
</tr>
<tr>
<td>6.9</td>
<td>The number of ON/OFF switchings of the network during one period $T$.</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Summary of notations ................................................................. 19
2.2 Simulation parameters ............................................................. 31
5.1 Summary of key notations .......................................................... 71
5.2 Simulation parameters ............................................................. 75
5.3 Computational latency in the case of Fig. 5.2. .............................. 75
6.1 Simulation parameters ............................................................. 92
Chapter 1

Motivation, Background, and Contributions

The Internet of Things (IoT) is expected to connect over 50 billion things worldwide, by 2020 [1, 2]. This massive number of IoT devices is expected to generate more than two Exabytes of data per day [3]. To process the data with low computing and communication latency, the local proximity of IoT devices can be exploited for offloading computational tasks, in a distributed manner. Such local computational offload gives rise to the emerging paradigm of edge computing. Edge computing allows overcoming the limitations of centralized cloud computation by enabling distributed, low-latency computation at the network edge, for supporting various wireless and IoT applications. To meet the low latency communication and computing requirements of such applications, relying on cloud computing will no longer be possible due to the round-trip delay needed to reach the cloud data center [3]. Therefore, at the beginning, the cloud computing architecture is modified by adding a new middle layer between the cloud layer and network edge. The new middle layer is called as a fog layer that consists of various networking equipment such as switches, routers, local servers, and access points that can be used to process the data offloaded from network edge. However, the role of the networking equipment in the fog layer can be fulfilled by using modern end-user devices in an IoT environment due to the advancement of device performance in both networking and computation. This results in moving the fog layer closer to the network edge. Consequently, edge computing is implemented to handle and process the IoT data by using the end-user devices located at the network edge. The advantages of the edge computing architecture comes from its ability to transfer some of the network functions to the network edge. In particular, some of the cloud’s functionalities such as caching, control, and computing are migrated to end-user devices called edge nodes. Indeed, significant amounts of data can be stored, controlled, and computed over edge networks that can be configured and managed by end-user nodes [4]. Within the edge computing paradigm, computational tasks can be intelligently allocated between the edge nodes and the cloud to meet computational and latency requirements [5]. Also, leveraging the physical proximity of edge nodes and pooling their resources allows for low-latency computation. Therefore, by pooling the computing resources of edge nodes located in proximity of one another at the edge of a wireless network, edge computing can achieve low-latency data transmission and computation.

1.1 Benefits and Applications of Edge Computing

In this section, we present the main advantages and potential applications of edge computing in wireless networks.
1.1.1 An Overview on Edge Architecture

To enable low-latency edge computing, a so-called fog-centric radio access network (FRAN) architecture such as the one introduced in [6] is typically used. Fig. 1.1 includes a FRAN architecture consisting of edge-enabled access points and a central cloud that operates a cloud-based radio access network (CRAN). In CRAN, the centralized baseband units are pooled in the cloud, and the remote radio heads are connected to the centralized baseband units through the fronthaul. To achieve low latency in a CRAN architecture, the interconnection between the radio remote heads and the baseband needs to have enough bandwidth. However, when the bandwidth of the fronthaul is limited, the latency at the fronthaul cannot be neglected. To meet the low-latency requirement of task computation for the IoT devices, relying on conventional, remote cloud solutions may not be suitable due to the high end-to-end transmission latency of the cloud [4].

To overcome the fronthaul limitations of the CRAN architecture, the FRAN architecture uses the intelligence at the network edge. It is possible to use an edge device to compute the tasks since the edge node’s hardware has been upgraded. By using the edge node’s hardware, instead of the servers at the cloud, data generated at the edge of the network does not need to be sent to the cloud through the fronthaul. Also, since the data is processed at the edge, the delay to move the data can be further reduced. Moreover, since the FRAN architecture is an extension of the CRAN architecture, the cloud is resources will remain accessible when needed. Therefore, based on edge computing, the computational tasks can be processed using a collaboration between the edge nodes and the cloud servers. In the hybrid architecture including edge and cloud layers, edge computing can be configured by the edge nodes to meet the different service requirement of the applications, data, and computing tasks. Consequently, FRAN can be seen as more versatile and flexible architecture accommodating the various types of services with different service quality
By extending typical FRAN to various real-world scenarios, Fig. 1.1 shows an illustration of a comprehensive edge architecture consisting of different types of edge nodes including user devices, vehicles, home appliances, and edge access points operating with a central cloud. While FRAN adopts edge computing for a low latency in a radio access network, the concept of edge computing is generally applied to various real-world applications in that the edge architecture can be implemented without a significant change to existing networks. For instance, when the edge nodes can form an edge network locally in Fig. 1.1, the edge nodes need to discover the neighboring edge nodes [7]. Discovering a neighboring edge node is possible by using existing discovery mechanisms from device-to-device (D2D) networks such as those in [8–15]. To use the discovery mechanisms of D2D networks, the edge nodes need to be located in proximity. A neighboring node can broadcast the beacon signal to inform other nodes on its presence and availability in joining edge computing. Upon receiving the beacon signal, the initial edge node is able to discover the presence of the neighboring node. For example, the neighboring discovery technique can be implemented by adopting the D2D discovery algorithms known as ProSe from the 3GPP LTE standard [9]. Alternatively, edge network can adopt procedures for edge node discovery and message exchange [16] that are analogous to those that have been previously implemented in wireless sensor networks [10], 802.11 [11], Bluetooth [12], or Zigbee [13].

In the edge architecture, the edge nodes can consist of various device types such as laptops, smartphones, or even home appliances, and, therefore, the number of edge nodes participating in edge computing can be readily scalable. As the number of edge nodes participating into edge computing increases, more computational resource can be pooled. Therefore, if the computational tasks running on the application suddenly require high computing resources, more edge nodes can be introduced into an existing edge network in order to accumulate and procure high computing resources. Also, sometimes the sensory raw data can be sensitive information. In edge computing, the raw data can be processed at the vicinity of its source. By storing and processing the data within a physical trust boundary, a strong privacy policy can be enforced. This can be particularly beneficial for users and companies that need high security and protection of their raw data.

As a result, edge computing can be implemented without a significant modification in the operation of existing wireless networks. Moreover, the deployment of edge computing has many benefits compared to only using cloud computing. Next, we will discuss how edge computing can be used for IoT sensor networks, vehicular networks, and smart factories, respectively.

### 1.1.2 Edge Computing in the IoT Sensor Networks

In an IoT environment, many types of sensors can be used in various environment. For instance, in a smart city, the building maintenance systems can have many sensors to monitor the status of the building, e.g., temperature, motions, water, ventilation, air-conditioning/heating, power generation, etc. Also, surveillance systems including security cameras and sensors generate the data. In smart grid systems, the sensors are used to monitor the power grid systems including generation, distribution, and transmission. The large volume of sensory data may not be amenable to direct, local computation at the sensors due to the insufficient computing resources of small sensors. Also, wireless sensor networks are generally designed to operate with low power consumption. Therefore, IoT devices in the sensor networks may not be suitable to run computation-intensive applications directly, and, hence, may solicit computational assistance from more capable, neighboring edge nodes.

To overcome these challenges, edge computing can be used in the IoT sensor networks. To implement the edge paradigm, a three-layer network architecture is typically needed to manage sensor, edge (fog), and cloud layers [6]. As shown in Fig. 1.1, the sensory data can be offloaded to the edge nodes. Once the data is transmitted from the sensor layer to the edge layer, edge nodes can be clustered for pooling their resources, and then, distributed edge computing is used to process the computational tasks of an edge node with low latency. For instance, if the sensory data is used to run a big data application requiring a large amount of computing resources, the data is first offloaded to an edge node located close to the sensor. Then, the edge node can initiate and manage edge computing. If the edge node does not
have sufficient computing resources, it can find other neighboring edge nodes that have idle computing resources that they are willing to share. After discovering neighboring nodes, the computing tasks can be distributed to the neighbors and computed in a distributed way.

Also, edge computing can be applied to image processing IoT applications by enabling the video cameras in the sensor layer to offload the raw video data to neighboring edge nodes. In particular, transmitting the raw video data from the video cameras to the cloud requires a significant bandwidth. For instance, when 12,000 users transmitting 1080p video requires 100 Gbps in a link [17]. In order to reduce the bandwidth usage between the video cameras and cloud layers, the framework proposed in [18] transmits the raw data to an edge node that performs computer vision analytics to extract the features of the video such as location, time, recognized face, and image content tags. After finishing the initial analytics of the videos, the edge node can transmit meta-data to the cloud. Therefore, the transmitted data size can be significantly reduced. Also, to further reduce the latency in the data analytics, the local proximity of neighboring edge nodes can be exploited for offloading computational tasks, in a distributed manner. For instance, the so-called “fog data service” provided by Cisco is a data analytics tool that enables network decisions on which data needs to be processed at a local edge node and which data is sent to the cloud [19]. In this fog data service, a large volume of data can be collected to the edge nodes and processed in a proximity of the location where the data is collected. Then, the processed and virtualized data is sent to the cloud for additional computation-intensive applications such as big-data analytics.

1.1.3 Edge Computing for Vehicular Networks

In the next decade, almost every type vehicles such as car, plane, rail, and drone must have a connectivity to wireless networks [20]. Also, as shown in the recent development of self-driving cars from Tesla or Uber, emerging vehicles will have on-board computing systems that have enough computing resources to run driving assistance and infotainment applications. By using wireless connectivity, the on-board systems of a vehicle will be able to communicate with sensors, neighboring vehicles, road infrastructure, and the cellular systems [21]. Under such a vehicle-to-everything (V2X) communication environment, various applications that rely on computing and caching capabilities can be used to enhance road safety and provide a better user experience.

V2X communications is a key component to manage the traffic flow of future autonomous connected vehicles in intelligent transportation systems (ITSs) [22]. Remarkably, in recent years, V2X communications technologies such as 5G New Radio (NR)-based cellular vehicle-to-everything (C-V2X) and 802.11p-based dedicated short range communications (DSRC) have been studied to enable communications among vehicles and everything around, such as infrastructure and pedestrians [23, 24]. For instance, the V2X communications can be used to share the current acceleration and location of vehicles though vehicle-to-vehicle (V2V) communications, to broadcast the real-time information on traffic signals through vehicle-to-infrastructure (V2I) communications, and to send warnings to pedestrians through vehicle-to-pedestrians (V2P) communications [25]. Therefore, if the vehicles are able to use V2X communications, active safety can be improved by providing non-line-of-sight awareness and a higher level of predictability. The road traffic flow can also be maximized while reducing the traffic delay and fuel consumption by obtaining information about switching times of traffic lights in advance. Thus, V2X communications is essential to realize the ITS in which the vehicle’s dynamics are intelligently controlled by exchanging the environment information with other vehicles and processing real-time information from sensors, camera, and ladar. To use real-time traffic information, it is essential to exchange the traffic data between vehicles via V2X communications and process the data via edge computing. Therefore, wireless resource management and optimization should be studied in the context of traffic management including vehicle trajectory optimization.

Also, to control a vehicle autonomously, the end-to-end latency of the message exchange needs to be less than 10 msec [26]. However, the average round-trip time to access Amazon Elastic Compute Cloud (EC2) can be 74 ms [27]. When V2X applications need to perform computation for data mining and analytics, offloading the computational tasks to the cloud cannot meet the stringent application latency requirement. Therefore, edge computing can be adopted to
achieve a low latency in running those applications. Also, edge computing can be applied to run the safety systems such as collision avoidance applications. The applications for collision prevention need computations of image processing to assist the driver when driver’s visibility is blocked by obstacles [28].

Moreover, edge computing can be used in smart traffic light systems [27]. Smart traffic light systems require latency between 10 ms and 100 ms [26]. To satisfy this latency requirement, the local data collected by the vehicles, roadside units, and sensors can be processed by a local edge network. To this end, V2V and vehicle-to-road infrastructure communications can be used to form a local edge network [29]. When edge computing is applied to the smart light systems, vehicles can send the locally processed data to the cloud via vehicle-to-Internet connection. The accumulated traffic information on the cloud can be used to predict the future traffic and alleviate the future road traffic congestion.

Furthermore, edge computing enables connected vehicles to cache data to reduce the traffic load of the wireless links between the vehicles and the cloud. When serving connected vehicles, the limited bandwidth of the wireless links between vehicles and the cloud is shared between safety and entertainment systems. In such a case, safety applications should have a priority in transmission. For instance, in an emergency, the real-time video and sensory data has to be transmitted from vehicles to the cloud through the cellular network. Then, the safety application running on the cloud is able to make a real-time decision to control the vehicles. To minimize the bandwidth use of entertainment systems, edge computing can be used to cache the data, e.g., movies, in a local edge network. By doing so, the multimedia data for entertainment systems can be processed and streamed to other neighboring vehicles through V2V communications.

### 1.1.4 Edge Computing for Smart Factories

Many areas of the industry including manufacturing, construction, and mining are deploying smart factories leveraging IoT sensor networks, wireless systems, and intelligent controllers to automate equipment and enhance productivity. Also, this broad technological adoption in the different forms of industrial areas will create various use cases exploiting autonomous vehicles and drones for factory automation [20]. In emerging smart factory scenarios (also known as Industry 4.0) [30], sensors will be used to detect and track the manufacturing process, but also sensors will be used to detect malfunctions and send diagnostics signals to actuators in the factory. The actuators in smart factories can control the physical system by carrying out the physical actions such as moving, opening/closing, starting/stopping, and deploying robots, sensors, machines, etc. Therefore, the actuators should be considered in the domain of wireless sensor networks [27].

When the actuators are connected to sensor networks, the control systems in smart factories need to have computation and storage functionalities. In this case, the sensors and actuators can be controlled by the applications running on the cloud servers located at remote locations. However, implementing the controllers at remote cloud servers can result in a long end-to-end latency in communications. Therefore, to support time-critical and mission-critical applications, edge computing can be applied by placing the controller in proximity of sensors and actuators [8]. Also, unlike traditional sensor networks where the information is moved from the sensors to the sink, smart factories need to move the information in both directions between the sensors and the actuators. The sensory data needs to be transmitted to the actuators to update the current status. Then, the decision is made by the controllers, and the feedback from the actuators has to be delivered to the sensors. Therefore, smart factory applications can have a closed-loop control system [31]. Edge computing moves the location of the controller near the sensors and actuators. Therefore, edge computing enables a smart factory to achieve a low latency in the closed-loop control between sensors and actuators.

Factory systems need to be optimized to manage the process of sensory data transmission, low-latency computation, and proactive decision making in order to quickly react to new situations [32]. Edge computing can use machine-to-machine communications to support opportunistic applications where the data is computed and stored locally [33]. Also, edge computing enables to achieve low latency and locality that are unachievable when the controlling applications are deployed and run at the cloud servers [8]. Therefore, to maximize the potential capabilities of edge computing and to enable the smart factory vision, some of the key challenges include effective in-network computing
and improvement of wireless connectivity to integrate physical and digital systems, i.e., networking and computation.

In summary, we have presented the main advantages and potential applications of edge computing in wireless networks. In particular, we have discussed the effective use of edge computing as an extension of cloud computing to deploy low-latency applications in next-generation wireless networks. Furthermore, in this section, the promising application of edge computing in IoT sensor networks and vehicular networks and their integration with smart factory systems were discussed. Despite such benefits and applications of edge computing in wireless networks, there are several technical challenges that must be also taken into account. Next, we will discuss various challenges related to edge computing in detail.

1.2 Challenges and Related Studies in Edge Computing

The effective deployment of edge computing requires meeting several key challenges. The first challenge pertains to the dynamic formation of an edge network for computational task processing. In other words, the system must be able to form suitable edge networks to service various tasks. In edge computing, any end-user device can be used as an edge node, and, also, the user of the edge device may decide whether to join in a local edge network for participating into distributed edge computing. Therefore, when the types of edge devices can be different, the future availability of each edge node may not be known in advance. Thus, it can be challenging to optimize the edge network formation if any prior information on the edge nodes is not given in advance.

Distributed task allocation is another challenging task as it depends on many parameters such as the number of edge nodes, wireless channel conditions between edge nodes, and the computing capabilities of edge nodes. In addition, making a decision to offload the proper number of tasks to each edge node becomes challenging since the wireless data rate can change depending on how wireless resources are being used to offload the task data. In fact, optimizing task allocation with wireless resource allocation is significantly more challenging than optimizing wireless resources only. The total latency in communications and computation is affected by both the allocated wireless resource and the tasks. This is due to the fact that wireless allocation changes the data rates of the edge nodes, and the date rates affect the optimal allocation of tasks for edge nodes. Moreover, while offloading computational tasks, if the energy consumption of the sensors needs to be taken into account, the task allocation problems become more challenging.

Enabling V2X communications requires a major paradigm shift in existing research endeavors. First of all, without a framework considering both communications and computing for vehicle and traffic controlling, the promises of automated connected vehicle technology will not be fully realized. In particular, it is challenging to process the data by using a vehicle’s on-board computer since the size of the video and sensory data is large. Therefore, radio resource management and optimization should be thoroughly investigated while considering the physical characteristics of vehicles and traffic flow. Also, to use real-time traffic information, it is essential to exchange the traffic data between vehicles via V2X communications and process the data via edge computing. In an urban environment, each vehicle’s trajectory can be jointly optimized with the design of edge computing system so that the communication and computing latency is minimized.

In certain application scenarios, such as smart factories, edge computing may be implemented over mobile nodes, such as unmanned aerial vehicles (UAVs). UAV-enabled edge computing can be quite challenging due to various reasons. For instance, to exploit edge UAVs in smart factories, the problems optimizing trajectory, task allocation, and network formation can be jointly considered. In fact, optimizing the flying path is challenging due to the physical constraints related to the hardware of UAVs. For example, the optimal trajectory of UAVs can be dependent on the mobility, energy consumption, and the wireless channel conditions. Also, regulations and policies are applied to UAVs, and, therefore, the optimal trajectory can be affected. Thus, optimizing the trajectory can be a multidimensional problem considering many key factors of UAV networks. Moreover, to optimize the trajectory of UAVs used for edge computing, the energy consumption of edge UAVs will be the sum of the energy consumption for computation, communication, and mobility. The task allocation for edge UAVs needs to consider the total energy consumption of edge UAVs since the
flying times of the UAVs are affected by the energy consumption. Furthermore, minimizing the latency of the edge computing on UAV networks can be challenging as it involves many variables and parameters related to UAVs, i.e., mobility, computation, and communications.

Caching data at an edge node can be a challenging problem in an edge computing environment. Since edge nodes have a limited storage size, the optimal set of data needs to be selected so that the storage is efficiently used to cache the data that will be used frequently in the near future. Edge computing can be used to support many applications. As the number of applications in edge computing increases, the efficient use of the data storage of edge nodes becomes an important problem. Also, considering that any devices can be temporarily used as edge nodes, the cached data can be lost when the edge nodes leave the edge network. Therefore, it is challenging to optimize in what edge node the data should be stored.

Other challenges for edge computing include exploiting energy harvesting at an edge node. As the number of edge nodes increases, the total energy consumption of the network will also increase. In fact, each edge node will have to spend part of its energy to compute other nodes’ tasks. Therefore, it is imperative to reduce the energy consumption by deploying self-powered edge nodes. When edge nodes use energy harvesting to power their circuits, due to the intermittent arrival of the harvested energy at the edge nodes, the stored energy of edge nodes can be highly uncertain. Thus, self-power edge nodes can be turned off at a certain moment due to the shortage of the stored energy in the devices. Hence, it is challenging to efficiently manage the stored energy of the self-powered edge nodes by properly scheduling turning their on/off status to maintain an edge network.

Having summarized the key applications of edge computing, next, we present the existing works that have addressed some of the challenges pertaining to network formation, task allocation, exploiting V2X and UAV networks, caching, and energy harvesting for edge computing.

### 1.2.1 Network Formation for Edge Computing

It is challenging for edge nodes to dynamically form and maintain an edge network that they can use for offloading their task. This challenge is exacerbated by the fact that edge computing devices are inherently mobile and will join/leave a network sporadically [34]. Moreover, to efficiently use the computing resource pool of the edge network, novel resource management schemes for the hybrid edge-cloud network architecture are needed [27]. To reap the benefits of edge networks, many architectural and operational challenges must be addressed [8, 35–48]. A number of approaches for edge network formation are investigated in [8, 35–38]. To configure an edge network, the authors in [8] propose the use of a D2D-based network that can efficiently support networking between an edge node and a group of sensors. Also, to enable connectivity for edge computing, the work in [35] reviews D2D techniques that can be used for reliable wireless communications among highly mobile nodes. The work in [36] proposes a framework for vehicular edge computing in which edge servers can form a distributed vehicular network for content distribution. In [37], the authors study a message exchange procedure to form a local network for resource sharing between the neighboring edge nodes. The work in [38] introduces a method to form a hybrid edge architecture in the context of transportation and drone-based networks.

### 1.2.2 Computational Task Allocation

Once an edge network is formed, the next step is to share resources and tasks among edge nodes as studied in [39–47]. For instance, the work in [39] investigates the problem of scheduling tasks over heterogeneous cloud servers in different scenarios in which multiple users can offload their tasks to the cloud and edge layers. The work in [40] studies the joint optimization of radio and computing resources using a game-theoretic approach in which mobile cloud service providers can decide to cooperate in resource pooling. Meanwhile, in [41], the authors propose a task allocation approach that minimizes the overall task completion time by using a multidimensional auction and finding
the best time interval between multiple auctions to reduce unnecessary time overheads. The authors in [42] study a latency minimization problem to allocate the computational resources of the mobile-edge servers. Moreover, the authors in [43] study the delay minimization problem in edge and cloud-assisted networks under heterogeneous delay considerations. Moreover, the work in [44] investigates the problem of minimizing the aggregate cloud fronthaul and wireless transmission latency. In [45], a task scheduling algorithm is proposed to jointly optimize the radio and computing resources to reduce the users’ energy consumption while satisfying delay constraints. The problem of optimizing power consumption is also considered in [46] subject to delay constraint using a queueing-theoretic delay model at the cloud. Moreover, the work in [47] studies the power consumption minimization problem in an online scenario subject to uncertain task arrivals. Furthermore, the work in [49], studies how tasks can be predicted and proactively scheduled. Last, but not least, the work in [50] implements a prototype for edge computing that can manage edge node’s resources in a distributed computing environment.

The IoT environment can include many small sensors for smart home or smart building control system that can deliver a wide range of services to the end-users. Considering a network that consists of a sensor layer and an edge layer, the IoT sensors with low computing power should offload their computational tasks to other edge nodes in the edge layer such as smartphones or APs. By doing so, the tasks from the sensors can be computed with more powerful computing resources provided by more capable edge nodes. The tasks are first offloaded from sensors to a certain edge node. Then, the edge node received the tasks initiates distributed edge computing, and this node is denoted by an initial edge node. To compute the tasks over the edge network, the initial edge node distributes the received tasks to other neighboring edge nodes. In the task distribution, the transmission latency can be defined as the sum of the waiting time before the tasks are transmitted and the wireless transmission delay from the initial edge node to another edge node. Once a neighbor successfully receives the tasks from the initial edge node, the computation procedure yields the computational latency that includes the waiting time in the computational queue and the actual processing time.

In this typical use case of edge computing, the initial edge node first needs to select an optimal set of neighboring edge nodes and distribute the tasks to this selected set so as to minimize the latency. However, in practice, neighboring edge nodes can dynamically join and leave the edge computing network. Therefore, a given edge node will typically not be able to know when neighboring edge nodes will join the network and also where those neighboring nodes are located. Thus, when the arrival of neighboring edge nodes is uncertain, the information cannot be known to the initial edge node. This uncertainty of the location and the arriving order of neighboring edge nodes can be modeled as a sequential input of an online optimization problem. Then, an online optimization problem can be formulated to minimize the total computational latency defined as the sum of the transmission latency and computational latency.

### 1.2.3 Radio Resource Management

Radio resource allocation schemes focusing on edge computing have been investigated in prior works [51–56]. In particular, the work in [51] investigates the use cases, system architecture, and resource management protocols of edge computing for a number of potential services ranging from the real-time video applications running on a low-mobility user device to the connected vehicle safety applications operating on high-mobility cars. Also, recent prior works [52–56] study the deployment and resource allocation of edge computing in general scenarios where the edge computing devices are remaining in a static or low-mobility network. In particular, the work in [52] investigates an edge computing platform deployed in network infrastructure such as base stations to provide contents to the users while maintaining a required quality of service. Also, the computational task offloading and radio resource allocation problem are jointly studied in a wireless powered edge computing system by using deep reinforcement learning [53].

A caching scheme is designed to maximize fairness in edge computing environment where the heterogeneous types of devices have the different communication and computing resources [54]. The authors in [55] propose Lyapunov optimization-based computation offloading algorithm to jointly optimizing the transmit power and CPU-clock speeds when edge computing devices are powered by energy harvesting techniques. Moreover, in [56], a computational offloading strategy is proposed to maximize the utilization of each edge node’s computing resources when a part of deep learning network is computed by edge nodes.
1.2.4 Intelligent Transportation System Supported by Mobile Edge Devices

With the emergence of V2X communications, the automated connected vehicle will be able to communicate with other connected vehicles while exchanging the real-time information to the road infrastructure such as road side units. Enabling V2X communications requires a major paradigm shift in existing research endeavors. Therefore, to use real-time traffic information, it is essential to exchange the traffic data between vehicles via V2X communications and process the data via edge computing. In particular, edge computing has been investigated in various scenarios incorporating connected vehicles [57–62]. The authors in [57] investigate the use case of vehicular edge computing and system architecture by providing a deployment scenario of V2X communications. The work in [59] proposes a low-complexity computation offloading algorithm that minimizes the computation cost at vehicles. The authors in [58] study a vehicular edge computing to develop a distributed reputation management system where the computing resources allocation is optimized to improve security protection. Also, edge computing is assumed to process a computation required to maintain a blockchain by using local computing resources of vehicular nodes [60, 63]. The authors in [61] study smart contract deployed on an edge computing system to enables the vehicles to store and share the data securely. In [62], the software-defined network concept is applied to propose an edge computing architecture in which the control plane protocol is designed to cluster a set of neighboring vehicles and a centralized edge computing server is used to optimize the data transmission path.

1.2.5 Smart Factory Supported by UAV-Assisted Edge Networks

Computing sensory data in a timely manner is essential to operate a physical factory system. To this end, the concept of edge computing can be applied in cyber-physical smart factory systems where UAVs are deployed and perform key functions such as data storage, computing, control, and transmission [64, 65]. Recently, the use of UAVs for wireless and computing scenarios has been studied in [66–72]. In particular, using UAVs as a relay is studied in [66–69]. The authors in [66] propose a framework that jointly optimizes UAV placement, user association, and uplink power control so that UAVs can collect data from ground sensors. In [67], a UAV relaying scheme is studied to support a wireless cellular network by temporarily offloading traffic of overloaded cells into neighboring cells. In [68], UAVs are used as message ferries that collect information in wireless sensor networks and carry the data to the destination. The authors in [69] propose a framework using UAVs for data collection from ground sensor networks where relay nodes act as cluster heads for adjacent sensors, and the data collected by relays is transmitted to the UAVs. Also, airborne edge computing using a UAV is studied in [70–72]. In [70], the authors investigate a UAV-mounted cloudlet in which UAVs equipped with a computing processor offload and compute the tasks offloaded from ground devices. The authors in [71] study UAV-enabled wireless powered mobile edge computing system. The authors in [72] propose a relaying system that uses a UAV to store the processed data in a buffer and optimize the receiving and transmitting data size to minimize the energy consumption.

In addition, as a use case of edge computing to control UAV network, the authors in [31] propose three layers to control functionalities of UAVs. For instance, when the autonomous drones are deployed for package delivery, global path planning is optimized at the cloud. Then, the edge nodes such as edge access points and edge UAVs control the drone to follow the path and track the trajectory of the drones. Finally, the drones control their velocity themselves by operating the PID controllers. Therefore, the architecture of edge computing can be applied to deploy and control the drones.

1.2.6 Caching on Edge Networks

Since the data can be stored at the edge nodes, exploiting the caching capabilities of edge nodes is deemed as an essential step to improve system throughput and reduce the latency [44, 73–77]. For instance, a case study that uses real measurement data in [73] shows that an edge caching architecture in vehicular networks can reduce the distance
traveled by the data in the network. To show the impact of both caching and fronthaul on latency, the authors in [74] provide an information-theoretic latency analysis in edge radio access network with edge caching and cloud processing. Meanwhile, the work in [75] introduces a coding technique to reduce the latency of computation and bandwidth consumption, when data is redundantly stored in the network. The authors in [76] study the problem of caching for optimizing the users’ quality of experience using unmanned aerial vehicles. In [77], the authors study a distributed cluster formation of edge nodes with caching capability while maximizing a system throughput. Moreover, the authors in [44] investigate the problem of maximizing the minimum delivery rate of requested content in a cache-enabled edge network while considering the fronthaul capacity and power constraint. Furthermore, if the popularity of data changes within a short time interval, the data requests can be modeled as an online input. When edge nodes is unable to know full knowledge on user requests, online optimization can be used to develop the caching strategy. For instance, the work in [78] proposes an online scheme to minimize the data delivery time by replenishing the edge node’s cache and scheduling the delivery of the requested files.

1.2.7 Edge Network with Energy Harvesting

To reduce energy consumption of edge computing, mobile network operators can deploy self-power edge nodes, e.g., edge APs, that use the harvested energy from ambient energy sources, e.g., solar or wind. However, when self-powered edge nodes are not connected to the conventional power grid, they can be turned off due to the unexpected energy outage. It is challenging to efficiently manage the stored energy of the self-powered edge nodes by properly scheduling turning on/off status to maintain an edge network [79, 80]. In particular, the authors in [81] study the optimal sleep policy based on dynamic programming with the statistical energy arrival information. Also, deployment of self-powered edge APs is challenging since the edge network may no longer be able to use the data cached at APs that are no longer operational due to power outage. This uncertainty on energy harvesting makes it very challenging to operate self-powered edge computing networks. Partially address this challenge, the problem of maximizing the caching payoff of self-powered APs is studied in [82]. In this work, the proposed algorithm enables self-powered edge APs to decide whether to accept the arriving requests from users and also whether to cache the downloaded data.

1.3 Limitations of Existing Works

As discussed in Sections 1.1 and 1.2, in order to reap the benefits of edge computing, many technical challenges such as computational task allocation, edge network formation, caching, and energy consumption need to be taken into account. The previous studies presented in Section 1.2 have addressed some of these challenges such as optimizing the task allocation and network formation. However, this prior art lacks comprehensive studies on modeling, performance analysis, and optimization of edge computing in dynamic changes of environment, notably when the future information on edge nodes such as their wireless channel gains, computing capabilities, requested computational tasks, and availabilities of joining edge computing is not known. In particular, when the future information on environment is sequentially revealed, several edge computing-related problems such as task allocation, network formation, caching data, and channel assignment are only marginally studied in the prior art. Moreover, the optimization problems introduced in the existing prior art has been solved by using offline algorithms that require the complete or statistical information on the environment although such information is sometimes not readily known in advance. In summary, the main limitations of the previous studies on edge computing are as follows:

- In the existing edge network formation and task scheduling works in edge computing [36–46], it is generally assumed that information on the formation of the edge network is completely known to all nodes. However, in practice, the edge network can be spontaneously initiated by an edge node when other neighboring edge nodes start to dynamically join or leave the network. Hence, the presence of a neighboring edge node to which one can offload tasks is unpredictable. Indeed, it is challenging for an edge node to know when and where another
edge node will arrive. Thus, there exists an inherent uncertainty stemming from the unknown locations and availability of edge nodes.

- The majority of the existing literature [37, 39–43, 45, 46, 83, 84] relied on centralized optimization algorithms or game-theoretic solutions for addressing edge challenges. However, all of those solutions require substantial overhead and information exchange to operate. In contrast, there is a need for fully online solutions that can be deployed to optimize the performance of edge computing, with little to no information exchange or message passing among the edge nodes.

- Most of the existing works [36, 37, 41–45] typically assume a simple transmission or computational latency model for an edge node. In contrast, the use of more practical delay models (e.g., via queuing theory) for both transmission and computational latency is necessary to capture realistic latency metrics.

- The prior art on edge computing employing both communications and computing [52–55, 58–62, 66, 68, 70, 71, 85–88], generally assumes that information on prospective computing tasks such as data size and arriving order is completely known. However, in practice, the information on tasks can be revealed gradually over time since sensory data is randomly generated. Hence, when a series of tasks are offloaded to a neighboring edge node, predicting prospective future tasks is often not possible.

- In existing works on edge computing [52–55, 58–62, 66, 68, 70, 71, 85–88], it is generally assumed that edge computing network is formed and used for a relatively long period of time, and, therefore, the total computing time of edge computing is not considered. However, edge computing be initiated and discontinued at any time due to the completion of running an application or mobility of the edge nodes. Therefore, a finite total time period can be used in edge computing. The study of such ephemeral edge computing scenarios with strict time constraints remains an open problem.

- Although a number of prior works [52, 53, 55, 58–62] investigated computational task offloading schemes in edge computing, these works assumed that the tasks are offloaded to the base stations connected to edge computing servers for processing. To further reduce the communication latency, the computational tasks can be offloaded and processed by using a D2D-type communications.

- In edge computing, edge nodes can further reduce their latency by caching the input data needed to process their computational operations. In all of these existing works on caching for edge computing [44, 74–77], it is generally assumed that the operation of the application running on the edge node has one corresponding input data given by either a single file or a set of files. When the computational operation of the application running on the edge node can have one corresponding input data, the goal is to fetch the necessary input data, and this caching technique can be viewed as data caching. However, for a given computational operation, multiple files can possibly be used for processing.

- Existing works such as [74] and [76] assume that information on the requested operation is completely known. However, in practice, a user can randomly use any application and arbitrarily request an operation in the application. Thus, the requested operation changes instantaneously and, hence, the sequential arrival of operations can be uncertain.

In summary, despite the existence of a notable body of work on edge computing, this existing literature is largely limited in scope and does not provide the necessary fundamentals needed to understand the full potential of wireless networking using edge computing.

### 1.4 Online Optimization Frameworks for Edge Computing

Edge computing will generally need to operate in highly dynamic networking environments. For instance, distributed computing can be performed among heterogeneous edge nodes such as smartphones, tablets, and other IoT devices.
and embedded systems. Therefore, co-existing edge nodes may have different computing resources. When those edge nodes participate in distributed computing, each edge node is able to individually control and manage its computing resources. In consequence, the amount of shared computing resources of an edge node can be controlled by each individual edge node. In this case, the edge network will generally not be able to know, in advance, the available computing resource shared by each edge node. Also, edge nodes such as handheld devices and vehicles can be mobile. While edge nodes are moving, if an edge node moves beyond a maximum communication distance, it will no longer be able to join the edge computing network. Therefore, edge nodes can dynamically join and leave a network and, thus, it is challenging to know, a priori, the full information on the location and the future availability of different edge nodes. Thus, complete information on the edge computing environment is not always known to the involved edge nodes. Under such lack of information availability, edge nodes will not be able to use conventional offline optimization techniques, for computing, task distribution, or caching purposes, as such techniques typically require at least some form of information availability (full information or statistical) at the level of each edge node. Thus, to capture the dynamically varying and largely uncertain environment of edge networks, one can rely on the powerful tools of online optimization [89]. In online optimization problems, newly updated information is revealed to the system in a sequential manner. The sequentially arriving information becomes the input of an online problem. Thus, unlike offline optimization problems, online problems can be updated according to the input. Therefore, when an input is initially unavailable and revealed sequentially, an online algorithm must be used for decision making and optimization purposes. Hence, for an online cost minimization problem, when an input set \( I \) is given in an online manner and the online algorithm yields feasible output \( O_A \), the objective function can be shown as \( C(I, O_A) \) where \( C \) is a function of the online input and the output of the algorithm [90].

When an online algorithm is used to solve an online optimization problem, competitive analysis can be used to measure the performance of an online algorithm. Using competitive analysis, the performance of the optimal offline algorithm is compared to the performance of the online algorithm, and the ratio between online and offline algorithms is known as a competitive ratio [90]. For instance, for a cost minimization problem, the costs of an online algorithm and optimal offline algorithm are denoted by \( ALG(I) = C(I, O_A) \) and \( OPT(I) = \min C(I, O_A) \), respectively. Then, the competitive ratio is \( c \) satisfying \( c \geq \frac{ALG(I)}{OPT(I)} \forall I \). Whenever \( c = 1 \), then the online algorithm and offline algorithm achieve the same performance. However, optimality cannot be readily achieved in online since the online algorithm is run without having the complete knowledge about the future events. Thus, the goal of developing online algorithms is to minimize the value of \( c \) in order to achieve a suboptimal solution whose performance is as close as possible to the optimal, offline approach. Also, when an input is given, the decision should be determined by an online algorithm before the next input arrives. In practice, the input arrival interval can be very short. In that case, the online algorithm needs to have low time complexity to make a decision in real time. Particularly, using a low time-complexity online algorithm, e.g., provided in essential literature [91-94], can further reduce the decision-making latency, thus helping to achieve the ultra-low latency in edge computing. This provides a key rationale for adopting online optimization solutions for distributed and real-time edge computing.

As will be evident from the rest of this dissertation, online optimization algorithms can play a central role in developing a low-complexity optimization framework that makes an on-the-fly decision in a highly dynamic environment where the information about the environment has to be known and revealed sequentially.

### 1.5 Summary of Contributions

The main contribution of this dissertation is to develop analytical foundation for deployment, performance analysis, and optimization of edge computing in online scenarios. In particular, we propose various distributed, low-complexity online frameworks for edge network formation, computational task allocation, radio resource allocation, caching data, and scheduling the on/off status of self-powered edge devices. By using the proposed frameworks, the performance of edge computing systems can be optimized in terms of communication and computation latency, sum data rate, and energy consumption, while considering the uncertainties innate in edge nodes, such as incomplete information on
their channel gains, computing capabilities, availabilities of joining edge computing, requested computational tasks, and stored energy levels. To achieve these contributions, this dissertation will rely on a number of analytical techniques that include competitive analysis of online algorithms, numerical optimization methods, and online optimization frameworks such as the secretary problem, ski-rental problem, and online weighted bipartite matching problem. Indeed, using such advanced mathematical tools, this dissertation will provide in-depth analytical foundations and efficient online and offline algorithms to design, optimize, and operate edge computing over wireless networks. In summary, our contributions are given as follows:

1.5.1 Distributed Edge Network Formation with Minimal Latency

As mentioned in Chapter 1.1, edge computing is emerging as a promising paradigm to perform distributed, low-latency computation by jointly exploiting the radio and computing resources of end-user devices and cloud servers. However, the dynamic and distributed formation of local edge networks is highly challenging due to the unpredictable arrival and departure of neighboring edge nodes. Therefore, a given edge node must properly select a set of neighboring nodes and intelligently offload its computational tasks to this set of neighboring edge nodes and the cloud in order to achieve low-latency transmission and computation. In Chapter 2, the problems of edge network formation and task distribution are jointly investigated while considering a hybrid edge-cloud architecture. The goal is to minimize the maximum computational latency by enabling a given edge node to form a suitable edge network and optimize the task distribution, under uncertainty on the arrival process of neighboring edge nodes. To solve this problem, a novel online optimization framework is proposed in which the neighboring nodes are selected by using a threshold-based online algorithm that uses a target competitive ratio, defined as the ratio between the latency of the online algorithm and the offline optimal latency. The proposed framework repeatedly updates its target competitive ratio and optimizes the distribution of the edge node’s computational tasks in order to minimize latency. Simulation results show that, for specific settings, the proposed framework can successfully select a set of neighboring nodes while reducing latency by up to 19.25% compared to a baseline approach based on the well-known online secretary framework. The results also show how, using the proposed framework, the computational tasks can be properly offloaded between the edge network and a remote cloud server in different network settings.

1.5.2 Ephemeral Edge Computing in the IoT

In the IoT environment, edge computing can be initiated at anytime and anywhere. However, in an IoT, edge computing sessions are often ephemeral, i.e., they last for a short period of time and can often be discontinued once the current application usage is completed or the edge devices leave the system due to factors such as mobility. Therefore, in Chapter 3, the problem of ephemeral edge computing in an IoT is studied by considering scenarios in which edge computing operates within a limited time period. To this end, a novel online framework is proposed in which a source edge node offloads its computing tasks from sensors within an area to neighboring edge nodes for distributed task computing, within the limited period of time of an ephemeral edge computing system. The online nature of the framework allows the edge nodes to optimize their task allocation and decide on which neighbors to use for task processing, even when the tasks are revealed to the source edge node in an online manner, and the information on future task arrivals is unknown. The proposed framework essentially maximizes the number of computed tasks by jointly considering the communication and computation latency. To solve the problem, an online greedy algorithm is proposed and solved by using the primal-dual approach. Since the primal problem provides an upper bound of the original dual problem, the competitive ratio of the online approach is analytically derived as a function of the task sizes and the data rates of the edge nodes. Simulation results show that the proposed online algorithm can achieve a near-optimal task allocation with an optimality gap that is no higher than 7.1% compared to the offline, optimal solution with complete knowledge of all tasks.
1.5.3 Blockchain Systems with Wireless Mobile Miners

In Chapter 4, a novel framework that uses wireless mobile miners (MMs) for computation purposes in a blockchain system is proposed. In the introduced system, the blockchain ledger is located at the communication nodes (CNs), and the MMs associated with CNs process the blockchain’s computational tasks of a consensus mechanism, such as proof-of-work (PoW) or proof-of-stake, to verify the originality of the data. For instance, in the case of PoW, the MM that is the first to finish its PoW will receive a reward by sending its computing result to the CNs that are connected to other MMs. In the considered scenario, a blockchain forking event occurs if the MM having the shortest PoW delay fails to be the first to update its computing result to other MMs. To operate blockchains over such a wireless mobile network with minimum forking events, it is imperative to maintain low-latency wireless communications between MMs and CNs that store the blockchain ledgers. To analyze the sensitivity of the system to latency, the probability of occurrence of a forking event is theoretically derived. Also, in mobile blockchain using MMs, minimizing energy consumption required for networking and computation is essential to extend the operation time of MMs. Hence, the average energy consumption of an MM is derived as a function of the system parameters such as the number of MMs and power consumed by the computing, transmission, and mobility processes of the MMs. Simulation results verify the analytical derivations and show that using a larger number of MMs can reduce the energy consumption by up to 94.5% compared to a blockchain system with a single MM.

1.5.4 Computational Caching in Edge Networks

Enabling effective computation for emerging applications such as augmented reality or virtual reality via edge computing requires processing data with low latency. In Chapter 5, a novel computational caching framework is proposed to minimize edge latency by storing and reusing intermediate computation results (IRs). Using this proposed paradigm, an edge node can store IRs from previous computations and can also download IRs from neighboring nodes at the expense of additional transmission latency. However, due to the unpredictable arrival of the future computational operations and the limited memory size of the edge node, it is challenging to properly maintain the set of stored IRs. Thus, under uncertainty of future computation, the goal of the proposed framework is to minimize the sum of the transmission and computational latency by selecting the IRs to be downloaded and stored. To solve the problem, an online computational caching algorithm is developed to enable the edge node to schedule, download, and manage IRs compute arriving operations. Competitive analysis is used to derive the upper bound of the competitive ratio for the online algorithm. Simulation results show that the total latency can be reduced up to 26.8% by leveraging the computational caching method when compared to the case without computational caching.

1.5.5 ON/OFF Scheduling of Energy Harvesting Base Stations

The co-existence of small cell base stations (SBSs) with conventional macrocell base station is a promising approach to boost the capacity and coverage of cellular networks. However, densifying the network with a viral deployment of SBSs can significantly increase energy consumption. To reduce the reliance on unsustainable energy sources, one can adopt self-powered SBSs that rely solely on energy harvesting. Due to the uncertainty of energy arrival and the finite capacity of energy storage systems, self-powered SBSs must smartly optimize their ON and OFF schedule. In Chapter 6, the problem of ON/OFF scheduling of self-powered SBSs is studied, in the presence of energy harvesting uncertainty with the goal of minimizing the operational costs consisted of energy consumption and transmission delay of a network. For the original problem, we show an algorithm can solve the problem in the illustrative case. Then, to reduce the complexity of the original problem, an approximation is proposed. To solve the approximated problem, a novel approach based on the ski rental framework, a powerful online optimization tool, is proposed. Using this approach, each SBS can effectively decide on its ON/OFF schedule autonomously, without any prior information on future energy arrivals. By using competitive analysis, a deterministic online algorithm (DOA) and a randomized online algorithm (ROA) are developed. The ROA is then shown to achieve the optimal competitive ratio in the approximation
problem. Simulation results show that, compared to a baseline approach, the ROA can yield performance gains reaching up to 15.6% in terms of reduced total energy consumption of SBSs and up to 20.6% in terms of per-SBS network delay reduction. The results also shed light on the fundamental aspects that impact the ON time of SBSs while demonstrating that the proposed ROA can reduce up to 69.9% the total cost compared to a baseline approach.

1.5.6 Energy-Efficient Networking with Reconfigurable Intelligent Surfaces

When deployed as reflectors for existing wireless BSs, RISs can be a promising approach to achieve high spectrum and energy efficiency. However, due to the large number of RIS elements, the joint optimization of the BS and reflector RIS configuration is challenging. In essence, the BS transmit power and RIS’s reflecting configuration must be optimized so as to improve users’ data rates and reduce the BS power consumption. In Chapter 7, the problem of energy efficiency optimization is studied in an RIS-assisted cellular network endowed with an RIS reflector powered via energy harvesting technologies. The goal of this proposed framework is to maximize the average energy efficiency by enabling a BS to determine the transmit power and RIS configuration, under uncertainty on the wireless channel and harvested energy of the RIS system. To solve this problem, a novel approach based on deep reinforcement learning is proposed, in which the BS receives the state information, consisting of the users’ channel state information feedback and the available energy reported by the RIS. Then, the BS optimizes its action composed of the BS transmit power allocation and RIS phase shift configuration using a neural network. Due to the intractability of the formulated problem under uncertainty, a case study is conducted to analyze the performance of the studied RIS-assisted downlink system by asymptotically deriving the upper bound of the energy efficiency. Simulation results show that the proposed framework improves energy efficiency up to 77.3% when the number of RIS elements increases from 9 to 25.

1.5.7 Channel Allocation for Device-to-Device Communications

Full-duplex D2D communications over cellular networks is a promising solution for maximizing wireless spectral efficiency. However, in practice, due to the unpredictable arrival of D2D users, the base station (BS) must smartly allocate suitable channels to arriving D2D pairs. In Chapter 8, the problem of dynamic channel allocation is studied for full-duplex D2D networks. In particular, the goal of the proposed approach is to maximize the system sum-rate under complete uncertainty on the arrival process of D2D users. To solve this problem, a novel approach based on an online weighted bipartite matching is proposed. To find the desired solution of the channel allocation problem, a greedy online algorithm is developed to enable the BS to decide on the channel assignment for each D2D pair, without knowing any prior information on future D2D arrivals. For an illustrative case study, upper and lower bounds on the competitive ratio that compares the performance of the proposed online algorithm to that of an offline algorithm are derived. Simulation results show that the proposed online algorithm can achieve a near-optimal sum-rate with an optimality gap that is no higher than 8.3% compared to the offline, optimal solution that has complete knowledge of the system.

1.6 List of Publications

As a byproduct of the above contributions, thus far, this dissertation has led to the following key publications:

1.6.1 Journal Publications


### 1.6.2 Conference Publications


Chapter 2

Distributed Edge Network Formation with Minimal Latency

2.1 Background, Related Works, and Contributions

As mentioned in Section 1, when the computing tasks are offloaded from the sensor layer to the edge and cloud layers, edge computing faces a number of challenges such as edge network formation and radio/computing resource allocation [95]. In particular, it is challenging for edge nodes to dynamically form and maintain an edge network that they can use for offloading their task. This challenge is exacerbated by the fact that edge computing devices are inherently mobile and will join/leave a network sporadically [34]. Moreover, to efficiently use the computing resource pool of the edge network, novel resource management schemes for the hybrid edge-cloud network architecture are needed [27]. To reap the benefits of edge networks, many architectural and operational challenges must be addressed [8,35–47]. Also, once an edge network is formed, the next step is to share resources and tasks among edge nodes as studied in [39–47]. However, the existing literature [37,41–45] assumes full information knowledge for edge network formation and relies on simple delay models. Therefore, when the presence of a neighboring edge node to which one can offload tasks is unpredictable, unlike the prior works [37, 41–45], our goal is to design an online approach to enable an on-the-fly formation of the edge network, under uncertainty, while minimizing the computational latency given an end-to-end latency model.

The main contribution of this chapter is a novel framework for online edge network formation and task distribution in a hybrid edge-cloud network. This framework allows any given edge node to dynamically construct an edge network by selecting the most suitable set of neighboring edge nodes in presence of uncertainty on the arrival order of neighboring edge nodes. The edge node can jointly use its edge network as well as a distant cloud server to compute given tasks. We formulate an online optimization problem whose objective is to minimize the maximum computational latency of all edge nodes by properly selecting the set of edge nodes to which computations will be offloaded while also properly distributing the tasks among those edge nodes and the cloud. To solve this problem without any prior information on the order of future arrivals of edge nodes, we propose an online optimization framework that achieves a target competitive ratio – defined as the ratio between the latency achieved by the proposed algorithm and the optimal latency that can be achieved by an offline algorithm. In the proposed framework, an online algorithm is used to form an edge network when the neighboring nodes arrives in an online manner, the task distribution is optimized among the nodes on the formed network, and the target competitive ratio is repeatedly updated. We show the target competitive ratio can be achieved by iteratively running the proposed algorithm. Simulation results show that the proposed framework can achieve a target competitive ratio of 1.21 in a given simulation scenario. For a specific simulation setting, simulation
results show that the proposed algorithm can reduce the latency by up to 19.25\% compared to the baseline approach that is a modified version of the popular online secretary algorithm [80]. Therefore, the proposed framework is shown to be able to find a suitable competitive ratio that can reduce the latency of edge computing while properly selecting the neighboring edge nodes that have high performance and suitably distributing tasks across edge nodes and a cloud server.

The rest of this chapter is organized as follows. In Section 2.2, the system model is presented. We formulate the online problem in Section 2.3. In Section 2.4, we propose our online optimization framework to solve the problem. In Section 2.5, simulation results are carried out to evaluate the performance of our proposed framework. Conclusions are drawn in Section 2.6.

### 2.2 System Model

Consider an edge network consisting of a sensor layer, an edge layer, and a cloud layer as shown in Fig. 2.1. In this system, the sensor layer includes smart and small-sized IoT sensors with limited computational capability. Therefore, when sensors generate the computational tasks, the sensors’ tasks are offloaded to the edge and cloud layers for purposes of remote distributed computing. Similarly, cloud tasks can also be offloaded to the edge layer. In our model, the cloud layer can be seen as the conventional cloud computing center. The edge layer refers to the set of IoT devices (also called edge nodes) that can perform edge computing jobs such as storing data and computing tasks. The edge nodes can have different hardware types and can be owned by different organizations. We assume that various kinds of sensors send their task data to a certain edge node \( i \), and the data arrival rate to this node is \( x_i \) packets per second where a task packet has a size of \( K \) bits\(^1\). Edge node \( i \) performs the roles of collecting, storing, controlling, and processing the task data from the sensor layer, as is typical in practical edge networking scenarios [4]. In our architecture, for efficient computing, edge node \( i \) must cooperate with other neighboring edge nodes and the cloud data center. We consider a network having a set \( N \) of \( N \) edge nodes other than edge node \( i \). For a given edge node \( i \), we focus on the edge computing case in which edge node \( i \) builds a network with a subset \( J \subset N \) of \( J \) neighboring edge nodes. Also, since the cloud is typically located at a remote location, edge node \( i \) must access the cloud via wireless communication links using a cellular base station \( c \).

\(^1\) The initial edge node can gather data from any other node, including sensors or a cloud.
Table 2.1: Summary of notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i$</td>
<td>Index of initial edge node</td>
</tr>
<tr>
<td>$j$</td>
<td>Index of neighboring edge nodes in $\mathcal{J}$</td>
</tr>
<tr>
<td>$c$</td>
<td>Index of cloud</td>
</tr>
<tr>
<td>$J =</td>
<td>\mathcal{J}</td>
</tr>
<tr>
<td>$x_i$</td>
<td>Total task arrival rate from sensors to node $i$</td>
</tr>
<tr>
<td>$\alpha_{k \in {i,j,c}}$</td>
<td>Tasks offloaded toward $k$</td>
</tr>
<tr>
<td>$\mu_{ij}$</td>
<td>Edge transmission service rate from $i$ to $j$</td>
</tr>
<tr>
<td>$\mu_c$</td>
<td>Cloud transmission service rate</td>
</tr>
<tr>
<td>$\mu_i$</td>
<td>Computing service rate of edge node $i$</td>
</tr>
<tr>
<td>$\mu_j$</td>
<td>Computing service rate of edge node $j$</td>
</tr>
<tr>
<td>$\frac{1}{\omega_{k \in {i,j,c}}}$</td>
<td>Processing speed of node $k$</td>
</tr>
<tr>
<td>$n$</td>
<td>Arrival order</td>
</tr>
<tr>
<td>$K$</td>
<td>Size of a task packet</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>Target competitive ratio</td>
</tr>
</tbody>
</table>

Once the initial edge node $i$ receives tasks that arrive with the rate of $x_i$ packets per second, it assigns a fraction of $x_i$ to other nodes. Then, each node within the considered edge-cloud network will locally compute the assigned fraction of $x_i$. The fraction of tasks locally computed by edge node $i$ is $\lambda_i(\alpha_i) = \alpha_i x_i$. Then, the task arrival rate offloaded from edge node $i$ to edge node $j \in \mathcal{J}$ is $\lambda_{ij}(\alpha_{ij}) = \alpha_{ij} x_i$. Therefore, the task arrival rate processed at the edge layer is $(\alpha_i + \sum_{j \in \mathcal{J}} \alpha_{ij}) x_i$. The number of remaining tasks $\lambda_c(\alpha_c) = \alpha_c x_i$ will then be offloaded to the cloud. When edge node $i$ makes a decision on the distribution of all input tasks $x_i$, the task distribution variables are represented as vector $\alpha = [\alpha_i, \alpha_c, \alpha_1, \ldots, \alpha_{ij}, \ldots, \alpha_{ij}]$ with $\sum_{j \in \mathcal{J}} \alpha_{ij} + \alpha_i + \alpha_c = 1$. Naturally, the total task arrival rate that arrives at edge node $i$ will be equal to the sum of the task arrival rates assigned to all computation nodes in the edge and cloud layers. Also, to model the random arrival of tasks from the sensors to edge node $i$, the total task arrival rate arriving at edge node $i$ can be modeled by a Poisson process [46]. The tasks offloaded to the edge nodes and the cloud also follow a Poisson process if the tasks are randomly scheduled in a round robin fashion [96]. Also, the initial edge node can determine the transmission order of the task packets offloaded from the sensor layer. Therefore, in future work, if the tasks offloaded from the sensor layer have different service-level latency requirements, the initial edge node can prioritize urgent task packets in its queue.

When the tasks arrive from the sensors to edge node $i$, they are first saved in edge node $i$’s storage, incurring a waiting delay before they are transmitted and distributed to other nodes (edge or cloud). This additional delay pertains to the transmission from edge node $i$ to $c$ or $j$ and can be modeled using a transmission queue. Moreover, when the tasks arrive at the destination, the latency required to perform the actual computations will be captured by a computation queue. In Fig. 2.1, we show examples of both queue types. For instance, for transmission queues, edge node $i$ must maintain transmission queues for each edge node $j$ and the cloud $c$. For computation, each edge node has a computation queue. To model the transmission queue, the tasks are transmitted to edge node $j$ over a wireless channel. Then, under the assumption of an additive white Gaussian noise channel capacity, the service rate (in packets per second) can be defined by

$$\mu_{ij} = \frac{W_l}{K} \log_2 \left(1 + \frac{g_{ij} P_{tx,i}}{W_l N_0}\right),$$  \hspace{1cm} (2.1)$$

where $g_{ij}$ is the channel gain between edge nodes $i$ and $j$ with $d_{ij}$ being the distance between them. If $d_{ij} \leq 1$ m, $g_{ij} = \beta_1$, and, if $d_{ij} > 1$ m, $g_{ij} = \beta_1 d_{ij}^{-\beta_2}$ where $\beta_1$ and $\beta_2$ are, respectively, the path loss constant and the path loss exponent. Also, $P_{tx,i}$ is the transmission power of edge node $i$ and $N_0$ is the noise power spectral density. The bandwidth per node is given by $W_l$ where $l = 1$ and $2$ indicate, respectively, two types of bandwidth allocation.
schemes: equal allocation and cloud-centric allocation. For equal bandwidth allocation, all nodes in the network will be assigned equal bandwidth, i.e., $W_1 = \frac{B}{\sum_{j=1}^{J+1}}$ where the total bandwidth $B$ is equally shared by $J+1$ nodes that include $J$ neighboring edge nodes and the connection to the cloud via the base station. For the cloud-centric bandwidth allocation, the bandwidth allocated to the cloud is twice that of the bandwidth used by an edge node, i.e., the cloud and the edge node will be assigned the bandwidth $\frac{2B}{J+2}$ and $\frac{B}{J+2}$, respectively.

Since the tasks arrive according to a Poisson process, and the transmission time in (2.1) is deterministic, the latency of the transmission queue can be modeled as an M/D/1 system\(^3\) [96]:

\[ T_j(\lambda_j(\alpha_{ij}), \mu_{ij}) = \frac{\lambda_j(\alpha_{ij})}{2\mu_{ij}(\mu_{ij} - \lambda_j(\alpha_{ij}))} + \frac{1}{\mu_{ij}}, \]  

where the first term is the waiting time in the queue at edge node $i$, and the second term is the transmission delay between edge nodes $i$ and $j$. Similarly, when the tasks are offloaded to the cloud, the transmission queue delay will be:

\[ T_c(\lambda_c(\alpha_c), \mu_c) = \frac{\lambda_c(\alpha_c)}{2\mu_c(\mu_c - \lambda_c(\alpha_c))} + \frac{1}{\mu_c}, \]  

where the service rate $\mu_c$ between edge node $i$ and cloud $c$ is given by (2.1) where edge node $j$ is replaced with cloud $c$.

Next, we define the computation queue. When an edge node needs to compute a task, this task will experience a waiting time in the computation queue of this edge node due to a previous task that is currently being processed. Since an edge node $j$ receives tasks from not only edge node $i$ but also other edge nodes and sensors, the task arrival process can be approximated by a Poisson process by applying the Kleinrock approximation [96]. Therefore, the computation queue can be modeled as an M/D/1 queue and the latency of edge node $j$'s computation will be:

\[ S_j(\lambda_j(\alpha_{ij})) = \frac{\lambda_j(\alpha_{ij})}{2\mu_j(\mu_j - \lambda_j(\alpha_{ij}))} + \frac{1}{\mu_j} + \omega_j \lambda_j(\alpha_{ij}), \]  

where the first term is the waiting delay in the computation queue, the second term is the delay for fetching the proper application needed to compute the task, and the third term is a function of the processor delay implying the processing delay for the task. The delay of this fetching procedure depends on the performance of the node’s hardware which is a deterministic constant that determines the service time of the computation queue. In the first and second terms of (2.4), $\mu_j$ is a parameter related to the overall hardware performance of edge node $j$. In the third term, $\omega_j \lambda_j(\alpha_{ij})$ is the actual computation time of the task with $\omega_j$ being a constant time needed to compute a task. For example, $1/\omega_j$ can be proportional to the CPU clock frequency of edge node $j$. $\omega_j \lambda_j(\alpha_{ij})$ implies that the delay needed to compute a task at a given node can increase with the task arrival rate since the number of concurrently running tasks increases with the task arrival rate. The increased number of the concurrently running tasks also increases the context switching delay that affects the computing delay. For edge node $j \in J$, it is assumed that the maximum of computing service rate and processing speed are given by $\mu_j$ and $1/\omega_j$, respectively. This information can be known in advance if the manufacturers of edge devices can provide the hardware performance in the database. Then, when edge node $i$ locally computes its assigned tasks $\lambda_i(\alpha_i)$, the latency will be:

\[ S_i(\lambda_i(\alpha_i)) = \frac{\lambda_i(\alpha_i)}{2\mu_i(\mu_i - \lambda_i(\alpha_i))} + \frac{1}{\mu_i} + \omega_i \lambda_i(\alpha_i), \]  

where $\mu_i$ is the computing service rate of edge node $i$ (dependent on hardware performance) and $\omega_i \lambda_i(\alpha_i)$ is the edge node $i$’s computing time. Since the cloud is equipped with more powerful and faster hardware than the edge node, the waiting time at the computation queue of the cloud can be ignored. This implies that the cloud initiates the

\(^{2}\)The problem of joint bandwidth optimization and edge computing can be subject for future work.

\(^{3}\)Instead of M/D/1 queueing, other delay models can be used to account for other characteristics, such as different packet size or finite buffer size.
computation for the received tasks without queueing delay; thus, we only account for the actual computing delay. As a result, when tasks are computed at the cloud, the computing delay at the cloud will be:

\[ S_c(\lambda_c) = \omega_c \lambda_c(\alpha_c). \]  

(2.6)

In essence, if a task is routed to the cloud \( c \), the latency will be

\[ D_c(\lambda_c(\alpha_c), \mu_c) = T_c(\lambda_c(\alpha_c), \mu_c) + S_c(\lambda_c(\alpha_c)). \]  

(2.7)

Also, if a task is offloaded to edge node \( j \), then the latency can be defined as the sum of the transmission and computation queueing delays:

\[ D_j(\lambda_{ij}(\alpha_{ij}), \mu_{ij}) = T_j(\lambda_{ij}(\alpha_{ij}), \mu_{ij}) + S_j(\lambda_{ij}(\alpha_{ij})). \]  

(2.8)

Furthermore, when edge node \( i \) computes tasks locally, the latency will be:

\[ D_i(\lambda_i(\alpha_i)) = S_i(\lambda_i(\alpha_i)), \]  

(2.9)

since no transmission queue is necessary for local computing. Since \( x_i \) is constant, \( \lambda_{k \in \{i,ij,c\}} \) is only dependent to \( \alpha_k \). From now on, for notational simplicity, \( \lambda_k(\alpha_k) \) is presented by \( \lambda_k \). Given this model, in the next section, we formulate an online latency minimization problem to study how an edge network can be formed and how tasks are effectively distributed in the edge network.

### 2.3 Problem Formulation

In distributed edge computing, the maximum latency of computing nodes must be minimized for effective distributed computing. To minimize the maximum latency, edge node \( i \) must opportunistically find neighboring nodes to form an edge network and carry out the process of task offload. In practice, such neighbors will dynamically join and leave the system. Also, the neighbors have to process their existing workloads [97]. As a result, the initial edge node \( i \) will be unable to know a priori whether an adjacent edge node will be available to assist it with its computation by sharing the communication and computational resources. Moreover, the total number of neighboring edge nodes as well as their locations and their available computing resources are unknown and highly unpredictable. Under such uncertainty, jointly optimizing the edge network formation and task distribution processes is challenging since selecting neighboring edge nodes must account for potential arrival of new edge nodes that can potentially provide a higher data rate and stronger computational capabilities. To cope with the uncertainty of the neighboring edge node arrivals while considering the data rate and computing capability of current and future edge nodes, we introduce an online optimization scheme that can handle the problem of edge network formation and task distribution under uncertainty.

We formulate the following online edge network formation and task distribution problem whose goal is to minimize the maximum latency when computing a new task that arrives at edge node \( i \):

\[
\min_{\alpha \in \Delta, \sigma} \max \left( D_i(\lambda_i), \ D_c(\lambda_c, \mu_c), \ D_j(\lambda_{ij}, \mu_{ij}) \right),
\]

\[
\text{s.t.} \quad \alpha_i + \alpha_c + \sum_{j \in \sigma} \alpha_{ij} = 1,
\]

\[
\alpha_i \in [0,1], \alpha_c \in [0,1], \alpha_{ij} \in [0,1], \forall j \in \mathcal{J}_\sigma \subset \mathcal{N}_\sigma,
\]

\[
\alpha_i x_i \leq \mu_i, \ \alpha_c x_i \leq \mu_c, \ \alpha_{ij} x_i \leq \mu_j, \ \forall j \in \mathcal{J}_\sigma,
\]

\[
|\mathcal{N}_\sigma| \leq N
\]
Since our goal is to minimize the worst-case latency among the edge nodes and the cloud, any task can be processed with a low latency regardless of which node actually computes the task\(^4\). By using an auxiliary variable \(u\), problem (2.10) can be transformed into the following:

\[
\begin{align*}
\min_{J_{\sigma}, c_{\alpha}} & \quad u, \\
\text{s.t.} & \quad u \geq \max(D_i(\lambda_i), \ D_c(\lambda_c, \mu_c), \ D_j(\lambda_{ij}, \mu_{ij})), \\
& \quad (2.11), (2.12), (2.13), (2.14),
\end{align*}
\]  

where \(u\) is the maximum latency of the edge network. In (2.15), \(u\) represents the largest value among \(D_i(\lambda_i), D_c(\lambda_c, \mu_c),\) and \(D_j(\lambda_{ij}, \mu_{ij})\). Then, minimizing \(u\) is equivalent to minimizing the \(\max\) function in (2.10). Hence, problems (2.10) and (2.15) are equivalent.

In constraints (2.11) and (2.12), all tasks arriving at edge node \(i\) are offloaded among the computing nodes in the edge network. Due to constraint (2.13), the tasks offloaded to a node cannot exceed the service rate of the computing node. In this problem, the initial edge node \(i\) determines the set of neighboring edge nodes \(J_{\sigma}\) when they arrive online and the task distribution vector \(\alpha\) so as to minimize the computing latency. Edge node \(i\) will observe a total number of \(N\) arriving edge nodes due to constraint (2.14). Edge node \(i\) has to make a decision on network formation and task distribution while observing \(N\) neighboring nodes. As the number of observations increases, edge node \(i\) may be able to discover neighboring edge nodes that have higher performance. However, due to constraint (2.14), edge node \(i\) cannot wait to observe an infinite number of neighboring edge nodes. Thus, while observing up to \(N\) arriving edge nodes, edge node \(i\) should select \(J \leq N\) neighboring edge nodes to minimize (2.10).

In our model, we assume that edge node \(i\) does not have any prior information on the neighboring edge nodes given by set \(N_{\sigma}\), and the information about each neighboring node is collected sequentially. Such random arrival sequence is denoted by \(\sigma = \sigma_1, \ldots, \sigma_n, \ldots, \sigma_N\) where the arrival of \(n\)-th neighboring node is shown as \(\sigma_n\). For example, a smartphone can choose to become an edge node spontaneously if it decides to share its resources. In practice, to discover the neighboring nodes, the edge nodes can use the node discovery mechanisms implemented in D2D networks [8].

When edge node \(i\) does not have complete information on other edge nodes, the nodes in \(N_{\sigma}\) arrive at edge node \(i\) in a random order, and index \(n\) can be the arriving order of the neighboring edge nodes. At the arrival of a neighboring node, the arrival order \(n\) increases by one; thus, \(n\) captures the time order of arrival. At time \(n\), node \(n\) can transmit a beacon signal to edge node \(i\) to indicate its willingness to join the network of edge node \(i\). The beacon signal can include an information tuple on node \(n\) that includes the distance \(d_{in}\), computing service rate \(\mu_n\), and the processing speed \(\omega_n\). At each time that \(\sigma_n\) is known, e.g., by receiving the beacon signal, edge node \(i\) will now have information on these parameters that pertain to node \(n\) [98]. Therefore, edge node \(i\) only knows the information on the nodes that have previously arrived (as well as the current node).

When edge node \(i\) observes \(\sigma_n\) and has knowledge of the \(n\)-th neighboring node, it has to make an online decision whether to select node \(n\). If edge node \(n\) is chosen by the initial edge node \(i\), it is indexed by \(j\) and included in a set \(J_{\sigma}\) which is a subset of \(N_{\sigma}\). Otherwise, edge node \(i\) will no longer be able to select edge node \(n\) at a later time period since the latter can join another edge network or terminate its resource sharing offer to edge node \(i\). For notational simplicity, \(J_{\sigma}\) and \(N_{\sigma}\) are hereafter denoted as \(J\) and \(N\), respectively. edge node \(i\) will not be able to have complete information about all \(N\) neighboring nodes before all neighboring nodes are selected by edge node \(i\). Therefore, since edge node \(i\) cannot know any information on future edge nodes, it is challenging for the initial edge node \(i\) to form the edge network by determining \(J\).

Even when the information on each node is known to edge node \(i\), it is difficult to calculate the exact service rates of the edge node in the formulated problem. This is due to the fact that the service rate in (2.1), that includes the wireless data rate, is a function of the network size \(J\). As the number of nodes sharing their wireless bandwidth increases, the available channel bandwidth per node decreases, thus reducing the data rate. Therefore, unlike the constant parameters

\[^4\text{If the objective function is defined with a minimum function, the initial edge node will minimize the latency of only one node, and, therefore, it will increase the latency of other nodes.}\]
Gilsoo Lee  

Chapter 2.  

2.4 Task Distribution and Network Formation Algorithms

In our problem, edge node $i$ has to decide whether to admit each neighboring node as the different neighboring nodes arrive in a random order. This problem can be formulated as an online stopping problem. In such problems, such as online secretary problem [99], the goal is to develop online algorithms that enable a company to hire a number of employees, without knowing in which order they will arrive to the interview. To apply such known solutions from the stopping problems, the following assumptions are commonly needed. For instance, the number of hiring positions should be deterministic and given in the problem. Also, the decision maker should be able to decide the preference order among the candidates by comparing the values that can be earned by hiring candidates. Under these assumptions, online stopping algorithms can be used to select the best set of candidates in an online manner. In this regard, even though the structures of our edge network formation problem and the secretary problem are similar, the edge network formation case has different assumptions. First, the number of neighboring edge nodes is an optimization variable in our problem. Second, the latency of computing nodes that somewhat maps to the valuation of hiring candidates in the secretary problem is not constant. Moreover, in our problem, each neighboring edge node exhibits two types of latency: transmission latency and computing latency. As a result, it is challenging to define the preference order of the neighboring nodes as done in conventional online stopping problems. To address those challenges, we propose a new online optimization framework$^5$ that extends existing results from online stopping theory to accommodate the specific challenges of the edge network formation problem$^6$. 

Figure 2.2: Online optimization framework for edge network formation and task distribution.
2.4.1 Overview of the Proposed Optimization Framework

Problem (2.15) has two optimization variables \( J \) and \( \alpha \) that constitute the solutions of the network formation and task distribution problems, respectively. To solve (2.15), edge node \( i \) must first optimize the network formation by selecting the neighboring edge nodes, and then decide on its task distribution. This two-step process is required due to the fact that the computing resources of the edge nodes are unknown before the network is formed. The online optimization framework consists of three highly inter-related components as shown in Fig. 2.2. In the network formation stage, an online algorithm is used to find \( J \) by determining the minimal network size and, then, selecting the neighboring edge nodes within \( N \) observations to satisfy (2.14). After \( J \) is determined, the task distribution among the selected nodes is optimized by using an offline optimization method during the task distribution stage. The output of the task distribution stage is the task allocation vector \( \alpha \) that satisfies constraints (2.11), (2.12), and (2.13). Finally, we use a parameter update stage, during which the target performance parameter \( \gamma \) that will be used in the next iteration is updated in order to satisfy constraint (2.14). After repeatedly running three components of our framework, edge node \( i \) is able to form a network without any prior information on the neighboring nodes and also offload the tasks to the nodes on the edge network. This algorithm is shown to converge in Theorem 3.

The performance of our online optimization framework will be evaluated by using competitive analysis [90]. In this analysis, the performance is measured by the competitive ratio \( \gamma \) that is defined by

\[
1 \leq \frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \leq \gamma, \tag{2.17}
\]

where \( \text{ALG}(\sigma) \) denotes the latency achieved by the online algorithm and \( \text{OPT}(\sigma) \) is the optimal latency achieved by an offline algorithm. If the online algorithm finds the optimal solution, the online algorithm achieves \( \gamma = 1 \). However, since the online algorithm cannot have complete information, it is challenging to find the optimal solution in an online setting. Therefore, in an online minimization problem, the online algorithm should be able to achieve \( \gamma \) that is close to one. We use this notion of competitive ratio to design our online optimization framework.

The online optimization framework is summarized in the flow chart shown in Fig. 2.3. In the network formation stage, edge node \( i \) needs to select the set of neighboring edge nodes with high service rates and processing speeds to achieve a given value of \( \gamma \). At each iteration, to achieve a target competitive ratio \( \gamma \), edge node \( i \) determines the number of neighboring nodes \( J \) by using Phase 1 of Algorithm 2.1, and it sequentially observes the arrivals of a total of \( N \) neighboring edge nodes while making an online decision in Phase 2 of Algorithm 2.1. After the network formation stage is finished, the task distribution is optimized by the initial edge node in an offline manner. Then, edge node \( i \) checks whether the number of selected neighboring nodes is \( J \). For a small value of \( \gamma \), edge node \( i \) must find the neighboring nodes having a high computing service rate and processing speed so as to achieve low latency. Therefore, in this case, edge node \( i \) must observe a large number of neighboring nodes until \( J \) neighboring nodes are selected. Hence, \( N \) observations may not be sufficient to find \( J \) neighboring nodes. On the other hand, a large \( \gamma \) can allow the target latency to be less stringent, thus allowing the edge node \( i \) to select the neighboring nodes with fewer observations. To find the proper value of \( \gamma \), the proposed framework iteratively updates \( \gamma \). For instance, the value of \( \gamma \) can be set to one initially. Then, if a smaller \( \gamma \) cannot be achieved in the network formation stage at that iteration, the value of \( \gamma \) increases by a small constant \( \tau \). By repeatedly increasing \( \gamma \), the proposed framework can find the achievable value of \( \gamma \). In the next section, we present the details of the proposed online algorithm that exploits the updated value of \( \gamma \) for the network formation stage.

---

5The framework proposed in this work is different from the one in [80] since a different definition of transmission service rate in (2.1) and objective function in (2.10) are used here.

6Edge networks can be formed by using game-theoretic approaches such as coalitional games which require a complete knowledge of the exact utility functions [83]. However, such knowledge can be difficult to gather, since the initial edge node cannot have the complete information on the neighboring nodes in an online scenario, and, therefore, an online optimization framework is more apropos.
Phase 1 (in Alg. 1): Determining the network size
Check if $|J| < J^*$ and $n < N$

Yes
A neighboring edge node arrives.

No
Phase 2 (in Alg. 1): Network formation with online decision
Optimize task distribution
Check if $|J| = J^*$

Yes
Terminate if $\gamma$ converges.

No
Increase $\gamma \leftarrow \gamma + \tau$

START with initial $\gamma = 1$

Figure 2.3: Flow chart of the proposed framework for edge network formation and task distribution.
Algorithm 2.1 Online Edge Network Formation Algorithm

1: inputs: $N, \gamma, \mu_i, \omega_i, \omega_c, d_c, \hat{\mu}_{ij}(\hat{d}_{ij}), \bar{\mu}_j, \hat{\omega}_j$.

Phase 1: Calculate $\hat{\lambda}_{ij}$, $\hat{J}$, and $\hat{u}$.

2: initialize: $J = 0$, $n = 0$.

3: while $\Delta \geq 0$

4: $J \leftarrow J + 1$.

5: $\Delta \leftarrow \left[ D_j(\lambda_{ij}, \bar{\mu}_j) \right]_{|J|=J-1} - \left[ D_j(\hat{\lambda}_{ij}, \hat{\mu}_{ij}) \right]_{|J|=J}$.

6: end while

7: Find $\hat{\lambda}_{ij}$ by optimizing task distribution when $|\mathcal{J}| = J - 1$.

8: Set $\hat{J} = J - 1$ and $\hat{u} = \left[ D_j(\hat{\lambda}_{ij}, \hat{\mu}_{ij}) \right]_{|\mathcal{J}|=J-1}$.

Phase 2: Decide $\mathcal{J}$.

9: while $|\mathcal{J}| < \hat{J}$ and $n < N$

10: if $D_n(\hat{\lambda}_{ij}, \bar{\mu}_m) \leq \gamma \hat{u}$,

11: $\mathcal{J} \leftarrow \mathcal{J} \cup \{n\}$.

12: end if

13: $n \leftarrow n + 1$.

14: end while

2.4.2 Edge Network Formation: Online Approach

In problem (2.15), the decision on $\mathcal{J}$ faces two primary challenges: how many edge nodes are needed in the network and which edge nodes join the network (at which time). Since the transmission service rates are functions of the wireless bandwidth that can vary with the network size, the service rates of neighboring edge nodes cannot be calculated without having a fixed network size. Therefore, the proposed algorithm includes two phases as shown in Algorithm 2.1. The goal of the first phase is to determine the parameters including the network size and the temporal task distribution so that the parameters can be used in the second phase of Algorithm 2.1. Then, the second phase of Algorithm 2.1 allows edge node $i$ to make an online decision regarding the selection of an arriving node.

In the first phase of Algorithm 2.1, the goal is to determine the parameters that will be used in the second phase of Algorithm 2.1. In the given system model, a neighboring node will be referred to as ideal in terms of minimizing the latency in (2.15) if it has the highest computing service rate $\hat{\mu}_j$, processing speed $1/\hat{\omega}_j$, and transmission service rate $\hat{\mu}_{ij}$ when the distance between two edge nodes is $\hat{d}_{ij}$. Such an ideal node is denoted by $\hat{j}$. If a network is formed with nodes having high computing resources, a smaller network size can effectively minimize the latency. When the service rates of the nodes are divided by the smallest network size, the transmission service rates of the nodes also can be maximized, and, hence, the latency can be minimized. In the case in which the ideal nodes construct a network, the minimized latency of (2.15) is denoted by $\hat{u}$. Also, when the latency is $\hat{u}$, the corresponding number of neighboring nodes and task distribution are denoted by $\hat{J}$ and $\{\hat{\lambda}_i, \hat{\lambda}_c, \hat{\lambda}_{ij}\}$, respectively.

First phase: The first phase of Algorithm 2.1 is used to calculate $\hat{J}$ and $\hat{\lambda}_{ij}$. The latency in (2.15) decreases as the number of neighboring nodes increases since the computational load per node can be reduced. However, if the number of neighboring nodes becomes too large, the bandwidth per edge node will be smaller yielding lower transmission service rates for the nodes. Consequently, the latency can increase with the number of neighboring nodes, due to these bandwidth limitations. By using the relationship between network size and latency, the first phase of Algorithm 2.1 searches for $\hat{J}$ while increasing the network size incrementally, one by one. Once the number of neighboring users $\hat{J}$ that minimizes $\hat{u}$ is found, the tasks offloaded to each ideal node is denoted by $\hat{\lambda}_{ij}$. Therefore, we will have $\hat{J}$, $\hat{u}$, and $\hat{\lambda}_{ij}$ as the outputs from the first phase of Algorithm 2.1 that will be used in the second phase of Algorithm 2.1.
Second phase: In the second phase of Algorithm 2.1, edge node $i$ decides on whether to select each neighboring node or not, by using a threshold-based algorithm. Our algorithm uses a single threshold so that the latency of each arriving node can be compared with the threshold value. Since comparing two values is a simple operation having constant time complexity, a threshold-based algorithm can be executed with low latency. However, before the network formation process is completed, edge node $i$ is not able to know the optimal latency of each node, and, therefore, finding the distribution of tasks that must be offloaded to each node is not possible. Nonetheless, edge node $i$ must set a threshold before the first neighbor arrives. To this end, edge node $i$ sets this initial threshold by assuming that an equal amount of tasks, $\tilde{\lambda}_{ij}$, is offloaded to each one of the $J$ neighboring nodes. Thus, in our threshold-based algorithm, the threshold value is compared with the latency that results from offloading $\tilde{\lambda}_{ij}$ tasks. For example, when a neighboring node $n$ arrives, the algorithm compares the latency of node $n$, $D_n(\tilde{\lambda}_{ij}, \mu_n)$, to the threshold $\gamma \tilde{u}$. If the latency of node $n$ is smaller than the threshold, edge node $i$ will immediately select node $n$. This procedure is repeated until edge node $i$ observes $N$ arrivals and selects $J$ neighboring nodes. In the proposed algorithm, the initial edge node needs to discover the neighboring nodes and know the information on the communication and computational performance of the neighboring nodes. This procedure can use any node-discovery and message exchanging protocols developed for D2D communications or wireless sensor networks [100]. Also, our framework requires a low signaling and communication overhead since each neighboring node can transmit its location and computing speed using a very small packet after which the initial edge node transmits a decision on node selection using a single bit. After the edge network is formed, the task distribution is done to minimize latency. In the next section, we investigate the property of the optimal task distribution, and show that the threshold can satisfy (2.17).

2.4.3 Task Distribution: Offline Optimization

Once the nodes are selected to form a network, the task distribution can be performed using an offline optimization problem which can be solved using known algorithms such as the interior-point algorithm [101]. From problem (2.15), the following properties can be derived, for a given problem which can be solved using known algorithms such as the interior-point algorithm [101]. From problem (2.15), the following properties can be derived, for a given $J$.

Theorem 1. If there exists a task distribution $\alpha^*$ satisfying $u^* = D_i(\lambda_i) = D_c(\lambda_c, \mu_c) = D_j(\lambda_{ij}, \mu_{ij}), \forall j \in J$, then $\alpha^*$ is the unique and optimal solution of problem (2.10).

Proof. Let $\alpha$ be the initial task distribution, and assume that any other task distribution $\alpha'$ different from $\alpha$ is the optimal distribution. When $\alpha'$ is considered, we can find a certain node $A$ satisfying $\alpha'_A < \alpha_A$ where $\alpha'_A \in \alpha'$ and $\alpha_A \in \alpha$. This, in turn, yields $D_A(\alpha'_A) < D_A(\alpha_A)$. Due to the constraint (2.11), there exists another node $B$ such that $B \neq A$, $\alpha'_B > \alpha_B$, and $D_B(\alpha'_B) > D_B(\alpha_B)$ where $\alpha_B \in \alpha'$ and $\alpha_B \in \alpha$. Since $D_B(\alpha'_B) > D_B(\alpha_B) = D_A(\alpha_A) > D_A(\alpha'_A)$, we must decrease $\alpha'_B$ to minimize the maximum, i.e., $D_B(\alpha'_B)$. Thus, we can clearly see that $\alpha'$ is not optimal, and, thus, the initial distribution $\alpha$ is optimal.

Furthermore, $D_j(\lambda_{ij}, \mu_{ij})$ is a monotonically increasing function with respect to $\lambda_{ij} = x_i \alpha_{ij}$ since $\frac{\partial}{\partial \lambda_{ij}} D_j(\lambda_{ij}, \mu_{ij}) > 0$. Therefore, there are no more than two points of $\alpha^*$ that have the same $u^*$. Hence, the distribution $\alpha$ is unique and optimal.

Theorem 1 shows that the optimal solution of the offline latency minimization problem results in an equal latency for all edge nodes and the cloud on the network (whenever such a solution is feasible). Using the objective function in (2.10), the initial edge node minimizes the worst-case latency among the nodes. To that end, the initial edge node can decrease the task arrival rate of the node having the highest latency, but, in turn, the latency of other node increases. This is due to the fact that reducing one node’s task arrival rate leads to increase the other node’s arrival rate since we have $\sum_{j \in J} \lambda_{ij} + \lambda_i + \lambda_c = x$. Therefore, as shown in Theorem 1, an equal latency for all edge nodes and the cloud is obtained by repeatedly reducing the arrival rate of the node having the highest latency. According to Theorem 1, selecting the node that has high computing resources is beneficial to minimize latency. Once edge node $i$ determines
the task distribution, the efficiency of the task distribution can be derived by applying the definition of task scheduling efficiency in [102]. For a task distribution \( \alpha \), the efficiency is given by
\[
\Gamma = 1 + \frac{\sum_{k \in \{i, c, \{ij\} \in J\}} \max(D_i(\alpha_i), D_c(\alpha_c, \mu_c), D_j(\alpha_{ij}, \mu_{ij})) - D_k}{D_i(\alpha_i) + D_c(\alpha_c) + \sum_{j \in J} D_j(\alpha_{ij})} \geq 1.
\]
(2.18)

In other words, \( \Gamma \) is defined as one plus the ratio between the total idle time of the edge computing nodes and the total transmission and computing time. Therefore, \( \Gamma = 1 \) means that all nodes in the edge network can complete their assigned tasks with the same latency. Theorem 1 shows that the optimal latency is \( u^* = D_i(\lambda_i) = D_c(\lambda_c, \mu_c) = D_j(\lambda_{ij}, \mu_{ij}) \). Since \( u^* \) is the maximum value among \( D_i(\lambda_i), D_c(\lambda_c, \mu_c), D_j(\lambda_{ij}, \mu_{ij}) \), from (2.10), the efficiency of the optimal task distribution will be equal to one. Thus, if the efficiency of the task distribution becomes one, the latency of the task distribution is the optimal latency \( u^* \) according to Theorem 1.

### 2.4.4 Performance Analysis of the Proposed Online Optimization Framework

Next, we show that the proposed framework can achieve the target competitive ratio \( \gamma \).

**Theorem 2.** For a given \( \gamma \), the proposed framework satisfies \( \text{ALG}(\sigma)/\text{OPT}(\sigma) \leq \gamma \) if: (i) a given \( \gamma \) enables edge node \( i \) to select \( \hat{J} \) nodes, and (ii) the optimal task distribution can always be found, i.e., \( \Gamma = 1 \).

**Proof.** The offline optimal latency of the nodes in \( J \) is greater than or equal to \( \hat{u} \), i.e., \( \hat{u} \leq \text{OPT}(\sigma) \). Also, in Algorithm 2.1, the selected nodes satisfy \( D_j(\hat{\lambda}_{ij}, \mu_{ij}) \leq \hat{u}, \forall j \in J \) where \( |J| = \hat{J} \). When the task distribution is not yet optimized with respect to \( J \), the latency that results from using distribution \( \{\hat{\lambda}_i, \hat{\lambda}_c, \hat{\lambda}_{ij}\} \) can be shown as
\[
\text{ALG}_{b}(\sigma) = \max\left(D_i(\hat{\lambda}_i), D_c(\hat{\lambda}_c, \mu_c), D_j(\hat{\lambda}_{ij}, \mu_{ij})\right).
\]
and, by Theorem 1, \( \hat{u} = D_i(\hat{\lambda}_i) = D_c(\hat{\lambda}_c, \mu_c) = D_j(\hat{\lambda}_{ij}, \mu_{ij}) \). Since the service rates and computing speeds of selected node \( j \in J \) are less than or equal to those of the ideal node, i.e., \( \mu_{ij} \leq \bar{\mu}_{ij}, \mu_j \leq \bar{\mu}_j \), and \( 1/\omega_j \leq 1/\bar{\omega}_j \), we have \( \hat{u} \leq D_j(\hat{\lambda}_{ij}, \mu_{ij}) \). Therefore, we have \( \text{ALG}_b(\sigma) = \max\left(\hat{u}, D_j(\hat{\lambda}_{ij}, \mu_{ij})\right) = \max\left(D_j(\hat{\lambda}_{ij}, \mu_{ij})\right) \leq \gamma \hat{u}, \forall j \in J \). By optimizing the task distribution for the nodes in \( J \), the latency can be further reduced, i.e, \( \text{ALG}(\sigma) \leq \text{ALG}_b(\sigma) \). Hence, it is possible to conclude that \( \text{ALG}(\sigma) \leq \text{ALG}_b(\sigma) \leq \gamma \hat{u} \leq \gamma \text{OPT}(\sigma) \) and, therefore, \( \text{ALG}(\sigma)/\text{OPT}(\sigma) \leq \gamma \).

This result shows that the online optimization framework can achieve the target competitive ratio \( \gamma \) by determining a proper number of neighboring nodes \( \hat{J} \) and optimizing the task distribution. According to Theorem 2, the ratio between the latency achieved by executing one iteration of the proposed framework and an offline optimal latency can be bounded by the value of \( \gamma \).

To satisfy the first condition of Theorem 2, the proper value of \( \gamma \) needs to be found iteratively as shown in Fig. 2.3. Then, we prove that \( \gamma \) converges to an upper bound. For this proof, we define the lowest transmission service rate as \( \underline{\mu}_{ij} \) when the maximum of \( \overline{d}_{in} \) is \( d_{ij} \). Also, the lowest computing service rate and the lowest processing speed are defined as \( \overline{\mu}_j \) and \( 1/\overline{\omega}_j \), respectively.

**Theorem 3.** The target competitive ratio \( \gamma \) converges to \( D_j(\hat{\lambda}_{ij}, \underline{\mu}_{ij})/\hat{u} \) if: (i) a given \( \gamma \) enables edge node \( i \) to select \( \hat{J} \) nodes, and (ii) the optimal task distribution can always be found, i.e., \( \Gamma = 1 \).

**Proof.** We show that there exists an upper bound of \( \gamma \) denoted by \( \bar{\gamma} \). Therefore, for a given sequence \( \sigma \), we show that
\[
\frac{\text{ALG}(\sigma)}{\text{OPT}(\sigma)} \leq \frac{\max_{\sigma'} \text{ALG}(\sigma')}{\min_{\sigma'} \text{OPT}(\sigma')} = \bar{\gamma}, \text{ where } \sigma' \text{ denotes any sequence.}
\]
In the first phase of Algorithm 2.1, since \( \hat{u} \)
is calculated by assuming that all neighboring nodes are ideal nodes, the lower bound of the offline latency for any sequence is given by \( \min_{\sigma} \text{OPT}(\sigma) = \bar{u} \). Also, if \( J \) neighboring nodes are located at the farthest distance \( d_{ij} \), the lowest edge transmission service rate denoted as \( \mu_{ij} \) is derived. Then, the worst case is defined by assuming that the neighboring nodes have the lowest service rates and computing speed, i.e., \( \bar{\mu}_{ij}, \bar{\mu}_{j} \), and \( 1/\bar{\omega}_{j} \). Therefore, the latency in the worst case can be presented by \( \max_{\sigma} \text{ALG}(\sigma) = D_{j}(\bar{\lambda}_{ij}, \bar{\mu}_{ij}) \). Finally, \( \gamma \) always increases when it is updated, and, hence, \( \gamma \) converges to a competitive ratio given by \( \bar{\gamma} = \frac{D_{j}(\bar{\lambda}_{ij}, \bar{\mu}_{ij})}{\bar{u}} \).

Therefore, the proposed framework is able to find the target competitive ratio by iteratively updating \( \gamma \) when \( d_{ij}, \mu_{ij} \), and \( 1/\omega_{j} \) are not known to edge node \( i \). Thus, once \( \gamma \) is found through the iterative process, Algorithm 2.1 is used to select the neighboring nodes, and the tasks are offloaded to the neighboring nodes as stated in Theorem 1. As a result, the proposed framework yields the set of \( J \) selected neighboring nodes and the corresponding task distribution that can achieve the target competitive ratio as shown in Theorem 2.

The upper bound in Theorem 3 is the performance in the worst case if a given \( \gamma \) enables edge node \( i \) to select \( \hat{J} \) neighboring nodes, and the optimal task distribution can always be found, i.e., \( \Gamma = 1 \). However, if \( N \) is small, the first condition on the network size in Theorem 3 cannot be satisfied. Therefore, for small \( N \), we statistically find an upper bound of the competitive ratio if the statistical information about neighboring nodes is known. To this end, we derive a lower bound of the probability, with respect to \( \gamma \), that the initial edge node forms an edge network with \( \hat{J} \) neighboring nodes in an iteration including \( N \) observations. To derive a statistical result, we assume that the values of the communications and computing capabilities of neighboring nodes are random variables. For example, the distance, \( d_{in} \), between the initial node and a neighboring node is a random variable within a finite range \( [d_{ij}, \hat{d}_{ij}] \), and, therefore, the service rate \( \mu_{in} \) from (2.1) is a random variable in the range \( [\mu_{ij}, \bar{\mu}_{ij}] \). Also, a neighboring node’s computing service rate \( \mu_{n} \) and computing delay \( \omega_{n} \) can be modeled as random variables that lie in the finite ranges \( [\mu_{j}, \bar{\mu}_{j}] \) and \( [\omega_{j}, \bar{\omega}_{j}] \), respectively.

**Proposition 1.** The probability that the initial edge node forms an edge network with \( \hat{J} \) neighboring nodes in an iteration including \( N \) observations is at least

\[
p'(\gamma) = \sum_{k=J}^{N} \binom{N}{k} p_{s}^{k} (1 - p_{s})^{N-k}
\]

where \( p_{s} = F_{d_{in}} \left( \left( \frac{W_{i} N_{u} \bar{\lambda}_{ij}}{\hat{d}_{ij} N_{u} \bar{\lambda}_{ij}} \right) \left( \frac{2}{\bar{\mu}_{ij}} \bar{\mu}_{ij} + \bar{\lambda}_{ij} - \bar{\lambda}_{ij} \right) + \left( \frac{2}{\bar{\mu}_{ij}} \bar{\mu}_{ij} + \bar{\lambda}_{ij} - \bar{\lambda}_{ij} \right) \right) - 1 \right) \left( 1 - F_{\mu(n)} \left( \frac{1}{\bar{\mu}_{ij}} \left( \frac{2}{\bar{\mu}_{ij}} \bar{\mu}_{ij} + \bar{\lambda}_{ij} - \bar{\lambda}_{ij} \right) + \bar{\lambda}_{ij} \right) \right) F_{\bar{\omega}_{n}}(\gamma \bar{\omega}_{j}).

**Proof.** See Appendix 2.7.1.

By using the probability in Proposition 1, the first condition of Theorem 3 can be replaced with the condition that \( p'(\gamma) \) is very close to 1. This is due to the fact that, for a given \( \gamma \), an edge network is always formed with \( \hat{J} \) neighboring nodes if \( p'(\gamma) = 1 \). We define \( \bar{\gamma}_{s} \) as the smallest value of \( \gamma \) with which the initial edge node forms a network including \( \hat{J} \) neighboring nodes with probability \( p'(\gamma) = 1 \) in an iteration including \( N \) observations, i.e., \( \bar{\gamma}_{s} = \min\{ \gamma | p'(\gamma) = 1 \} \).

**Theorem 4.** The competitive ratio resulting from our proposed framework is \( \max (\bar{\gamma}, \bar{\gamma}_{s}) \) if the optimal task distribution can always be found, i.e., \( \Gamma = 1 \).

**Proof.** If \( \bar{\gamma}_{s} \leq \bar{\gamma} \), then \( \gamma \) is set to \( \bar{\gamma} \) in order to satisfy the first condition of Theorem 3. Under the assumption that the optimal task allocation, i.e., \( \Gamma = 1 \), is achieved, the result of Theorem 3 is directly applied, and, therefore, \( \bar{\gamma} \) becomes the competitive ratio. If \( \bar{\gamma}_{s} > \bar{\gamma} \), then we set \( \gamma = \bar{\gamma}_{s} \) so that the first condition of Theorem 3 is satisfied. Therefore, Theorem 3 states that \( \bar{\gamma} \) is the upper bound of the competitive ratio. Since \( \gamma = \bar{\gamma}_{s} \) is greater than \( \bar{\gamma} \) that is the upper bound found in Theorem 3, \( \bar{\gamma}_{s} \) becomes a new upper bound. Hence, the competitive ratio is \( \max (\bar{\gamma}, \bar{\gamma}_{s}) \).

The upper bound in Theorem 4 needs only one condition related to efficiency \( \Gamma = 1 \) whereas the upper bound in Theorem 3 requires two conditions on the number of neighboring nodes \( |J| = \hat{J} \) and efficiency \( \Gamma = 1 \).
Fig. 2.4 shows the upper bound $\bar{\gamma}$ derived in Theorem 3. Fig. 2.4 also shows the probability $p'(\gamma)$ derived in Proposition 1 with respect to the target competitive ratio $\gamma$ for different numbers of observations $N$. In Fig. 2.4, the neighboring nodes are randomly located on a circular area with the maximum distance $\bar{d}_{ij} = 50$ m. Also, $\mu_n$ and $\omega_n$ follow uniform distributions in the ranges $[15, 40]$ and $[0.05, 0.10]$, respectively. In Fig. 2.4, we use $\bar{J} = 6$, $\lambda_{ij} = 1.4$, and $l = 1$.

In Fig. 2.4, if the initial edge node sets $\gamma = \bar{\gamma}$, we can see that $p'(\gamma) = 1$ for a large value of $N$. For example, the probability $p'(\gamma)$ is one when $\gamma = 2.37$ and $N = 300$. In this case, since the first condition of Theorem 3 is satisfied, the competitive ratio becomes $\bar{\gamma}$ if the optimal task allocation is achieved. Also, Fig. 2.4 shows that there exists values for $N$ that do not satisfy $p'(\bar{\gamma}) = 1$. For instance, the probability $p'(\gamma)$ is not one when $\gamma = 2.37$ and $N = 40$. To satisfy $p'(\gamma) = 1$ for this $N$, the initial edge node needs to set $\gamma = \bar{\gamma}_s$, e.g., $\bar{\gamma}_s = 2.6$. Since $p'(\gamma)$ approaches to one with increasing $\gamma$, it is possible to determine $\bar{\gamma}_s$ by numerically finding the smallest $\gamma$ such that $p'(\gamma)$ is very close to 1. Then, in Fig. 2.4, we can observe that $p'(\bar{\gamma}_s)$ becomes one. Thus, from Theorem 4, the competitive ratio becomes $\bar{\gamma}_s$, if the optimal task allocation is achieved. Consequently, the results of Theorems 3 and 4 can be used to avoid any trial and error in the network formation stage by setting the initial value of $\gamma$ to the competitive ratio derived using these two theorems. If the conditions of Theorems 3 and 4 are satisfied, a network can be formed at once, and updating $\gamma$ is not required. To do so, the initial edge node however has to know the information assumed to derive the competitive ratio in Theorems 3 and 4. When the information is unknown, the proposed framework in Fig. 2.3 can be used to iteratively optimize the target competitive ratio.

### 2.5 Simulation Results and Analysis

For our simulations, we use a MATLAB simulator\footnote{For future research, this work can be extended to an experimental analysis pertaining to edge computing in which we can deploy the proposed system in an actual real-world environment.} in which we consider an initial edge node that can connect to neighboring edge nodes uniformly distributed within a circular area of radius 50 m. The arrival sequence of the edge nodes follows a uniform distribution. The task arrival rate at edge node $i$ is $x_i = 10$ packets per second. The computing
Table 2.2: Simulation parameters

<table>
<thead>
<tr>
<th>Notation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega_i = \omega_j, \omega_c$</td>
<td>50, 25 msec/packet</td>
</tr>
<tr>
<td>$\mu_i = \mu_j, \bar{\mu}_i = \bar{\mu}_j$</td>
<td>15, 40 packet/sec</td>
</tr>
<tr>
<td>$N, \tau$</td>
<td>300, 0.002 (0.005 in Fig. 2.7)</td>
</tr>
<tr>
<td>$P_{tx,i}, \beta_1, \beta_2$</td>
<td>20 dBm, $10^{-3}$, 4</td>
</tr>
<tr>
<td>$K$</td>
<td>64 kilobytes</td>
</tr>
<tr>
<td>$B, N_0$</td>
<td>3 MHz, $-174$ dBm/Hz</td>
</tr>
</tbody>
</table>

Figure 2.5: Latency for different task arrival rates at the initial edge node $i$.

The service rate of the edge nodes is randomly drawn from a uniform distribution over a range of 15 to 40 packets per second. All statistical results are averaged over a large number of simulation runs. Similar to prior work [80], the simulation results are evaluated with the parameters listed in Table 2.2.

### 2.5.1 Performance Evaluation of the Online Optimization Framework

Fig. 2.5 shows the latency when the total task arrival rate increases from 10 to 19 packets per second with $d_c = 100, 120,$ and 140 m, respectively. For comparison purposes, we use a baseline algorithm in which the algorithm observes the first 110 over 300 observations nodes and then selects the neighboring nodes from the rest of the arrivals by using the secretary algorithm in [80]. In Fig. 2.5, we show that the proposed framework can reduce the latency compared to the baseline, for all task arrival rates. For instance, the latency can be reduced by up to 19.25% compared to the baseline when $x_i = 19$ and $d_c = 140$ m. Also, from Fig. 2.5, we can see that the latency decreases as the distance to the cloud is reduced. With a shorter distance to the cloud, the cloud transmission service rate becomes higher. Therefore, the cloud is able to process more tasks with a low latency, and the overall latency of the edge network is improved. For example, at $x_i = 19$, if $d_c$ decreases from 140 m to 100 m, the latency is reduced by 4.29%. Moreover, we show that the latency decreases as less tasks arrive at the initial edge node $i$. For instance, when $x_i$ decreases from 19 to 10, the latency is reduced by about 25% with $d_c = 100$ m.
Fig. 2.6: Latency for different number of neighboring nodes.

Fig. 2.6 shows the relationship between the latency and the number of neighboring nodes when the total task arrival rate is given by $x_i = 10$ and 13 packets per second, respectively, and the processing delays of the edge nodes are given by $\omega_i = \omega_j = 50$ and 30 milliseconds, respectively. In Fig. 2.6, a smaller processing delay indicates that the edge nodes have a higher processing speed. From Fig. 2.6, we can observe the tradeoff between scenarios having a large number of edge nodes with low processing power and scenarios having a small number of edge nodes with high processing power. If edge nodes with higher processing speed are deployed, latency is reduced, and the formed network size decreases. This is due to the fact that the edge nodes having a faster processing speed do not need to form a large network. In fact, a larger network size can lead to lower transmission service rates. For instance, if the processing delay of edge nodes decreases from 50 to 30 milliseconds, the latency is reduced by up to 18.8% while the number of neighboring nodes decreases from 7 to 5.

Fig. 2.7 plots the value of $\gamma$ during 700 updates for different distances to the cloud, $d_c = 100$ m and 120 m, respectively. Fig. 2.7 shows that the value of $\gamma$ approaches a constant value. For instance, $\gamma$ first reaches 1.17 at 38 iterations with $d_c = 120$ m. Then, $\gamma$ becomes 1.21 at 329 iterations, and this value is maintained thereafter. From Fig. 2.7, we can see that edge node $i$ can find a proper $\gamma$ after a finite number of trials and updates. Also, the results of Fig. 2.7 show that $\gamma$ becomes larger as the distance to the cloud is closer. This is because $\hat{\alpha}$ and the threshold value decrease when $d_c$ is reduced. If the threshold value decreases, it becomes more challenging to select the $J$ neighboring nodes within the limited number of observations since the selected neighboring nodes must have a lower latency than the threshold. Therefore, in order to maintain a proper threshold value, $\gamma$ will be larger when $d_c$ decreases.

Fig. 2.8 shows the relationship between the edge transmission service rate and the number of neighboring nodes when $x_i = 10$ and 13, respectively. Here, we can see that the edge transmission service rate increases as the number of neighboring nodes decreases. This stems from the fact that the bandwidth per node increases as less edge nodes share the total bandwidth. For instance, the edge transmission service rate can increase by 15.6% if $\hat{J}$ goes from 6 to 4 with $x_i = 10$. Fig. 2.8 also shows that the formed network size becomes larger if $x_i$ increases. This is due to the fact that offloading tasks to a larger size of the network can reduce the tasks per node, and, hence, the maximum latency of the network will decrease. For instance, when $x_i = 10$, the range of $\hat{J}$ is between 4 and 6. However, if $x_i = 13$, $\hat{J}$ falls in the range between 5 and 7.

In Fig. 2.9, we show the task distribution among neighboring nodes, the cloud, and edge node $i$ for different num-
Figure 2.7: Changes in the target competitive ratio $\gamma$ over 700 updates.

Figure 2.8: Edge transmission service rate with respect to the number of neighboring nodes.
Figure 2.9: Task distribution with respect to the number of neighboring nodes.

bers of neighboring nodes when two bandwidth allocation approaches are used, respectively. It can be seen that the cloud-centric bandwidth allocation increases the tasks offloaded to the cloud when compared to the equal-bandwidth allocation. This is because the cloud transmission service rate increases, so offloading more tasks to the cloud can lower the latency. For instance, if the cloud-centric bandwidth allocation is used and \( \hat{J} = 4 \), the cloud is allocated 22.86% more tasks than in the case of equal bandwidth allocation. Also, in Fig. 2.9, we show that the optimal network size is different, depending on the bandwidth allocation scheme. For instance, the cloud-centric bandwidth allocation yields a larger network size than the equal bandwidth allocation. When the network size is large, the cloud can maintain a high transmission service rate by using the cloud-centric bandwidth allocation. Therefore, the high cloud transmission service rate enables to offload most tasks to the cloud with a low transmission latency. For example, Fig. 2.9 shows that the number of neighboring nodes is between 4 and 6 if equal bandwidth allocation is used. However, if the cloud-centric bandwidth allocation is used, the number of neighboring nodes varies from 4 to 9.

Moreover, Fig. 2.9 shows that the number of tasks offloaded to the cloud decreases when \( \hat{J} \) increases from 4 to 6 for both bandwidth allocation schemes. In this phase, the number of tasks offloaded to neighboring nodes will increase because offloading more tasks at the edge layer can reduce the latency at the cloud. However, if the number of neighboring nodes increases when using the cloud-centric bandwidth allocation, e.g., there are 7 or more neighboring nodes, the number of tasks offloaded to the neighboring nodes will decrease with the network size. This is due to the fact that the edge transmission service rates are smaller for larger networks which yields higher edge transmission latency. As a result, more tasks will be allocated to the cloud so as to utilize its fast computing resources.

2.5.2 Performance Evaluation of Algorithm 2.1 for a fixed \( \gamma \)

In Figs. 2.10 and 2.11, we evaluate the performance of Algorithm 2.1 when the proposed framework uses a fixed value of \( \gamma \) without constraint (2.14). While the target competitive ratio is used in the proposed framework to determine the threshold value and make a decision on node selection, the baseline algorithm has a different mechanism to determine threshold values. Therefore, the latency results of the baseline do not depend on the target competitive ratio. By using a predefined \( \gamma \), the update step of \( \gamma \) is not needed, which can be useful for scenarios in which the delay of this update can hinder the network latency. Fig. 2.10 shows the latency for the different preset values of \( \gamma \) ranging from 1.2 to 1.5 with \( d_c = 100 \) m and 120 m, respectively. From Fig. 2.10, we can see that the proposed framework achieves lower
latencies than the baseline, for all $\gamma$. For instance, the latency of the proposed framework can be reduced by up to 20.3% compared to that of the baseline if $\gamma = 1.2$ and $d_c = 100$ m. Also, Fig. 2.10 shows that the latency achieved by the proposed framework becomes smaller when $\gamma$ decreases. This stems from the fact that a low threshold value with small $\gamma$ allows the initial edge node to only select neighboring nodes having a high performance. For example, the latency can be reduced by up to 12.1% if $\gamma$ decreases from 1.5 to 1.2 with $d_c = 100$ m.

In Fig. 2.11, we show the number of observations of the neighboring node arrivals until $\hat{J}$ neighboring nodes are selected for different $\gamma$ with $d_c = 100$ m and 120 m, respectively. In this figure, we can see that a large value of $\gamma$ results in a small number of observations due to the associated increase in the threshold value. For instance, as $\gamma$ increases from 1.2 to 1.5, the number of observations can be reduced by about 96% with $d_c = 100$ m. Fig. 2.11 shows that a large value of $d_c$ lowers the number of observations since increasing $d_c$ results in a large $\hat{u}$ and threshold value. For example, the number of observations can be reduced by about 42% if $d_c$ increases from 100 m to 120 m with $\gamma = 1.2$. Moreover, from Figs. 2.10 and 2.11, we can characterize the tradeoff between the latency and the number of observations. In particular, a small $\gamma$ results in a lower latency, but requires a large number of observation.

Fig. 2.12 shows the percentage of tasks offloaded to the cloud and the scheduling efficiency of the task distribution when two bandwidth allocation schemes are used, respectively, with $\gamma = 1.2$ and $d_c = 100$ m. In Fig. 2.12 (a), the tasks offloaded to the cloud decreases as the number of edge nodes increases since the cloud transmission service rate decreases. Also, Fig. 2.12 (b) shows that, when equal bandwidth allocation is used for a large network size, the scheduling efficiency may not be optimal, i.e., $\Gamma > 1$ due to a large latency for the transmissions to the cloud. In this case, although the equal-bandwidth allocation still achieves $\Gamma$ that is close to 1, the cloud-centric bandwidth allocation can be used to enhance efficiency. This is because the cloud-centric bandwidth allocation increases the cloud transmission service rate by allocating more bandwidth. It can be seen for instance that the equal bandwidth allocation yields $\Gamma = 1.013$ in the case of 6 neighboring nodes, but the efficiency of the cloud-centric bandwidth allocation becomes $\Gamma = 1$.

### 2.5.3 Optimal Network Size in an Offline Setting

Fig. 2.13 shows the optimal latency for different network sizes when all neighboring nodes are located at $d_{ij}$ varying from 10 m to 40 m. In Fig. 2.13, it is assumed that complete information on the network is known and that the edge
Figure 2.11: The required number of observations for different values of $\gamma$.

Figure 2.12: Performance comparison of two bandwidth allocation schemes with respect to the number of neighboring nodes.
Figure 2.13: Latency for different number of neighboring edge nodes in an offline setting.

nodes have identical parameters, i.e., $\mu_i = \mu_j = 20$ when $d_c = 150$ m. In this offline setting, we study the impact of the network size on the latency by using an offline optimization solver to find the optimal latency for a given network. Fig. 2.13 shows that the optimal latency is directly affected by the number of neighboring nodes. When the network size increases, latency starts to decrease since fewer tasks can be offloaded to each neighboring node. However, if the network size increases, the latency will eventually increase since the bandwidth per node is smaller. For example, the optimal latency decreases when the number of neighboring nodes increases from 1 to 3 with $d_{ij} = 40$. However, once the number of neighboring nodes increases beyond 3, the latency starts to increase. Moreover, from Fig. 2.13, we can see that the optimal network size changes with the distances between edge nodes. For instance, for $d_{ij} = 40$ m, the latency can be minimized when there are 3 neighboring nodes in the edge network. However, if $d_{ij} = 10$ m, the latency is minimized when the number of neighboring nodes is 5. Therefore, if the edge transmission service rate is high (for shorter distances), increasing the number of neighboring nodes to 5 can reduce the latency. On the other hand, if the edge transmission service rate is low (due to poor wireless channel), having a smaller network size with 3 nodes is required to minimize the latency. Also, we note that the results in Fig. 2.13 show that there exists an optimal network size that can be found by running Phase 1 of Algorithm 2.1. Finally, Fig. 2.13 clearly shows that the latency is reduced by offloading the tasks to both the edge layer and the cloud, instead of relying solely on the cloud. For example, if the tasks are offloaded to the cloud, initial edge node, and 5 neighboring nodes located at $d_{ij} = 10$ m, the latency can be reduced by up to 43.9% compared to the case using the cloud only.

2.6 Summary

In this chapter, we have proposed a novel framework to jointly optimize the formation of edge networks and the distribution of computational tasks in a hybrid edge-cloud system. We have addressed the problem using an online optimization formulation whose goal is to minimize the maximum latency of the nodes in the edge network in presence of uncertainty about edge nodes’ arrivals. To solve the problem, we have proposed online optimization algorithms whose target competitive ratio is achieved by suitably selecting the neighboring nodes while effectively offloading the tasks to the neighboring edge nodes and the cloud. The theoretical analysis and simulation results have shown that the proposed framework achieves a low target competitive ratio while successfully minimizing the maximum latency in
edge computing. Extensive simulation results are used to showcase the performance benefits of the proposed approach. For future work, a dynamic bandwidth scheme can be designed to further reduce the latency. Also, packet prioritizing can be adopted at the initial edge node to meet different service-level latency requirements. Moreover, the proposed framework can be extended to the scenario in which multiple edge networks are formed by multiple initial edge nodes. Finally, one can extend this work to an experimental analysis pertaining to edge computing in which the proposed system can be deployed in an actual wireless testbed.

## 2.7 Appendix

### 2.7.1 Proof of Proposition 1

**Proof.** For a given $\gamma$, the arriving node $n$ is selected by the initial edge node if $D_n(\bar{\lambda}_{ij}, \mu_{in}) \leq \gamma u$. The probability of node selection event $E_s$ is $p_s = \Pr\{D_n(\bar{\lambda}_{ij}, \mu_{in}) \leq \gamma u\}$. With the same target competitive ratio $\gamma$, $E$ is defined as the event where $E_j$ happens more than $J$ times during $N$ trials within an iteration. Since event $E$ is a sufficient condition to form a network for a given $\gamma$, the probability to form a network is at least given by $p = \sum_{k=J}^{N} \left( \frac{N}{k} \right) p_s^k (1 - p_s)^{N-k}$ where $N$ is the maximum number of observations allowed within an iteration, and all inputs $\sigma_n, \forall n \in [1, N]$ are independent.

Since $p_s = \Pr\left\{ \frac{1}{\mu_{in} - \bar{\lambda}_{ij}} + \frac{1}{\mu_{n} - \bar{\lambda}_{ij}} + \frac{1}{\mu_{n}} + 2\omega_n \leq \gamma \left( \frac{1}{\mu_{in} - \bar{\lambda}_{ij}} + \frac{1}{\mu_{ij} - \bar{\lambda}_{ij}} + \frac{1}{\mu_{n}} + 2\omega_j \right) \right\}$, a lower bound of $p_s$ can be given by $p'_s = \Pr\{E_1 \cap E_1' \cap E_2 \cap E_2' \cap E_3\}$ where $E_1$ is the event where $\frac{1}{\mu_{in} - \bar{\lambda}_{ij}} - \frac{1}{\mu_{ij} - \bar{\lambda}_{ij}} \leq 0$, $E_1'$ is the event where $\frac{1}{\mu_{in} - \bar{\lambda}_{ij}} - \frac{1}{\mu_{ij} - \bar{\lambda}_{ij}} \leq 0$, $E_2$ is the event where $\frac{1}{\mu_{in} - \bar{\lambda}_{ij}} - \frac{1}{\mu_{ij} - \bar{\lambda}_{ij}} \leq 0$, $E_2'$ is the event where $\frac{1}{\mu_{in} - \bar{\lambda}_{ij}} - \frac{1}{\mu_{ij} - \bar{\lambda}_{ij}} \leq 0$, and $E_3$ is the event where $\omega_n - \gamma \omega_j \leq 0$. Then, $p'_s$ can be rewritten as $\Pr\{E_1'\} \Pr\{E_1\} \Pr\{E_2\} \Pr\{E_2'\} \Pr\{E_3\}$. Then, due to the relationship $\frac{1}{\mu_{in} - \bar{\lambda}_{ij}} \leq \frac{1}{\mu_{in} - \bar{\lambda}_{ij}} \leq \frac{1}{\mu_{in} - \bar{\lambda}_{ij}} \leq \frac{1}{\mu_{ij} - \bar{\lambda}_{ij}}$, if the condition for $E_1$ is satisfied, i.e., $\frac{1}{\mu_{in} - \bar{\lambda}_{ij}} \leq \gamma \frac{1}{\mu_{ij} - \bar{\lambda}_{ij}}$, then it is clear that the condition for $E_1'$ is also satisfied, i.e., $\frac{1}{\mu_{in} - \bar{\lambda}_{ij}} \leq \gamma \frac{1}{\mu_{ij} - \bar{\lambda}_{ij}}$. This, in turn, implies $\Pr\{E_1'\} = 1$. Similarly, if $E_2$ happens, then it always incurs $E_2'$, and thus, $\Pr\{E_2\} = 1$. In consequence, $p'_s$ can be simplified as $p'_s = \Pr\{E_1\} \Pr\{E_2\} \Pr\{E_3\}$. Note that $\Pr\{E_1\}$ can be expressed by using $d_j$ since $\mu_{in}$ is a function of $d_j$ in (2.1). When $F_{d_{in}}, F_{\mu_{in}}$, and $F_{\omega_n}$, respectively, are the cumulative probability functions with respect to $d_{in}$, $\mu_{in}$, and $\omega_n$, $\Pr\{E_1\}$, $\Pr\{E_2\}$, and $\Pr\{E_3\}$ are $\Pr\{E_1\} = F_{d_{in}} \left( \left( \frac{W_n \lambda_n \mu_{in}}{\mu_{ni} - \bar{\lambda}_{ij}} \right)^{\hat{\gamma} - 1} - 1 \right)^{-1/\beta_2}$, $\Pr\{E_2\} = 1 - F_{\mu_{in}} \left( \frac{1}{\gamma} (\bar{\lambda}_{ij} - \hat{\bar{\lambda}}_{ij}) + \hat{\bar{\lambda}}_{ij} \right)$, and $\Pr\{E_3\} = F_{\omega_n} (\gamma \omega_j)$.

Finally, it is clear that $p' = \sum_{k=J}^{N} \left( \frac{N}{k} \right) p_s^k (1 - p'_s)^{N-k} \leq p$ due to $p'_s \leq p_s$. Hence, $p'$ is a lower bound of the probability that a given target competitive ratio is used to form a network without an update. \(\square\)
Chapter 3

Ephemeral Edge Computing in the IoT

3.1 Background, Related Works, and Contributions

Next-generation wireless networks will bring in new IoT services that can potentially transform people’s daily lives [103]. Much of these emerging IoT and 5G services require low latency in terms of both communication and computing. To deliver low-latency IoT services, one can resort to edge computing [4, 104] techniques that can use radio and computing resources at a network’s edge. In particular, by using local computing resources, edge computing can significantly reduce the distance of data transmission, thus inducing smaller communication latency. To enable large-scale and distributed edge computing among heterogeneous devices, there is a need to enable edge devices to pool their computing resources by instantaneously forming a local edge network to process the computational tasks received from various user applications [4]. Clearly, if properly deployed, edge computing will bring forth key benefits for low-latency IoT services by ensuring that a local edge network is instantaneously deployed by edge devices. Therein, fundamental challenges include joint radio and computing resource management and application-oriented edge computing system and architecture design.

3.1.1 Related Works

Edge computing in general IoT environments

Edge computing enables a diverse set of IoT services ranging from real-time IoT applications running on user devices to safety applications operating on connected vehicles [51]. Recent prior works in [52–55, 85, 86, 105] studied deployment scenarios and resource allocation problems for standard edge computing in static or low-mobility networks. In particular, the work in [52] proposed an edge computing platform deployed in network infrastructure nodes such as base stations to provide contents to users while maintaining a required quality-of-service. Meanwhile, the authors in [53] studied the problem of joint computational task offloading and radio resource allocation in a wireless powered edge computing system by using deep learning. The work in [54] introduced a caching scheme so as to maximize fairness for an edge computing environment consisting of heterogeneous devices with different communication and computing resources. The authors in [55] proposed a Lyapunov optimization-based computation offloading algorithm to jointly control transmit power and CPU-clock speeds when edge computing devices are powered by energy harvesting techniques. The work in [85] studied a partial computational task offloading and radio allocation problems are jointly studied. Moreover, in [86], a joint strategy of computational offloading and content caching is proposed to maximize the utilization of each edge node’s radio and computing resources when the statistical information on the
content request is previously known. In [105], the authors used edge computing for enhancing virtual reality services.

**Edge computing with high mobility**

The works in [58–62, 66, 68, 70, 71, 87, 88] studied various problems related to edge computing in IoT networks that integrate highly mobile devices such as UAVs and connected vehicles. First, in [66, 68, 70, 71, 87, 88], the authors studied the use of UAVs for wireless and computing scenarios. For instance, the authors in [66] proposed a framework that jointly optimizes UAV placement and uplink power control so that UAVs can collect edge data from ground sensors. In [68], the authors employed UAVs as edge message ferries that collect information in wireless sensor networks and carry the data to the destination. In [70, 71, 87, 88], the authors proposed various use cases for deploying airborne edge computing using a UAV. In [70], the authors investigated a UAV-mounted cloudlet in which UAVs equipped with a computing processor offload and compute the tasks offloaded from ground devices. The work in [71] studied a UAV-enabled mobile edge computing system in which the users harvest the energy from the signal transmitted by the UAV in downlink, and the harvested energy is used to transmit in uplink. The work in [87] investigated a UAV-enabled edge computing system in which a UAV offloads computational tasks from users and decides whether to compute the tasks or transmit the tasks to a remote server. The authors in [88] studied the joint problem of user association and computational task allocation in a mobile edge computing system where UAVs act as edge computing devices.

Next, edge computing is investigated in various scenarios incorporating connected vehicles [58–62]. The authors in [58] developed a distributed reputation management system in which the edge computing resources are allocated in a way to optimize security. The work in [59] proposed a low-complexity computation offloading algorithm that minimizes the computing cost at connected vehicles. Also, the work in [60] proposed the use of edge computing techniques to process the computational tasks required in a blockchain system by using the local computing resources of vehicular nodes. The authors in [61] developed a smart contract deployed on an edge computing system to enable connected vehicles to store and share the data securely. In [62], the authors applied a software-defined networking concept to develop an edge computing architecture in which the control plane protocol is designed to cluster a set of neighboring vehicles and a centralized edge computing server is used to optimize the data transmission path.

**Limited time constraints within edge computing**

The aforementioned prior works [52–55, 58–62, 66, 68, 70, 71, 85–88] assume that edge computing operates during a relatively long time period and does not have a constraint on the total edge computing time period. However, in IoT scenarios such as UAV or vehicular systems, edge computing can be initiated and discontinued at any time due to the completion of running an application or the mobility of the edge nodes. To capture such use cases, we propose the concept of *ephemeral edge computing* in which edge computing occurs among IoT devices that have a stringent time constraints within which they can perform edge computing. Next, we first provide the real-world examples of ephemeral edge computing scenarios and, then, we outline our key contributions in this area.

**3.1.2 Ephemeral Edge Computing**

In real world applications, various edge devices can be used to form a local edge network spontaneously and process computational tasks of different applications. One common observation here is that the total time period is limited in real-world IoT examples. In particular, the running time of a local edge network can be limited due to mobility of edge devices. Also, when edge computing is initiated to operate an IoT user’s application, the usage time of the application can be finite. Therefore, we introduce a notion of *ephemeral edge computing* to capture cases in which edge computing occurs in a relatively short time period. As discussed next, this concept admits many real-world IoT
Figure 3.1: Ephemeral edge computing framework where the vehicles are edge devices that form a local edge computing network during a limited time period. The vehicles offload their computational tasks from sensors and process in a distributed way while moving on the same direction.

Intelligent transportation systems

As shown in Fig. 3.1, edge computing can be applied to an urban road environment in which a number of sensors monitor the status of the road traffic, vehicle flow, and pedestrian generating a large data volume [62]. For example, the generated sensory data from the road environment can be used to detect the current traffic status or to predict safety hazards. Moreover, the generated data can also be used to decide the signal light timing and schedule the vehicles at a merging ramp or intersection [58]. Therefore, processing the sensory data from a road environment is essential to optimize and control the various physical components of transportation systems. In a road environment, since the road sensors have a low computational capability, edge computing on the vehicles can be used to offload the sensory data from environment. Then, the data is processed to extract meaningful information such as traffic forecast and safety warnings [106, 107]. Once the data is processed by the vehicles’ on-board computers, the vehicles can transmit the processed information to adjacent road side unit that can then use the processed information to control traffic flows.

Therefore, intelligent transportation systems provide an important use case for ephemeral edge computing. In an urban environment such as the one shown in Fig. 3.1, a set of vehicles move from an intersection to the next intersection while maintaining a formation. When edge computing is implemented on the vehicles, it can only be maintained for a limited time period due to mobility. Those vehicles can cooperatively process the offloaded data within a limited time period that is the travel time between two intersections. Therefore, these vehicles will form an ephemeral edge computing network. In this case, the total time period dedicated to edge computing in a vehicular network will be affected by the vehicles’ speed and trajectory. In particular, the vehicles can share the information on the destination and trajectory to estimate the time period during which a set of vehicles moving the same direction. This is just one example of edge computing among many others in the context of transportation systems.

Smart factory

In emerging smart factory scenarios, also known as Industry 4.0 [30], sensors can detect malfunctions and send diagnostics signals to actuators in the factory. Therefore, factory systems must be optimized to manage the process of sensory data transmission, low-latency computation, and proactive decision making in order to quickly react to new situations [32]. Some key challenges for enabling the smart factory vision include effective in-network computing...
Figure 3.2: Ephemeral edge computing framework to offload computational tasks from sensors and allocate the offloaded tasks to neighboring edge-computing UAVs when the source edge-computing UAV is moving to the destination.

Figure 3.3: An IoT environment where the sensory IoT data is instantly requested and processed by an ephemeral edge computing network to operate real-time applications such as augmented reality and gaming.

and improvement of wireless connectivity to integrate physical and digital systems, i.e., networking and computation. 

Computing sensory data in a timely manner is essential to operate a physical factory system. To this end, the concept of ephemeral edge computing can be applied in cyber-physical smart factory systems where UAVs, robots, and drones are deployed and perform key functions such as data storage, computing, control, and transmission [108].

As shown in Fig. 3.2, we consider a smart factory in which sensors monitor the status of the manufacturing process and generate a large data volume. For example, the generated sensory data can be used as an input to machine learning algorithms, e.g., for classification, to predict any abnormality in the manufacturing process. Hence, a number of computational tasks must be processed in order to make a decision on how to control the physical systems of the factory based on the information extracted from the data. However, due to the low computational capability of the sensors, it is not possible to compute those tasks locally at the sensors. Also, sensors are not able to transmit data over a long distance, and, hence, a flexible relay is necessary [69]. For example, edge-enabled UAVs can be used in a smart factory to gather the tasks from the sensors, compute the tasks, and deliver the computed results to the destination, e.g., a central factory controller that can control the actuators. This is a meaningful use case of ephemeral edge computing in that the local edge network can be maintained until the UAVs arrive at the destination. Here, the total time period of ephemeral edge computing corresponds to the moving time from the source location to destination.
IoT sensor systems for end users

As shown in Fig. 3.3, consider an IoT environment in which the generated sensory data from the IoT devices is used to control and monitor the status of home appliances, to detect a user’s motion and voice [109], or to run gaming and augmented reality applications at a museum, sport events, and sightseeing places [51]. Those applications require processing and analysis of the real-time IoT data. In particular, augmented reality and gaming applications must process the data depending on the user’s location and orientation. In this case, the time duration within which a user’s device is at a stable location in space can be relatively short, and ephemeral edge computing is needed to process the IoT data in a limited time period.

As a result, the aforementioned examples in this section show that: a) Ephemeral edge computing admits a diverse set of IoT applications and b) in these applications, the time period dedicated to ephemeral edge computing can be limited depending on the various factors such as mobility and usage patterns of applications. When the total time period of ephemeral edge computing is limited, there is a need for new approaches to efficiently allocate the radio and computing resources to process a maximum number of computational tasks while considering the time-sensitive nature of the system.

3.1.3 Contributions

The main contribution of this chapter is a novel framework for distributed ephemeral edge computing that can be operated within a limited time period, as needed in the applications of Figs. 1 – 3. In particular, our framework allows tasks from sensors to be offloaded to a source edge node, which can subsequently allocate tasks to neighboring edge nodes for computation before the source node finishes edge computing. When the exact information on the offloaded tasks is unknown to the source node, it is challenging to decide which neighboring edge node has to compute which task. Therefore, we formulate an online optimization problem whose goal is to maximize the number of computed tasks when the total time period dedicated to ephemeral edge computing is constrained. To solve this problem without any prior information on the future task size, we propose a new online greedy algorithm that is used by the source edge node to make an on-the-fly decision for selecting one of the neighboring node upon the sequential arrival of the computational tasks while a prior information on the task size is unknown. Then, we analyze the performance of the proposed algorithm by using the notion of competitive ratio; defined as the ratio between the number of computed tasks achieved by the proposed algorithm and the optimal number of computed tasks that can be achieved by an offline algorithm. To this end, we apply the concept of primal-dual approach where the ratio between the dual problem and the original problem constitutes a competitive ratio. Therefore, we derive dual problem so as to analyze the worst-case performance of the proposed online algorithm. By doing so, the worst-case competitive ratio can be derived as a function of the task sizes and the communication and computing performance of the neighboring edge nodes. Simulation results show that the proposed online algorithm can maximize the number of computed tasks and achieve a performance that is near-optimal compared to an offline solution that has full information on tasks.

The rest of this chapter is organized as follows. In Section 3.2, we present the system model. Section 3.2.2 formulates the proposed online problem. Section 3.3 presents our proposed solution and performance analysis. Simulation results are analyzed in Section 3.4 while conclusions are drawn in Section 3.5.
Figure 3.4: Online edge computing framework to offload computational tasks and allocate the offloaded tasks to neighboring edge nodes in total edge computing period $t_{tot}$ within an ephemeral edge computing system.

3.2 System Model and Problem Formulation

3.2.1 System Model

We consider an ephemeral edge computing system in which sensors generate a set $\mathcal{I}$ of $I$ tasks that are offloaded to a given edge node that we refer to hereinafter as the source edge node. The source edge node can be seen as a node with mobility such as vehicles and UAVs. Also, the source edge node can be a static node. While the scenarios of ephemeral edge computing can be diverse, the role of the source edge node is to offload the computational task data from the sensors, allocate the computational tasks to neighboring edge nodes, and finally deliver the computed results to the destination. Hence, the source edge node is able to know the total computing time. When tasks reach the source edge node, they are labeled by their order of arrival. Thus, a task that arrives a time instant $i$ is denoted as task $i \in \mathcal{I}$.

Since the source edge node processes the tasks using a first-input-first-output policy, it will sequentially compute its tasks. The set $\mathcal{J}$ of $J$ edge nodes that are neighbors to the source. Each edge node $j \in \mathcal{J}$ is used to compute some allocated task $i$ from the source edge node. We also consider that the set of neighboring edge nodes $\mathcal{J}$ is initially selected by the source edge node. In this regard, the source edge node selects the neighboring edge nodes that are moving towards its same destination.

The source edge node allocates the computational tasks to other neighboring edge nodes. Such distributed computing can reduce the overall computational latency when multiple tasks are computed. Also, to prevent an excessive energy consumption at neighboring edge nodes, we assume that only one task is allocated to one edge node. Therefore, when neighboring edge node $j$ computes task $i$, the decision variable is set as $y_{ij} = 1$. The other edge nodes are not used to process the same task $i$, i.e., $y_{ij'} = 0, \forall j' \in \mathcal{J} \setminus \{j\}, \forall i \in \mathcal{I}$. Task allocation to neighboring edge node incurs a transmission latency. The data rate pertaining to the transmission of the data of task $i$ to neighboring edge node $j$ will be:

$$r_j = B \log_2 \left(1 + \frac{g_j P_t}{\sigma^2}\right),$$

(3.1)

where $P_t$ is the transmit power of the source edge node, $B$ is the bandwidth, $\sigma^2$ is the noise power, and $g_j$ is the channel gain between the source edge node and neighboring edge node $j$. Therefore, when the data size of task $i$ is $d_i$ bits, the transmission latency becomes $d_i / r_j$. Once task $i$ is received by neighboring edge node $j$, it will be processed with a computational latency $d_i / f_j$, where $f_j$ is the computational speed of edge node $j$.

In the proposed ephemeral edge computing system, the time period that the source edge node actively use edge computing is given by $t_{tot}$. To determine $t_{tot}$, key features of the edge nodes can be considered. From the aforementioned cases of ephemeral edge computing, the total time period $t_{tot}$ can be determined as the moving time period of a set of edge computing vehicles on the road or UAVs in a smart factory. For example, $t_{tot}$ can depend on the mobility that
is characterized by the speed and moving distance of the source edge node. \( t_{\text{tot}} \) could also depend on the different trajectories of the source edge node. In the IoT scenarios, the total time period can be given as the running time of an application or the time period where a smart device is staying near the other edge devices to deploy an edge computing network. For a given \( t_{\text{tot}} \), if a certain number of tasks is processed as shown in Fig. 3.4, then the tasks’ transmission and computation must be completed within \( t_{\text{tot}} \). As shown in Fig. 3.4, the tasks are sequentially offloaded from the source edge node to one of neighboring nodes. For instance, when the first task, \( i = 1 \), is being allocated, the latency including transmission and computation of task 1 will be:

\[
\sum_{j=1}^{J} d_1 \left( \frac{1}{r_j} + \frac{1}{f_j} \right) y_{1j} \leq t_{\text{tot}}.
\]

Subsequently, since tasks are sequentially transmitted to the neighboring edge nodes in the order of index \( i \), there will be \( i - 1 \) transmissions before task \( i \) is transmitted. Therefore, the latency needed to complete any task \( i, \forall i \in \mathcal{I} \setminus \{1\} \),

\[
\sum_{j=1}^{J} \sum_{j'=1}^{i-1} d_{ij'} \left( \frac{1}{r_{ij'}} \right) y_{i'j'} + \sum_{j=1}^{J} d_i \left( \frac{1}{r_j} + \frac{1}{f_j} \right) y_{ij} \leq t_{\text{tot}},
\]

where the first term is the sum of the transmission latency of \( i - 1 \) tasks, and the second term is the transmission and computation of task \( i \). Since tasks are allocated to and computed by neighboring edge nodes in the order of index \( i \), if task \( i \) is computed within period \( t_{\text{tot}} \), then \( i - 1 \) tasks will also be computed in the given period. Next, we formulate an online task allocation problem to study how tasks are distributed within an edge computing network.

### 3.2.2 Problem Formulation

Our goal is to allocate tasks to neighboring edge nodes in order to complete the maximum number of tasks during the period \( t_{\text{tot}} \) needed for the source edge node to reach its destination. To compute the tasks, the source edge node must allocate each task to a neighboring edge node that can yield low latency. In practice, when the computational tasks arrive dynamically to the source edge node, their different data sizes cannot be known in advance. As a result, the source edge node will be unable to know a priori the information on future tasks, and, therefore, optimizing the task distribution process under this uncertainty is very challenging. Under such uncertainty, selecting a neighboring edge node that computes a current task must also account for potential arrival of future tasks. When the future information is revealed sequentially, the arrival of information can be captured within an online optimization framework. In particular, by using online optimization techniques such as those in [94], it is possible to make an on-the-fly decision while the future information is given in an online manner. To cope with the uncertainty of the future task arrivals while considering the data rate and computing capabilities of given neighboring edge nodes, we will thus propose a rigorous online optimization framework that can handle the problem of task allocation under uncertainty.

First, we formulate the following online task allocation problem whose goal is to maximize the number of computed tasks when the total latency is limited by \( t_{\text{tot}} \):

\[
\text{(D)}: \max_{y} \sum_{i=1}^{I} \sum_{j=1}^{J} y_{ij}
\]

s.t. \( (3.2), (3.3), \)

\[
\sum_{i=1}^{I} y_{ij} \leq 1, \forall j \in \mathcal{J},
\]

\[
\sum_{j=1}^{J} y_{1j} \leq 1,
\]

(3.4)
where \( y \) is the vector of decision variables \( y_{ij}, \forall i \in I, \forall j \in J \). Hereinafter, this problem is called the dual problem. Constraints (3.2) and (3.3) show that task \( i \)'s transmission and computation latency must be smaller than \( t_{\text{tot}} \). (3.5) implies that each neighboring edge node can compute at most one task. In constraint (3.6), the first task is allocated to one of the neighboring edge nodes. Constraint (3.7) implies that task \( i \) can be allocated to a neighboring edge node if the task allocation of task \( i-1 \) is successful, i.e., \( \sum_{j=1}^{J} y_{i-1j} = 1 \). Otherwise, if \( \sum_{j=1}^{J} y_{i-1j} = 0 \), then, task \( i \) cannot be allocated to any edge node, and \( \sum_{j=1}^{J} y_{ij} = 0 \). Due to (3.6) and (3.7), we have \( \sum_{j=1}^{J} y_{ij} \leq 1, \forall i \in I \), and, thus, each task is allocated to only one of neighboring edge nodes.

Note that problem (D) is an online optimization problem and is challenging to solve using conventional offline approaches. This is because the value of \( d_i, \forall i \), is sequentially revealed. When the tasks that different sensors send to the source edge node have a random size, the arrival sequence of \( d_i \) is assumed to be unpredictable and unknown. At the moment when \( d_i \) is disclosed, the source edge node knows only the current and past tasks. However, the source edge node must make an instant and irrevocable online decision on which neighboring edge node will compute task \( i \). Under such uncertainty on \( d_i \), allocating tasks to existing neighboring edge nodes must also account for potential arrival of new tasks. In fact, even if a given task allocation can compute an existing task successfully, it may have a detrimental effect on the allocation of incoming tasks. In particular, if an edge node having a high data rate and high computational speed is already assigned to compute a previous task, it may not be possible to compute a future task having a large size. Therefore, it is challenging to optimize the task allocation between incoming tasks and neighboring edge nodes.

In an online setting, the ad-auction problem in [94] shows a generalized structure of an online linear programming problem and its algorithmic solution. We observe that the ad-auction problem and our problem have a key difference in the dependency of the constraints. In particular, the ad-auction problem includes the independent constraints about the maximum allocation size for each buyer that corresponds to the edge node in our problem. However, in our problem, the constraints about the maximum allocation size of edge nodes are dependent on each other. For instance, in (3.2) and (3.3), the sum of the transmission latency of the previous tasks and the processing latency of the current latency should be less than \( t_{\text{tot}} \). The total time period is a function of the task allocation decisions of all edge nodes while each edge node has an independent task allocation size. Therefore, if the given budget of total time period is previously spent to offload and compute previous tasks, the source node cannot offload a new task to a neighboring node that is still available to accept a task. Additionally, our problem assumes that the arriving tasks are sequentially allocated to the neighboring node. For instance, the current task cannot be allocated to any node, if the previous task is not allocated due to constraints (3.6) and (3.7). Due to the aforementioned differences, we need to develop a novel online task allocation strategy to solve problem (D).

### 3.3 Proposed Online Task Allocation Framework

Our goal is to determine the vector of decision variables \( y \) so that the maximum number of sequentially arriving tasks is successfully computed by our distributed ephemeral edge computing system. When task size \( d_i \) is unpredictable, the decision is not trivial since the current decision may affect the task allocation of future tasks, and all tasks cannot be computed due to the limited time resource \( t_{\text{tot}} \). In this case, making an on-the-fly online decision, can process a smaller number of tasks than that of offline decision in which the complete information on all tasks is initially known. Therefore, the gap between the results achieved by online and offline cases must be minimized. To this end, the notion of competitive ratio [94] from competitive analysis can be used to measure the performance of our online algorithm. It is an effective metric that compares the ratio between the the objective function’s value achieved by an online algorithm
and that of the offline optimal solution. In particular, a competitive ratio can be defined as a constant $\gamma$ such that

$$1 \leq \frac{D_{\text{OPT}}}{D_{\text{IP}}} \leq \gamma,$$

(3.8)

where $D_{\text{OPT}}$ denotes the offline optimal solution (OPT) of problem (D) in the form of integer programming (IP), i.e., the maximum number of computed tasks with the integer solution of $y_{ij}$. We will measure the performance of our proposed algorithm by observing the upper bound value defined by $\gamma$.

To find the upper bound of problem (D), we use the structure of the primal and dual approach [94]. To this end, the optimization variables $y_{ij}$ are relaxed to be linear, i.e., $y_{ij} \in [0, 1]$. By using the duality of linear programming, problem (D) can be rewritten as:

$$(P): \min_{x, z, u_i} \sum_{i=1}^{l} t_{ti}x_i + \sum_{j=1}^{J} z_j + u_1,$$

(3.9)

s.t. $\left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i x_i + \left(\frac{d_i}{r_j}\right) \sum_{i' = i+1}^{l} x_{i'} \geq 1, \forall i \in I \setminus \{I\}, \forall j \in J,$$

(3.10)

$$\left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_j x_j + z_j + u_j \geq 1, \forall j \in J,$$

(3.11)

$$x_i \geq 0, z_j \geq 0, u_i \geq 0,$$

(3.12)

where $x$ and $z$ are vectors with elements $x_i, \forall i \in I$, and $z_j, \forall j \in J$, respectively. This problem is called the primal problem. In problem (3.9), $x_i, x_i \geq 2$, $z_j$, $u_i$, and $u_i \geq 2$ are the dual variables associated, respectively, with constraints (3.2), (3.3), (3.5), (3.6), and (3.7) in problem (D).

The values of (3.4) and (3.9) are denoted by $P_{\text{LP}}$ and $D_{\text{IP}}$, respectively. With $P_{\text{LP}}$ and $D_{\text{IP}}$, a competitive ratio in (3.8) is derived. From the dual and primal problem formulation, it can be shown that $D_{\text{IP}} \leq P_{\text{LP}} \leq D_{\text{LP},\text{OPT}} \leq P_{\text{LP},\text{OPT}} \leq P_{\text{LP}}$. The first inequality is due to the fact that a linear relaxation allows problem (D), which is in the form of linear programming (LP), to have a higher value. The second inequality indicates that the offline optimal solution always achieves a value higher than or equal to the online solution of problem (D). The third inequality captures the slackness of the primal and dual problems. In the fourth inequality, the offline optimal solution of problem (P) is smaller than or equal to any online solution. Also, we have $D_{\text{IP}} \leq D_{\text{LP},\text{OPT}} \leq D_{\text{LP}}$. The first inequality follows from the optimality gap between the online and offline solutions when $y_{ij}$ is an integer. The second inequality shows that linear relaxation of $y_{ij}$ allows us to have a higher value in problem (D). Thus, the ratio in (3.8) becomes:

$$\frac{D_{\text{LP},\text{OPT}}}{D_{\text{IP}}} \leq \frac{P_{\text{LP}}}{D_{\text{IP}}},$$

(3.13)

where $P_{\text{LP}}/D_{\text{IP}}$ corresponds to $\gamma$ in (3.8). Therefore, $P_{\text{LP}}/D_{\text{IP}}$ becomes a competitive ratio.

### 3.3.1 Online Greedy Algorithm

To find the ratio $P_{\text{LP}}/D_{\text{IP}}$, we develop a new online greedy algorithm (Algorithm 3.2) specifically designed to solve problems (D) and (P), based on a general online optimization framework using the primal and dual approach of [94]. In Algorithm 3.2, the decision variables $y_{ij}$, $x_i$, $z_j$, and $u_i$ are updated while observing the new value of $d_i$. In particular, when task $i$ arrives to the source edge node, the original dual problem is solved by determining the value of $y_{ij}$. Also, other dual variables $x_i$, $z_j$, and $u_i$ are updated in order to find the performance bound of the proposed online algorithm. At the initial step of Algorithm 3.2, all variables are set to 0. The algorithm selects which edge node should compute task $i$. Since it is beneficial to offload task $i$ from the source node to the neighbor with a high data rate and computing speed, this decision rule can be designed to select an edge node with the shortest communication and
Online Task Allocation Algorithm

Algorithm 3.2 Online Task Allocation Algorithm

1: Initialize $y_{ij} = x_i = z_j = u_i = 0, \forall i, j.$
2: for $i \in \mathcal{I}$
3: Task $i$ arrives at source node.
4: Select edge node by using (3.14).
5: if (3.2) and (3.3) are satisfied, and $\sum_j y_{i-1j} = 1$,
6: $y_{ij} \leftarrow 1.$
7: Allocate task $i$ to edge node $j^*$ defined in (3.14).
8: Update $z_j, x_i,$ and $u_i$, respectively, by using (3.15), (3.16), and (3.17).
9: otherwise,
10: $y_{ij} \leftarrow 0.$
11: Set $\Delta u_i = 1$ and update $u_{i'}, \forall i' \leq i$
12: end if
13: end for

computing latency to process the task $i$, $(\frac{1}{r_j} + \frac{1}{f_j}) d_i$. To this end, edge node $j^*$ is selected by following the decision rule:

$$j^* = \arg \max_{\forall j} \frac{(1-z_j)^\alpha}{\frac{1}{r_j} + \frac{1}{f_j}} d_i,$$  \hspace{1cm} (3.14)

where $\alpha \geq 1$ is a constant used to guarantee that at least one of the tasks can be fairly allocated among the neighboring edge nodes. Since $z_j$ is initially zero, the decision rule in (3.14) only considers the latency required to process task $i$. In Algorithm 3.2, if a neighboring node $j$ accepts a task, the value of $z_j$ is updated to become positive. By doing so, $1-z_j$ is reduced, and, hence, another neighboring node can be selected when the next task arrives. However, if the neighboring node $j$ results in $(1-z_j)^\alpha / (\frac{1}{r_j} + \frac{1}{f_j}) d_i \geq (1-z_{j'})^\alpha / (\frac{1}{r_j} + \frac{1}{f_j}) d_i, \forall j \in \mathcal{J} \setminus \{j\}$, the same node $j$ can be selected again. This can violate constraint (3.5) that restricts each neighbor to accept one task. Therefore, a large value of $\alpha$ can be used to make $(1-z_j)^\alpha$ close to zero. Then, at the arrival of a new task, a different neighboring node is selected as $j^*$ by using decision rule (3.14).

After a neighboring node $j^*$ is selected for task $i$, if the time budget is still available for the current task $i$ from constraints (3.2) and (3.3), neighbor node $j^*$ finally receives task $i$ from the source node and performs processing. At this moment, the dual and primal variables are updated in Algorithm 3.2. The algorithm sets $y_{ij^*} = 1$ showing that task $i$ is allocated to edge node $j^*$. Next, the value of $z_{j^*}$ must be updated since $z_{j^*}$ is the primal variable associated with the dual problem’s constraint (3.5) with $j = j^*$. When a neighboring node initially does not have any accepted task, all $z_j, \forall j \in \mathcal{J}$ are set to zero. However, if a neighboring node $j$ accepts a task $i$, $z_j$ will be updated as follows:

$$z_j = z_j \left(1 + (\frac{1}{r_j} + \frac{1}{f_j}) \frac{d_i}{t_{tot}}\right) + \left(\frac{1}{r_j} + \frac{1}{f_j}\right) \frac{d_i}{t_{tot}} (\frac{1}{c-1}),$$  \hspace{1cm} (3.15)

where $c > 1$ is a positive constant that will be defined later. Also, the total time period $t_{tot}$ is assumed to be enough to process at least one task, and, thus, $(\frac{1}{r_j} + \frac{1}{f_j}) \frac{d_i}{t_{tot}} < 1$. Meanwhile, the update of $x_i$ must satisfy constraints (3.10) and (3.11). The value of $x_i$ is updated by using the rule:

$$x_i = \frac{(1-z_j)^\alpha}{\frac{1}{r_j} + \frac{1}{f_j}} d_i.$$  \hspace{1cm} (3.16)

Moreover, the values of $u_{i'}, \forall i' \leq i$, is updated as follows:

$$u_{i'} = u_{i'} + \Delta u_i, \forall i' \leq i,$$  \hspace{1cm} (3.17)
where we define, \( \forall j' \in \mathcal{J}, \)
\[
\Delta u_i \triangleq \max_{j' \in \mathcal{J} \setminus \{j^*\}} \left( 1 - \left( \frac{1}{r_{j'} + 1/f_{j'}} \right)^\alpha \left( \frac{1 - z_{j'}}{r_{j'} + 1/f_{j'}} \right) z_{j'} \right),
\]
(3.18)

Otherwise, if the edge nodes in \( \mathcal{J} \) do not satisfy (3.2) and (3.3), then, the tasks arriving after task \( i \) cannot be computed, i.e., \( y_{ij} = 0 \). In this case, to satisfy constraints (3.10) and (3.11), Algorithm 3.2 updates any \( z_{j} \) that has a value of 0 to 1 if \( J \leq I \), or, otherwise, \( \Delta u_i \) is set to 1. This update is intended to satisfy the constraints (3.10) and (3.11) for all \( i \in \mathcal{I} \) and \( j \in \mathcal{J} \). For the arrival of each task, the proposed algorithm is a one-shot decision making process to find a feasible solution. Therefore, by iterating the proposed algorithm for all arriving tasks during \( t_{tot} \), our algorithm converges to a feasible solution of problem \((D)\).

### 3.3.2 Performance Analysis

For the analysis hereinafter, we assume that \( \alpha = 1 \) for analytical tractability. In practice, this assumption implies that the decision rule (3.14) tends to select the neighboring node with a high data rate and computing speed. As \( \alpha \) increases, the decision rule selects a new neighboring node that has not been used to process any previous task. Now, as a first step to derive the competitive ratio of the proposed algorithm, we find the following result.

**Lemma 1.** The constraints of the primal problem (3.10) and (3.11) will be satisfied if \( z_j, x_i, \) and \( u_i \) are updated by (3.16), (3.15), and (3.17), respectively.

**Proof.** See Appendix 3.6.1.

The next step of our analysis is to check whether the constraints in problem \((D)\) is satisfied. In particular, since it is observable that the upper bound of the left-hand side of the constraint (3.5) can be greater than one, (3.5) is not satisfied for \( \alpha = 1 \), as shown next.

**Lemma 2.** In (3.5), \( \sum y_{ij} \) is violated by at least 2.

**Proof.** See Appendix 3.6.2.

This result implies that more than two tasks can be offloaded to the same neighboring node. However, there exists a condition under which constraint (3.5) is satisfied.

**Lemma 3.** (3.5) is satisfied if \( d_i > \left( \left( 1/r_{j_i^*} + 1/f_{j_i^*} \right)^{-1} - \left( 1/r_{j_i} + 1/f_{j_i} \right)^{-1} \right) t_{tot}(c - 1) \) where \( j_i^* \) is the node selected to process task \( i \), \( \forall i \in \mathcal{I} \).

**Proof.** After task \( i \) is offloaded to node \( j_i^* \), Algorithm 3.2 updates \( z_{j_i} = \left( 1/r_{j_{i}} + 1/f_{j_{i}} \right) \frac{d_i}{t_{tot}(c - 1)} \). Next, when task \( i+1 \) arrives, the condition above yields the inequality \( \frac{1}{r_{j_{i+1}} + 1/f_{j_{i+1}}} \frac{d_i}{t_{tot}(c - 1)} > \frac{1}{r_{j_{i}} + 1/f_{j_{i}}} \frac{d_i}{t_{tot}(c - 1)} \), \forall i \in \mathcal{I} \) with \( \alpha = 1 \). Therefore, (3.14) is used to select a new node \( j_{i+1}^* \) to process task \( i + 1 \). Hence, a different neighboring node is selected for each task.

For instance, the condition in Lemma 3 can be satisfied if the value of \( d_i \) is decreasing over time. In that case, every neighboring node can be used to process different tasks, thus satisfying constraint (3.5). As a last step, we derive the increment rate of the \( \Delta P/\Delta D \) when a new task \( i \) arrives in an online manner.
Lemma 4. When the dual problem's objective function increases by one, the primal problem's objective function increases by 
\[\frac{\ln c}{(1/r_j + 1/f_j) \mu_c} \left( 1 + \frac{1}{c-1} \right) + \Delta u_i \text{ for any given } c > 1.\]

Proof. See Appendix 3.6.3. \(\square\)

Now, to derive a competitive ratio for the proposed algorithm, we will adopt a primal-dual online analysis analogous to the one done in [94]. In Lemma 1, it is shown that the primal variable is updated while satisfying the constraints (3.10) and (3.11). Then, we show that the dual constraints from (3.2) to (3.7) are satisfied under the derived condition in Lemma 3. Finally, the increment rates of the primal and dual problems are, respectively, derived in Lemma 4. As a result, from Lemmas 1, 3, and 4, we obtain the following key result:

Theorem 5. The competitive ratio of Algorithm 3.2 is \(O(1/\min_i \beta_{ij})\) where \(\beta_{ij} \triangleq \left( \frac{1}{r_j} + \frac{1}{f_j} \right) \frac{d}{\mu_c}\) if \(d_i > \left( (1/r_j + 1/f_j)^{-1} - (1/r_j' + 1/f_j')^{-1} \right) t_{tot}(c-1)\).

Proof. Lemma 1 first shows that the constraints of problem (P) are satisfied for all tasks that are assigned to the set of edge nodes. At each iteration, Lemma 3 shows that the increment of \(\Delta P/\Delta D\) is at most
\[\Delta P \leq \frac{1}{\min_i \beta_{ij}} \left( 1 + \frac{1}{(1+\delta)^{D/\mu_c} - 1} \right) + \max_i \Delta u_i, \tag{3.19}\]
where \(\beta_{ij} = \left( \frac{1}{r_j} + \frac{1}{f_j} \right) \frac{d_i}{\mu_c}\). Also, (3.19) has an upper bound at \(\delta = 1\). Since \(D_{IP} = \sum_{i,j} y_{ij}\), the future tasks \(i > D_{IP}\) cannot be allocated to any neighbor. In that case, Algorithm 3.2 sets \(\Delta u_i = 1\). Then, all values of \(u_i', \forall i' \leq i\) increase by one, resulting in \(\Delta D = 0\) and \(\Delta P = 1\). Thus, we have \(\gamma \leq \frac{\Delta P}{\Delta D} + (I - D_{IP})\). We observe that \(\Delta P/\Delta D\) increases with the rate of \(O(1/\min_i \beta_{ij})\) as \(\beta_{ij} \to 0\). At the same time, \(I - D_{IP}\) can decrease with \(D_{IP}\) when the number of processed tasks increases. Hence, the ratio \(\gamma\) can be bounded by \(O(1/\min_i \beta_{ij})\). \(\square\)

This result characterizes the online performance bound achieved by Algorithm 3.2 in which the competitive ratio can decrease as \(\min_i \beta_{ij}\) approaches 1. If \(\min \left( \frac{1}{r_j} + \frac{1}{f_j} \right) \frac{d_i}{\mu_c} \approx 1\), we have an environment in which all neighboring edge nodes have similar communication and computing performance, thus resulting in the smallest competitive ratio close to 1. In such a case, the online and offline performance gap is minimized. Also, Algorithm 3.2 can be usefully converted into another simple algorithm that updates \(\Delta u_i = 0\) for all tasks \(i \in I\) so that problem (P) has a value of \(I\), by assuming \(\left( \frac{1}{r_j} + \frac{1}{f_j} \right) \frac{d_i}{\mu_c}, \forall i, j\), equals to 1. This algorithm shows that the competitive ratio is inherently upper bounded by \(P_{LP}/D_{IP} = I\) in the worst case.

As shown in Theorem 5, it is essential to investigate how the value of \(1/\beta_{ij}\) is determined when measuring a realistic performance of the proposed ephemeral edge computing system. We conduct a statistical analysis to derive the probability corresponding to different values of \(1/\beta_{ij}\). To this end, it is assumed that the data rate and task size are randomly determined. In particular, the size of data \(d_i\) is generated by following a uniform distribution random variable \(D \sim U(0, D_{max})\) where \(D_{max}\) is the maximum size of a task. We assume that the data rate is denoted by a random variable \(R \triangleq \log_2 (1 + P)\) where \(P\) is the received power in a fading channel modeled as an exponential distribution with parameter \(\lambda\), i.e., \(P \sim \exp(\lambda)\). This statistical model is a simplified version of our edge computing system model. This statistical modeling facilitates the observation of factors that affect the performance of the proposed algorithm. Then, we derive the probability to have a certain value of \(1/\beta_{ij}\).

Theorem 6. If \(k \geq \frac{t_{tot} f}{D_{max}}\), the probability that \(1/\beta_{ij} \leq k\) is \((F_K(k) - F_K(1))/(1 - F_K(1))\) where
\[F_K(k) = \frac{1}{D_{max}} \left[ \int_0^{t_{tot}} \left( 1 - \exp \left( -\lambda \left( \frac{1}{2 \pi \sigma^2} \right)^{\frac{1}{2}} - 1 \right) \right) \right] dx + \left( D_{max} - \frac{t_{tot} f}{k} \right). \tag{3.20}\]
**Figure 3.5:** Example of the cumulative probability distribution of $F_{K|H}(k)$.

**Proof.** We define a random variable $K \triangleq \frac{t_{tot}}{\log_2(1+P) + \frac{1}{f}}$. Therefore, if $k \geq \frac{t_{tot}}{D_{\text{max}}}$, the cumulative density function of a random variable $K$ is shown as:

$$F_K(k) = \Pr\left(\frac{t_{tot}}{\log_2(1+P) + \frac{1}{f}} > k\right)$$

(3.21)

$$= \int_0^{D_{\text{max}}} \Pr\left(\frac{t_{tot}}{\log_2(1+P) + \frac{1}{f}} > k \mid D = x\right) \Pr(D = x) dx$$

(3.22)

$$= \frac{1}{D_{\text{max}}} \left[ \int_0^{\frac{t_{tot}}{D_{\text{max}}}} \left( 1 - \exp\left( -\lambda \left( \frac{2^{\frac{1}{f}} - 1}{f} \right) \right) \right) dx + \left( D_{\text{max}} - \frac{t_{tot}}{f} \right) \right].$$

(3.23)

When $H$ is defined as the event in which $K \geq 1$, the cumulative density function of a random variable $K$ conditioned on $H$ is $F_{K|H}(k) = \frac{\Pr(K < k | K \geq 1)}{\Pr(K \geq 1)} = (F_K(k) - F_K(1))/(1 - F_K(1))$.

When the tasks are randomly generated and wireless performance dynamically changes, Fig. 3.5 shows an example of the cumulative probability distribution of $F_{K|H}(k)$ when $t_{tot} = 2$, $1/f = 0.5$, and $D_{\text{max}} = 4$. In Fig. 3.5, if $k = 2$, the probability that $k = 1/\beta$ is less than 2 is around 50%. Therefore, the probability that $k$ becomes the empirical value of a competitive ratio in Theorem 6 is: $\Pr(1/\min_i \beta_{ij} \leq k) = \Pr(\max_i 1/\beta_{ij} \leq k) = (F_{K|H}(k))^f$. Also, from Theorem 6, the derived probability does not change if the total time period is equal to the processing time of the maximum task size, i.e., $t_{tot} = D_{\text{max}}/f$. Hence, if an ephemeral edge computing system is designed to use the maximum task size given by $t_{tot} f$, it is possible to expect the empirical value of the competitive ratio when the data rate and task size are randomly determined in a wireless environment.
3.4 Simulation Results and Analysis

For our simulations, we use a MATLAB simulator in which we consider that the source edge node initially forms a network with \( J = 10 \) neighboring edge nodes uniformly distributed within a circular area of radius between 10 m and 100 m. For instance, this can be seen as a generalized scenario in which an edge-enabled UAV (or vehicle) forms an edge network with \( J \) neighboring nodes in a smart factory (or on a road environment). The task size follows a uniform distribution between 50 and 100 Mbits, and the number of tasks is \( I = 10 \). The power spectral density of the noise is -174 dBm/Hz, the carrier frequency is 2.1 GHz, and \( P_t = 20 \) dBm. The computational speed of each neighboring edge node is randomly determined from a uniform distribution between \( 1 \times 10^8 \) and \( 5 \times 10^8 \) bits/sec. The offline optimal solution is calculated by using a mixed-integer linear programming (MILP) solver with the assumption that the size \( d_i \) of task \( i \), \( \forall i \in I \), is completely known. All statistical results are averaged over a large number of independent simulation runs.

Fig. 3.6 first shows the empirical ratio between the offline optimal and online solutions, \( D_{IP,OPT}/D_{IP} \) for the different values of \( \min_i \beta_{ij} \) when \( t_{tot} = 1 \), \( \alpha = 1 \), and \( f_j \in [7 \times 10^7, 10 \times 10^7] \). The numerical results in Fig. 3.6 confirm that the ratio \( D_{IP,OPT}/D_{IP} \) decreases as \( \min_i \beta_{ij} \) increases as shown in Theorem 5. For example, the empirical competitive ratio can be reduced up to 19.2\% if the smallest \( \beta_{ij} \) increases from 0.58 to 0.85. Also, in Fig. 3.6, the cases in which the ratio is one correspond to scenarios in which the proposed algorithm finds the optimal solution. For instance, when \( \min_i \beta_{ij} \) is greater than 0.79, Fig. 3.6 shows that the empirical ratio becomes one since \( D_{IP,OPT} = D_{IP} \).

Fig. 3.7 shows the average number of accepted tasks per node, i.e., \( \sum_i y_{ij} \), for two values of \( \alpha = 1 \) and 100. In Fig. 3.7, the number of accepted tasks per node needs to be one due to constraint (3.7). When \( 0 < z_j < 1 \), the selection rule in (3.14) can decide to offload a new task to a neighboring node that already accepted a task. In particular, Fig. 3.7 shows that the average number of accepted tasks per node increases with \( t_{tot} \) for \( \alpha = 1 \). This is due to the fact that the selection rule in (3.14) is affected by two factors, i.e., \( (1 - z_j)^\alpha \) and \( 1/ (1/r_j + 1/f_j) \frac{d_i}{t_{tot}} \) where \( (1 - z_j)^\alpha \) prevents the algorithm from choosing the same node multiple times. It is observable that \( 1/ (1/r_j + 1/f_j) \frac{d_i}{t_{tot}} \) increases as \( t_{tot} \) increases. Therefore, with a large \( t_{tot} \), the selection rule in (3.14) is determined by \( 1/ (1/r_j + 1/f_j) \frac{d_i}{t_{tot}} \), rather
than \((1 - z_j)^\alpha\). For example, Fig. 3.7 shows the average number of accepted tasks can reach up to 2 when \(t_{\text{tot}}\) increases from 1 to 3. Thus, to avoid offloading more than one task to the same neighboring node, a large \(\alpha\) is used in Fig. 3.7. If \(\alpha\) is set to a large value, e.g., 100, Fig. 3.7 shows that the selection rule in (3.14) only offloads the tasks to different nodes. This is due to the fact that \((1 - z_j)^\alpha\) is close to zero for a large \(\alpha\) when \(0 < z_j < 1\). For instance, when \(\alpha = 100\), the average number of accepted tasks is 1 for all \(t_{\text{tot}}\). To evaluate Algorithm 3.2 in a general task arrival, \(\alpha = 100\) is used for the rest of our simulations.

Fig. 3.8 shows the percentage of computed tasks for different values of \(t_{\text{tot}}\) from 0 to 7 seconds when the total bandwidth is 10 MHz. For comparison, we calculate the offline optimal solution of the dual integer problem, i.e., \(D_{\text{IP,OPT}}\), by assuming that all task sizes, \(d_i\), \(\forall i\), are known in advance. The offline optimal \(D_{\text{IP,OPT}}\) shows that the percentage of computed tasks increases with \(t_{\text{tot}}\) that is a given parameter in problem (D). The design goal of our online algorithm is to achieve a performance that is similar to the offline optimal when the task size \(d_i\) is revealed one by one. To this end, in Fig. 3.8, we can observe that the optimal solution and the solution found by Algorithm 3.2 are very close for all values of \(t_{\text{tot}}\). This demonstrates the effectiveness of the proposed algorithm that can select properly neighboring edge nodes to offload tasks while maximizing the number of computed tasks. For instance, Fig. 3.8 shows that the maximum gap between the offline optimality and the online solution is only 7.1\% when \(t_{\text{tot}} = 4\). Also, in Fig. 3.8, as \(t_{\text{tot}}\) increases, more tasks can be readily processed within a given time period, and, therefore, the percentage of computed tasks approaches to 100\%. In particular, when \(t_{\text{tot}} = 7\), Fig. 3.8 shows that all computational tasks are processed on the edge computing network in both online and offline cases, respectively.

Fig. 3.9 shows the cumulative frequency of the empirical ratio, \(D_{\text{IP,OPT}}/D_p\), for both the offline optimal and online solutions and the analytical competitive ratio, \(P_{\text{LP}}/D_p\) in (3.13), respectively, when \(t_{\text{tot}} = 1\). In Fig. 3.9, the ratio \(D_{\text{IP,OPT}}/D_p\) is shown to have a step-like shape since both \(D_{\text{IP,OPT}}\) and \(D_p\) are integers, and there exists a limited number of possible values for \(D_{\text{IP,OPT}}/D_p\) for specific settings of the simulations. In Fig. 3.9, the cases in which the ratio is one correspond to scenarios in which the proposed algorithm finds the optimal solution. For example, in Fig. 3.9, about 63.2\% iterations result in the slope of 1 where the optimal solution is achieved by running the proposed algorithm. By the definition of the competitive ratio \(\gamma\) in (3.8), the number of computed tasks with the proposed algorithm is at least \(D_{\text{IP,OPT}}/\gamma\). For instance, in Fig. 3.9, the largest empirical competitive ratio is shown to be 2 which implies that the number of computed task is at least \(D_{\text{IP,OPT}}/2\) when the proposed algorithm is executed with the given simulation parameters. Moreover, Fig. 3.9 shows that the analytical competitive ratio \(P_{\text{LP}}/D_p\) provides an
Figure 3.8: Comparison between the proposed algorithm’s result and the offline optimal solution in terms of percentage of computed tasks for different $t_{tot}$.

Figure 3.9: The empirical competitive ratio $D_{IP,OPT}/D_{IP}$ in (3.8) and the analytical competitive ratio $P_{LP}/D_{IP}$ in (3.13).
Figure 3.10: Percentage of computed tasks for different computational speeds of neighboring edge nodes and different task sizes when bandwidth is varying between 3 and 7 MHz.

upper bound on the empirical competitive ratio $D_{IP,OPT}/D_{IP}$, and the resulting value of $P_{LP}/D_{IP}$ is at most 7.

Fig. 3.10 shows the percentage of computed tasks for two different ranges of computational speeds of the edge nodes and different task sizes when the bandwidth is changed from 3 to 10 MHz with $t_{syn} = 7$ and distance randomly distributed in range from 10 m to 70 m. In Fig. 3.10, neighboring edge nodes with low computational speeds are represented by $f_j \in [5 \times 10^7, 8 \times 10^7]$, whereas edge nodes with high computational speeds are assumed to have $f_j \in [5 \times 10^8, 8 \times 10^8]$. Also, we consider two scenarios with small-size tasks $d_i \in [50 \times 10^6, 70 \times 10^6]$ and large-size tasks $d_i \in [70 \times 10^6, 90 \times 10^6]$, respectively. From Fig. 3.10, we can see that the number of computed tasks increases with more bandwidth. This is due to the fact that a higher bandwidth can increase the data rate and reduces tasks’ transmission latency. Therefore, more tasks can be allocated to neighboring edge nodes. For instance, the number of computed tasks can increase about two-fold if the bandwidth changes from 3 MHz to 10 MHz in the case of edge nodes with low computational speeds and large-size tasks. Also, Fig. 3.10 shows that using edge nodes with high computational speeds increases the number of computed tasks. For example, the percentage of computed tasks increases from 88% to 99.5% by using edge nodes having high computational speeds when bandwidth is 5 MHz and the task sizes are small. Moreover, Fig. 3.10 shows that more tasks can be computed as task sizes become smaller; for example, small-sized tasks result in 32.8% more computed tasks compared to that of large-sized tasks in the case of 4 MHz in a high computational speed case.

In Fig. 3.11, the percentage of computed tasks is shown for different numbers of neighboring edge nodes ranging from 10 to 60. The scenario in Fig. 3.11 assumes that neighboring edge nodes are randomly distributed with a maximum distance that is varied in range from 30 m to 110 m with $I = 10$, $B = 5$ MHz, and $t_{syn} = 7$. Simulations assume that the small-size tasks are in the range of $d_i \in [40 \times 10^6, 70 \times 10^6]$. Also, the neighboring nodes use low computational speeds in the range of $f_j \in [5 \times 10^7, 8 \times 10^7]$. In Fig. 3.11, it is clear that the number of computed tasks increases with the number of neighboring edge nodes. As the set of neighboring edge nodes becomes larger, the source edge node has a higher probability to allocate its tasks to the neighboring edge nodes having a high data rate and computational speed. For instance, the number of computed tasks can increase by about 8.2% if the number of edge nodes increases from 10 to 60 when the maximum distance is 110 m. Fig. 3.11 also shows that the number of computed tasks increases if the maximum communication distance between edge nodes is reduced. For example, the percentage of computed tasks increases from 91.8% to 99.6% by reducing the maximum distance between neighboring edge nodes and the
Figure 3.11: Percentage of computed tasks for different number of neighboring edge nodes and different maximum communication distances.

source edge node.

In Fig. 3.12, the percentage of computed tasks is shown for different transmit powers from 20 dBm to 25 dBm when the neighboring nodes use identical computing speed that varies from $10^8$ to $7.5 \times 10^8$. Fig. 3.12 shows that the number of computed tasks increases with the transmit power of the source edge node. This is due to the fact that the increased data rate reduces the wireless transmission latency, and, therefore, more tasks can be processed within a limited time period. For example, the percentage of computed tasks increases by up to 10.7% if the transmit power changes from 20 dBm to 25 dBm with $f_j = 10^8$. Also, Fig 3.12 shows that increasing a computing speed is beneficial to process notably more tasks. For instance, if the computing speed of edge nodes increases from $10^8$ to $7.5 \times 10^8$, the edge computing network can process up to about 20% more tasks. Thus, Fig 3.12 shows that reducing the computing latency by using a high computing speed is needed while reducing the transmission latency with a high transmit power.

3.5 Conclusion

In this chapter, we have proposed a new concept of ephemeral edge computing in which the total time period dedicated to edge computing is limited. This concept of ephemeral edge computing is applicable to a wide range of scenarios including Industry 4.0 smart factory, intelligent transportation systems, and smart homes. By modeling a generalized scenario of ephemeral edge computing, we have proposed a novel framework to maximize the number of successful computations over an edge computing network within a limited time period. This framework allows a source edge node to offload tasks from sensors and distributed tasks to neighboring edge nodes in order to compute the tasks before the source edge node discontinues its current edge computing network. When the exact information on the offloaded tasks is unknown to the source edge node, it is challenging to optimize the decision of which neighboring edge node has to compute each task. Therefore, we have formulated an online optimization problem that jointly optimizes the communication and computation latency is formulated and introduced an online greedy algorithm to solve the problem. Then, by using the structure of the primal-dual problem formulation, we have derived a feasible competitive ratio as a function of the task sizes and the data rates of the edge nodes. Simulation results have shown that
the empirical competitive ratio defined as the ratio between the number of computed tasks achieved by the proposed online algorithm and offline optimal case is at most 2 in a given simulation setting. Thus, the simulation results confirm that the proposed online algorithm can efficiently allocate tasks to neighboring edge nodes under uncertainty.

3.6 Appendix

3.6.1 Proof of Lemma 1

Proof. We will show that the first constraint is always satisfied for all $i$ when using the updating rule. When allocating task $i'$, $x_i = 0$, $\forall i \geq i'$ and $u_i = 0$, $\forall i$ due to the initialization. From the constraints in (3.10) and (3.11), we have that

$$\left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i x_i + \left(\frac{d_i}{r_j}\right) \sum_{i'=i+1}^{I} x_{i'} + z_j + u_i - u_{i+1} = \left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i (1 - z_j) \left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i = 1. \quad (3.24)$$

Then, we consider the constraints regarding other edge nodes $j \in J \setminus \{j^*\}$ for a given task $\forall i \in I$. When $u_i$ is updated, $u_i - u_{i+1}$ is equal to $\Delta u_i$. Therefore, we can show that edge node $\forall j \in J$ satisfy the constraint (3.6)

$$\left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i x_i + \left(\frac{d_i}{r_j}\right) \sum_{i'=i+1}^{I} x_{i'} + z_j + u_i - u_{i+1} = \left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i (1 - z_j)^\alpha \left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i = 1 + z_j + \Delta u_i \quad (3.25)$$

$$= \left(\frac{1}{r_j} + \frac{1}{f_j}\right) (1 - z_j^*)^\alpha + z_j + \max_{j' \in J} \left(1 - \left(\frac{1}{r_j} + \frac{1}{f_j}\right) \left(\frac{1}{r_j} + \frac{1}{f_j}\right) (1 - z_j^*)^\alpha + z_j^*\right), 0 \quad (3.26)$$

Figure 3.12: Percentage of computed tasks for different transmit powers with respect to different computing speeds of neighboring nodes.
Therefore, the primal constraints (3.10) and (3.11) are satisfied.

### 3.6.2 Proof of Lemma 2

**Proof.** For a given \( j \), the upper bound of \( \sum_{i} y_{ij} \) in (3.5) is derived by using the fact that the proposed algorithm does not update \( z_{j} \) if \( \sum_{i} y_{ij} \geq 1 \). In particular, when the task is indexed by \( i' \), suppose that the task allocation is not possible for the first time, i.e., \( y_{ij} = 0, \forall i > i' \). Before the last task \( i' \) arrives, the value of \( \sum_{i} y_{ij} \) is still less than the total budget of edge node \( j \). However, after allocating task \( i' \) to edge node \( j \), \( \sum_{i} y_{ij} \) can be greater than one.

The violation of the constraint (3.5) makes the value of \( z_{j} \) be greater than 1. Therefore, for any \( c > 1 \), the inequality

\[
z_{j} \geq \frac{1}{c - 1} \left( c \sum_{i=1}^{i'} y_{ij} - 1 \right)
\]

is used to derive the upper bound of \( \sum_{i} y_{ij} \). From this relationship, if \( i' \leq i \), we have

\[
z_{j} \geq \frac{1}{c - 1} \left( c \sum_{i=1}^{i-1} y_{ij} - 1 \right)
\]

When we define \( \beta_{i'j} = \left( \frac{1}{c} + \frac{1}{f_{j}} \right) \frac{d_{i'}}{d_{ij}} \), the update rule of \( z_{j} \) in (3.15) is used as following:

\[
z_{j} = z_{j}(1 + \beta_{i'j}) + \beta_{i'j} \frac{1}{c - 1}
\]

From the definition of \( c \), (a) holds due to the relationship \( 1 + \beta_{i'j} \geq \left( 1 + \delta \right)^{1/\delta} \) when \( 0 \leq \beta_{i'j} \leq \delta \leq 1 \). Also, the definition of \( z_{j} \) in (3.15) has an upper bound \( z_{j} \leq \varepsilon \leq (1 + \delta) + \frac{1}{c} \), and, therefore, we can rewrite (3.32) as following:

\[
\sum_{i=1}^{i'} y_{ij} \leq \log_{e}(\varepsilon(c - 1) + 1) - \beta_{i'j}.
\]

Thus, an upper bound of \( \sum_{i=1}^{i'} y_{ij} \) is derived as:

\[
\sum_{i=1}^{i'} y_{ij} \leq \log_{e}(\varepsilon(c - 1) + 1) - \beta_{i'j} + 1
\]

where (a) hold when \( \sum_{i=1}^{i'} y_{ij} = \sum_{i=1}^{i'-1} y_{ij} + 1 \) if task \( i' \) is allocated. Then, if \( \delta = \beta_{i'j} = 0 \), we can have a lower bound \( 1 + \log_{e}(1 + \delta)c^{\delta_{i'j}} = 2 \).
3.6.3 Proof of Lemma 4

Proof. By using the definition of $z_j$ and $x_i$, we derive the change of the objective function of problem (P), denoted by $\Delta P$. When a task $i$ is allocated to an edge node $j$, $z_j$ and $x_i$ are updated, and, therefore, the objective function of problem (P) increases. In particular, $\Delta P$ increase with $\Delta z_j$ since we want to observe the increment of $z_j$ at current iteration while the value of $z_j$ can be updated multiple time. Also, $\Delta P$ increase with $x_i$ since $x_i$ is initially given by 0 and updated only once. Thus, we have the result:

$$\Delta P = \Delta z_j + t_{tot}x_i + \Delta u_i$$

$$= \left(\frac{1}{r_j} + \frac{1}{f_j}\right) \frac{d_i}{t_{tot}} \left(z_j + \frac{1}{c-1}\right) + t_{tot}(1-z_j)\alpha \frac{1}{\left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i} + \Delta u_i$$

$$\leq \left(\frac{1}{r_j} + \frac{1}{f_j}\right) \frac{d_i}{t_{tot}} \left(z_j + \frac{1}{c-1}\right) + t_{tot} \left(1-z_j\right) \left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i + \Delta u_i$$

$$= \frac{t_{tot}}{\left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i} \left(1 + \frac{1}{c-1}\right) + \Delta u_i,$$

where (a) holds due to $\left(\frac{1}{r_j} + \frac{1}{f_j}\right) \frac{d_i}{t_{tot}} \leq 1$ with $\alpha = 1$. Next, the objective function of problem (D) is increases by one, and it is denoted by $\Delta D = 1$. This is due to the fact that $y_{ij}$ is initially set to zero, and we update $y_{ij} = 1$ when task $i$ is assigned to edge node $j$. Hence, we have $\frac{\Delta P}{\Delta D} \leq \frac{t_{tot}}{\left(\frac{1}{r_j} + \frac{1}{f_j}\right) d_i} \left(1 + \frac{1}{c-1}\right) + u_i$. \qed
Chapter 4

Blockchain Systems with Wireless Mobile Miners

4.1 Background, Related Works, and Contributions

Blockchains will be an integral part of the emerging IoT system [103]. Blockchain applications can securely store data without a central trusted authority by leveraging distributed consensus mechanisms. Recently, blockchains have been adopted in a number of wireless and IoT domains [63, 110–113]. For instance, the authors in [63] investigate the performance of a blockchain system operating in a vehicular network in which each vehicle mining node only has a limited time period to exchange blockchain information. The authors in [110] develop a mobile blockchain application that executes a mining process on a mobile device platform. The work in [111] proposes a firmware update scheme for the autonomous vehicles by using a blockchain to ensure the authenticity and integrity of software updates. Also, the authors in [112] develop a false-report attack detection scheme in a vehicular network by exploiting the moving vehicles to compute a consensus mechanism of a blockchain for the sake of data authentication. Moreover, the work in [113] introduces edge computing used to maintain a blockchain for operation of mobile applications.

This prior art on mobile blockchains [63, 110–113] generally assumes that the miners store the ledger while updating their ledger by communicating with each other. However, a device cannot perform networking functions such as updating their ledger if the miners are unable to maintain a stable network connectivity due to the randomness of the wireless channel and the dynamic networking environment (e.g., as experienced in a vehicular network). Also, a device that does not have sufficient computing capabilities cannot perform mining functions. Therefore, it is more effective to consider a novel blockchain architecture in which mining and networking functions are migrated to two different types of nodes, respectively. Moreover, the prior works in [63, 110–113] do not account for the impact of the transmission latency at the wireless link of mobile devices on the blockchain performance. Furthermore, the existing literature mostly relies on isolated experimental results focused on simplistic use cases. In contrast, a rigorous and generalized performance analysis of mobile blockchains in a wireless environment is needed to show how the parameters of a wireless blockchain system can affect its performance metrics such as forking (i.e., a situation when the node who is eligible to verify a block fails to be the first to transmit a verification message to other nodes) and device energy consumption. Consequently, unlike the existing literature [63, 110–113] which investigates the use cases of mobile blockchains that store the ledger at the miners, our goal is to design and analyze a novel blockchain system using wireless MMs such as drones, vehicles, or computationally capable mobile nodes to process the mining computation while the ledgers are stored at the CNs connected to MMs. For instance, our architecture can be used to operate a blockchain over an existing wireless communication system that includes a network consisting of low...
computing capability devices. In our architecture, the existing network nodes are reused as CNs while state-of-the-art MMs are newly deployed to process computational tasks. Therefore, the proposed architecture can facilitate different applications in manufacturing systems of smart factories [111, 114], software security of intelligent transportation systems [63, 112], and automation systems in smart building and cities [115].

The main contribution of this chapter is a novel, mobile blockchain system architecture and the performance analysis of the proposed system. In our architecture, each MM is connected to a CN via a wireless link, and the computing result of an MM is transmitted to other MMs through the backhaul network that interconnects the CNs. In such an architecture, forking events can occur when an MM propagates its computing result to other MMs, since the transmission latency between an MM and its associated CN can be large due to the wireless and mobile nature of the system. To this end, we derive an exact closed-form expression for the probability of occurrence of a forking event, as a function of the wireless network parameters such as the number of MMs and the MM power consumed by the MMs for computing, transmission, and mobility. We use the derived metric to find the average energy consumption required for an MM to earn a reward by processing a blockchain consensus algorithm. Our analytical result shows that the delay required for movement and the possibly high latency resulting from a wireless link can incur a forking event. Simulation results corroborate the analytical derivations and show that the energy consumption for blockchain computation can be reduced by using a lower transmission power and decreasing movement of each MM.

### 4.2 Mobile Blockchain Architecture

Consider a blockchain network that consists of a set \( \mathcal{I} \) of \( I \) MMs and CNs as shown in Fig. 4.1. CNs essentially represent fixed wireless network infrastructure such as base stations associated with MMs. MMs can be computationally capable mobile devices such as industrial drones or ground vehicles gathering transaction data from other ground devices (e.g., see [63] for additional motivation on the use of such mobile mining nodes). Mining computation is executed by each MM that can be either an independent computing node or the head node of a local edge computing cluster as shown in Fig. 4.1 [110]. We mainly focus on drone-type MMs due to their ability to flexibly travel and move in nearly unconstrained locations [66]. However, our model can accommodate any other type of MMs. In Fig. 4.1, MMs in remote locations are unable to directly communicate each other. Therefore, each MM is associated with a different CN that is connected to a backhaul network.

In the considered system, the ledger is located at the CNs while the MMs are used for computing. MMs process
computational tasks required to run a consensus mechanism. For instance, a consensus can be guaranteed by using a PoW where MMs. When transaction records are stored as blocks at the CNs, those blocks must be validated by PoW schemes so as to guarantee that the transaction in the block is original. Then, the CNs can delegate the PoW computation needed for this validation to the wireless MMs. Once each MM completes its PoW computation, an acknowledgment (ACK) message is sent from the MM to the associated CN, called the source CN. The source CN propagates the reception of the ACK message to other CNs through the backhaul links among the CNs. We assume high-bandwidth, fiber backhaul links between CNs and, hence, the message propagation latency in backhaul network will be negligible. By using the considered architecture, mobile blockchain can be applied to different applications as follows.

### 4.2.1 Mobile Blockchain Applications

By using the considered blockchain architecture, the conventional systems having low computational capability can be modified to adopt and operate a blockchain-based application. Particularly, conventional manufacturing systems can be converted into a smart factory using a blockchain to maintain the system up-to-date [111] and run machine learning-based factory quality control software [114]. In a conventional factory, a static legacy control system attached to manufacturing hardware is connected to a network, but it does not have enough computing capability. Therefore, the mobile nodes can be deployed as MMs to assist the computation. Also, by deploying mobile edge computing overlaid on the top of the conventional manufacturing system, MMs can provide a sufficient computing power to the legacy system, while the existing network of the legacy systems can be reused to exchange the messages with other MMs in remote locations. Moreover, in smart factory or smart grid [115], industrial IoT devices owned by multiple operators will need to call for consensus for a purpose of system setup or security enhancement. To this end, wireless surveillance UAVs and warehouse robots can be used as MMs, while CNs are wired-networked static robotic assemblers and machines. Furthermore, in a vehicular network [112], MMs are vehicles and drones, and CNs are infrastructure such as road-side units and traffic lights. Finally, in smart building and home automation, MMs are surveillance unmanned robots and cellular devices, and the CNs are the access points installed in the building [95]. Thus, the conventional systems can be upgraded to use a blockchain by integrating the communication capability of CNs and the high computation power of MMs.

### 4.2.2 System Model: Computation, Mobility, and Transmission

In a blockchain system, the reception order of the ACK messages from multiple MMs to the backhaul network should be identical to the order of completion of the PoW computation. If the ACK message sent earliest arrives to the CNs interconnected by the backhaul network later than other ACK messages, it will lead to a so-called forking event. In a blockchain system, the MM that completes the PoW with the shortest delay will receive a unit of reward. However, when a forking event occurs, the MMs can no longer discern which MM completed the current PoW computation with the shortest delay. Thus, the probability of occurrence of a forking event is an important metric in a blockchain system that we will derive in Section 4.3. This metric will also allow us to analyze the average number of PoW computations and average MM energy consumption required to earn a reward by processing blockchain computation at an MM.

In Fig. 4.1, the computing latency $s_i$ and transmission latency $t_i$ of an MM $i \in I$ are the realization of random variables $S_i$ and $T_i$, respectively. We assume that $S_i$ and $T_i$, for all MMs $\forall i \in I$, will follow identical probability distributions. This is reasonable for the case in which all independent MMs are set to use the same computing power and wireless parameters. For notational simplicity, we use $S$ and $T$ to denote $S_i$ and $T_i$, respectively. Therefore, we will derive the probability distributions of $S$ and $T$.

According to the PoW, all MMs start their PoW computation at the same time and keep executing the PoW computation until one of the MMs completes the computational task by finding the desired hash value [110]. When an MM executes the computational task for the PoW of the current block, the time period needed to finish this PoW computation will
be an exponential random variable $S$ whose distribution is $f_S(s) = \lambda_c e^{-\lambda_c s}$ where $\lambda_c = \lambda_0 P_c$ refers to the computing speed of an MM with $P_c$ being the power consumption for computation of an MM and $\lambda_0$ being a constant scaling factor.

Once an MM finishes its PoW computation for the current block, the ACK message must be delivered to the associated CN so that other MMs can stop their current PoW computation. We derive the transmission latency when an MM transmits the ACK message to the associated CN via a wireless link. Under a Rayleigh fading channel, the small-scale fading gain between an MM and the CN is a random variable $H$ with distribution $f_H(h) = \exp(-h)$ where the statistical average gain of the Rayleigh fading is unity. We assume that MMs move along a circular trajectory around the associated CN and, thus, they have a constant path loss $g$. Then, the signal-to-noise-ratio (SNR) of any MM at its associated CN is the realization for the random variable given by:

$$\Gamma_0 = gH P_{tx}/\sigma_n^2,$$

where $P_{tx}$ is the transmit power of an MM, and $\sigma_n^2$ is the noise power. Since $H$ is the only random variable in $\Gamma_0$, the distribution of the random variable $\Gamma_0$ will be $f_{\Gamma_0}(\gamma) = k_0 e^{-k_0 \gamma}$ where $k_0 = \sigma_n^2/(gP_{tx})$.

An MM transmits the ACK if the channel gain is higher than a threshold $\gamma_0$ that can be seen as the minimum SNR required to decode the transmitted data at the receiver. Hence, achieving an SNR higher than $\gamma_0$ is necessary to transmit the ACK data from an MM to the CN. In particular, each MM observes the SNR at any given location, and if the SNR is lower than $\gamma_0$, the MM moves to another location to obtain a better SNR. Hence, each MM will dynamically seek a location that yields an SNR higher than $\gamma_0$. The number of new location that an MM needs to visit can be given by $n + 1, n \in \mathbb{Z}^{>0}$. At a given location, the probability that a certain MM achieves an SNR higher than $\gamma_0$ is $p_s = \Pr(\Gamma_0 \geq \gamma_0) = 1 - F_{\Gamma_0}(\gamma_0) = e^{-k_0 \gamma_0}$ where $F_{\Gamma_0}(\gamma)$ is the cumulative probability distribution of random variable $\Gamma_0$. Hence, the number of movements, $N$, can be modeled using a geometric distribution with the probability mass function:

$$f_N(n) = (1 - p_s)^n p_s.$$  

In order to change the small-scale fading gain by moving from one location to another, an MM needs to move by a distance of $\lambda/2$ where $\lambda$ is the wavelength of the carrier frequency. The time period needed to move by a distance of $\lambda/2$ is given by $t_m = (\lambda/2)/v$ where $v$ is the speed of the MM. The power consumed\(^1\) to move an MM is $P_m$. Therefore, the movement latency of the MM during $N$ movements becomes $T_m = t_m N$.

After finishing $N$ movements\(^2\), the probability density function of SNR $\Gamma$, be $f_{\Gamma}(\gamma) = g(\gamma)/(1 - F_{\Gamma}(\gamma_0))$ where $g(\gamma) = f_{\Gamma_0}(\gamma) = k_0 e^{-k_0 \gamma}$, if $\gamma_0 < \gamma$, and $g(\gamma) = 0$, otherwise. Therefore, the probability distribution of $\Gamma$ is rewritten as

$$f_{\Gamma}(\gamma) = \begin{cases} k_0 e^{-k_0 (\gamma - \gamma_0)}, & \text{if } \gamma_0 < \gamma, \\ 0, & \text{otherwise.} \end{cases}$$

The data rate of the MM is $R = B \log_2 (1 + \Gamma)$ where $B$ is the bandwidth. The wireless transmission latency of the MM in the uplink will then be $T_u = K/R$ where $K$ is the size of the ACK message. Then, the probability density function of $T_u$ becomes $-f_{\Gamma}(\gamma)\frac{d}{dt}v(t)$ where $v(t) = 2\tilde{\gamma}t - 1$, and, therefore, we have

$$f_{T_u}(t) = \begin{cases} k_0 e^{-(2\tilde{\gamma}t - 1 - \gamma_0)} \frac{k \ln^2 2 \tilde{\gamma}}{B t^2}, & \text{if } 0 < t < \bar{t}, \\ 0, & \text{otherwise,} \end{cases}$$

\(^1\)The power consumption needed to move a drone-type MM will be $P_m = P_H(v) + P_t(v)$ [66]. The closed-form equations of $P_H(v)$ and $P_t(v)$, are given in equations (57)-(59) in [66].

\(^2\)By setting zero velocity in our system model, a MM can be seen as a stationary mining node connected to backhaul network via a wireless link.
where \( \bar{t} = \frac{K}{(B \log_2(1 + \gamma_0))} \) is the largest wireless transmission latency in the uplink since the SNR is higher than \( \gamma_0 \). Thus, the total transmission latency including both movement and wireless transmission latencies becomes \( T = T_m + T_u \).

By using the random variables \( S, T_m, \) and \( T_u \), the energy consumption of an MM in a single round of the PoW computation becomes a random variable given by:

\[
E = P_c S + P_m T_m + P_{tx} T_u.
\]

Next, we analyze the performance of the proposed system by deriving the probability of no forking and the average energy consumption of an MM.

### 4.3 Average Energy Consumption Analysis

We now analyze the performance of the proposed mobile blockchain system in terms of the occurrence of a forking event and the MM energy consumption. The occurrence of a forking event increases the energy consumption required to complete the current block’s PoW due to the increment of the total computing and transmission latency used for the PoW re-computation. Therefore, the probability of having no forking event (called \textit{no-forking probability} hereinafter), that we derive next, is essential to derive the average MM energy consumption until a reward is earned by an MM.

#### 4.3.1 No-forking Probability

For the PoW computation of a block, the MM that is the first to finish its PoW is indexed by \( i^* \), i.e., \( i^* = \arg\min_{i \in \mathcal{I}} S_i \). Therefore, when \( s_{i^*} < s_{i'}, \forall i' \in \mathcal{I} \setminus \{i^*\} \), the ACK of MM \( i^* \) should arrive to the source CN so that the ACK information is propagated to all CNs via the backhaul network before any ACKs from other MMs arrive, i.e., \( s_{i^*} + t_{i^*} < s_{i'} + t_{i'} \) as shown in Case 1 of Fig. 4.1. However, the order of arrival of ACK messages from multiple MMs to the source CN can be different, i.e., \( s_{i^*} + t_{i^*} > s_{i'} + t_{i'} \) as shown in Case 2 of Fig. 4.1. The change in the order of arrival happens when the transmission latency of MM \( i' \) is shorter than that of MM \( i^* \), i.e., \( t_{i^*} > t_{i'} \), due to the different mobility patterns and the wireless transmission latency. Therefore, the ACK message of MM \( i' \) can arrive at the source CN earlier than the ACK message of MM \( i^* \), thus resulting in a forking event. We next derive the no-forking probability by calculating the probability that the ACK from MM \( i^* \) arrives earlier than any ACK from any other MM \( i' \in \mathcal{I} \setminus \{i^*\} \).

\textbf{Theorem 7.} The no-forking probability is given by

\[
p_n = e^{-\lambda_c (I-1)} \int_0^{t_{i^*}} \left( \int_0^{t_{i^*}} e^{-\lambda_c (t_{i^*} - t)} f_T(t) dt + \int_{t_{i^*}}^\infty f_T(t) dt \right)^{I-1} f_S(s_{i^*}) ds_{i^*} f_T(t_{i^*}) dt_{i^*}.
\]

\[(4.3)\]

\textbf{Proof.} See Appendix 4.6.1.

The no-forking probability \( p_n \) is derived as a closed-form function of the wireless parameters such as the number of MMs, the MMs’ transmission power, and the computing speed. If a forking event occurs, additional energy is needed for processing computation for the next block, and, hence, we analyze the average MM energy consumption next.
4.3.2 Average Energy Consumption

Since a forking event incurs an additional round of computation, the number of PoW computations needed to earn a reward follows a geometric distribution with mean of \(1/p_n\). Also, MM \(i^*\) in each PoW computation will consume the average energy \(E = P_cE[S_i] + P_{tx}E[T_u] + P_mE[T_m]\). Our goal is to derive the average energy consumption of MM \(i^*\) in each PoW computation round until a block’s PoW computation is completed without forking, i.e., \((1/p_n)E\).

We outline how to derive \(E\) by finding the average latency of the computation, movement, and wireless transmission, i.e., \(E[S_i], E[T_u],\) and \(E[T_m]\), respectively. Since the shortest computing latency among all MMs is \(S_i^*\), the complementary cumulative probability distribution (CCDF) of \(S_i^*\) is given by
\[
\Pr(S_i^* > z) = \Pr\left(\min_{i \in I} (S_i) > z\right) = \prod_{i=1}^{I} \Pr(S_i > z) = (1 - \Pr(S \leq z))^I.
\]

Therefore, the average computational latency of MM \(i^*\) is derived as
\[
E[S_i^*] = \int_{0}^{\infty} (1 - \Pr(S \leq z))^I \, dz = \int_{0}^{\infty} e^{-\lambda_c I z} \, dz = \frac{1}{\lambda_c I}.
\]

The average latency due to mobility is given by \(E[T_m] = t_m (e^{(\gamma_0 \sigma_n^2)/(g p_s)} - 1)\) since the average of \(N\) is \((1 - p_s)/p_s = e^{(\gamma_0 \sigma_n^2)/(g p_s)} - 1\). Also, the average wireless transmission latency can be calculated by using the CCDF of the probability of \(T_u\) given by
\[
\Pr(T_u > z) = \begin{cases} 
1 - e^{-k_0(2^{\frac{K}{10}} - 1 - \gamma_0)}, & \text{if } 0 \leq z \leq I \\
0, & \text{otherwise.}
\end{cases}
\]

The average of the transmission latency is:
\[
E[T_u] = \int_{0}^{I} 1 - e^{-k_0(2^{\frac{K}{10}} - 1 - \gamma_0)} \, dz
\]

Hence, by combining \(E[S_i^*], E[T_u],\) and \(E[T_m]\), we can have a closed-form expression of \((1/p_n)E\).

4.4 Simulation Results

For our simulations, we consider that an MM is associated with a CN at a distance of 50 m, and the path loss gain \(g\) is calculated by using a free space model due to air-to-air communications. The power spectral density of the noise is \(-174\) dBm/Hz, and the bandwidth is 180 kHz. The ACK message size is set to 1 Mbits. To model the computational speed of an MM, the power consumption for computation is set to 8 W and the scaling factor \(\lambda_0\) is 0.04.

Figs. 4.2 and 4.3 show the no-forking probability and the average energy consumption per MM. First of all, Figs. 4.2 and 4.3 show that simulation and analysis results are matched. In Fig. 4.2, a forking event can occur with a high
Figure 4.2: Probability of a no forking event in one PoW computation.

Figure 4.3: Average energy consumption of an MM required to compute the PoW of a block.
probability as the number of the MMs increases. This is due to the fact that, as more MM join in the PoW computing, the blockchain network is more likely to use an MM having a lower ACK reception period than that of MM $i^*$. Fig. 4.3 shows that the average energy consumption to complete the PoW computation decreases as the number of MMs increases because using more MMs reduces the PoW computation time period, thus decreasing the energy consumption. For example, using 20 MMs can reduce the energy consumption by up to 94.5% compared to using 1 MM in a blockchain system.

Fig. 4.4 shows the energy consumption required to complete the PoW computation. The energy consumption of an MM increases with the SNR threshold $\gamma_0$ and the transmission power $P_{tx}$. This is because the energy consumption for mobility increases with $\gamma_0$. Also, the total energy consumption can increase with the transmission power because a high transmission power increases the forking probability, thus increasing the number of repeated PoW computations.

4.5 Conclusion

In this chapter, we have proposed a novel framework using MMs to operate a blockchain over a wireless mobile network. We have derived the no-forking probability and the average MM energy consumption required to earn a reward. Simulation results have shown that the wireless transmission power, SNR threshold, and the number of MMs significantly impact the energy consumption of the MMs. The analytical results serve as a nexus to minimize the forking probability over wireless and device energy consumption as a future work. Also, the proposed framework will be extended to analyze the forking event within other consensus mechanisms such as proof-of-stake algorithm in the scenarios of wireless networks.
4.6 Appendix

4.6.1 Proof of Theorem 7

Proof. As a first step, suppose that the shortest computing latency is \( s^* \), and that MM \( i^* \) has a transmission latency \( t^* \). Given \( s^* \) and \( t^* \), the probability that all MMs other than MM \( i^* \), do not incur a forking event becomes:

\[
\Pr \left( \bigcap_{i' \in \mathcal{I} \setminus \{i^*\}} s^* + t^* < S_{i'} + T_{i'} | S_{i^*} = s^*, T_{i^*} = t^*, s^* < S_{i^*} \right)
\]

\[
= \prod_{i' \in \mathcal{I} \setminus \{i^*\}} \Pr \left( s^* + t^* < S_{i'} + T_{i'} | S_{i^*} = s^*, T_{i^*} = t^*, s^* < S_{i^*} \right)
\]

\[
= \left( \Pr (s + t < S + T | S = s^*, T = t^*, s^* < S) \right)^{I-1}
\]

\[
= \left( \int_0^\infty \int_0^{\infty} f_S(s) s^* < S f_T(t) t^* dt ds \right)^{I-1}
\]

\[
= e^{-\lambda^*(I-1)} \int_0^{t^*} e^{-\lambda^*(t^* - t)} f_T(t) dt + \int_{t^*}^\infty f_T(t) dt.
\]

(4.4)

The equality (a) in (4.4) holds since the MMs independently process the computation and transmit the ACK message. Also, equality (b) in (4.4) holds since all MMs have identical distributions for \( S_i \) and \( T_i \). From the equality (c) in (4.4), the computing latency of an MM is non-negative and is greater than \( s^* \), i.e., \( s = \max(0, s^*) \). To avoid a forking event, the value of the computing latency has to be greater than \( s^* + t^* - t \), i.e., \( s = \max(0, s^* + t^* - t) \). Therefore, the range of the computing latency is \( \max(s^*, s^* + t^* - t), \infty \). Moreover, if \( s^* \) and \( t^* \) are given, the PoW computing latency of all MMs other than MM \( i^* \) must be greater than \( s^* \). Therefore, the computing latency of MM \( i^* \) becomes the conditional probability distribution given by:

\[
f_S(s | s^* < S) = \frac{\lambda^* e^{-\lambda^* s}}{1 - F_S(s^*)} = \frac{\lambda^* e^{-\lambda^* s}}{e^{-\lambda^* s^*}}.
\]

(4.5)

Thus, given \( s^* \) and \( t^* \), the no-forking probability yields (4.4).

Next, the values of \( s^* \) and \( t^* \) in (4.4) follow the probability distributions \( f_S(s^*) \) and \( f_T(t^*) \), respectively. By integrating (4.4) multiplied with \( f_S(s^*) \) and \( f_T(t^*) \) over the intervals of \( s^* \) and \( t^* \), the no-forking probability is derived as \( p_n \) in (4.3).
Chapter 5

Computational Caching in Edge Networks

5.1 Background, Related Works, and Contributions

Emerging 5G applications such as augmented reality (AR) and virtual reality (VR) require ultra low latency transmission and computation [5]. The latency requirements of these applications cannot be accomplished by using traditional cloud computing due to the round-trip delay needed to reach the cloud [3]. Thus, edge computing is proposed as an extension of cloud computing in which end-user devices, called edge nodes, perform key functions such as storing and computing data at the network edge [4]. Leveraging the physical proximity of edge nodes and pooling their resources allows for low-latency computation.

As mentioned in Section 1.2, exploiting the caching capabilities of edge nodes is deemed as an essential step to improve system throughput and reduce the latency [44, 73–77]. In edge computing, edge nodes can further reduce their latency by caching the input data needed to process their computational operations. The prior works on caching for edge computing [44, 74–77] generally assume that the operation of the application running on the edge node has one corresponding input data given by either a single file or a set of files. Therefore, the goal is to gather all parts of the required input data. Such a caching technique can be viewed as data caching. However, for a given computational operation, multiple files can possibly be used for processing. Among the many possible input files, the operation can select a specific file as input. When each input file represents the intermediate IR of an operation, storing a partial set of the possible input files can be seen as caching IRs, or more formally computational caching. We therefore propose the paradigm of computational caching as a technique to reduce the computational latency at an edge node. Consequently, unlike the existing literature [44,74–77] that considers data caching, under prior knowledge on user and application behavior, our goal is to design an online approach to enable an online computational caching framework, under uncertainty, while minimizing the transmission and computational latency.

The main contribution of this chapter is a novel framework for online computational caching in an edge network. This framework allows a given edge node to download the necessary IR from a neighboring edge node and use the downloaded or stored IR for computations by selecting the most suitable input in the presence of uncertainty on the arrival order of the user’s operation. We formulate an online computational caching problem whose objective is to minimize the sum of the transmission and computational latency. To solve this problem, we propose an online computational caching algorithm. Then, we derive a competitive ratio of the formulated online problem. In addition,

1Note that computational caching here is different from the original notion of computational caching used in [116] in which computer networks cache the act of computation, i.e., they store the trajectories that are encountered during the execution of a software’s instructions, and apply it in new contexts.
we propose a bandwidth allocation scheme to achieve a desired target competitive ratio. Simulation results show that the proposed online algorithm minimizes the latency while achieving a performance that is near-optimal compared to an offline solution that has full information on all input arrivals.

## 5.2 System Model

Consider an edge network that consists of an edge node $i$ and a set $\mathcal{J}$ of $J$ neighboring edge nodes as shown in Fig. 5.1. Each edge node $i$ is running an AR or VR application supporting six degrees of freedom videos, it must process a huge volume of input data that includes video clips at different angles. While using application $k \in \mathcal{K}$, the user of edge node $i$ may want to change the angle of view or watch video scenes related to specific persons. We assume that each application $k$ has $L = |\mathcal{L}|$ different commands. The user’s specific input command is indexed by $l \in \mathcal{L}$. When command $l$ is executed by using application $k$, the computational operation is denoted by $\alpha_{k,l}$.

To compute operation $\alpha_{k,l}$ at time $t$, the required input of the operation will be $\beta_{k(t),l(t)}$. When $\alpha_{k,l}$ is computed, this yields an IR $\delta_{k(l)}$. $\alpha_{k,l}$ can correspond to the operation that is used to find a relevant image or video data about a certain location by using application $k$. Also, there will be another operation $\alpha_{k,l'}, l' > l$, that is used to find the image or video about a more specific location. These two operations $\alpha_{k,l}$ and $\alpha_{k,l'}$ yield outputs $\delta_{k,l}$ and $\delta_{k,l'}$, respectively. Then, $\delta_{k,l}$ can be seen as the result including the broader information, and $\delta_{k,l'}$ can be the result including the information about the more specific requirement. Therefore, $\alpha_{k,l'}$ can be computed by having IR $\delta_{k,l}$ as an input. In other words, if $\delta_{k,l}$ is stored in edge node $i$, then this IR can be reused for other operation $\alpha_{k,l'}, l' > l$. We also define $\delta_{k,0}$ as the raw data that can be used to compute operation $\alpha_{k,1}$, for notational simplicity, even though the raw data is not an IR. In the VR example, $\delta_{k,0}$ can correspond to the raw video data that includes all degree angles of videos.

Edge node $i$ can store (cache) a different set of IRs in its memory (e.g., RAM, flash memory, or others). The set of cached IRs of edge node $i$ at time $t$ is denoted by $\mathcal{R}^{(t)}_{i}$.

The data rate needed to download this information using a wireless link is

$$R_{ji} = B \log_{2} \left( 1 + \frac{P_{ij} R_{ji}}{B N_{0}} \right)$$

where $g_{ji}$ is the wireless channel gain between edge nodes $i$ and $j$, $P_{ij}$ is the transmission power of edge node $j$, $N_{0}$ is the noise power spectral density, and $B$ is the bandwidth. The downloaded IR at time $t$ is denoted by $u^{(t)}_{k(t),l(t)}$. Thus, if $u^{(t)}_{k(t),l(t)}$ is transmitted from edge node $j$ to edge node $i$, the transmission latency will be:

$$T_{ij}(u^{(t)}_{k(t),l(t)}) = \frac{u^{(t)}_{k(t),l(t)}}{R_{ji}^{(t)}}.$$  \hspace{1cm} (5.1)

We assume that the computational latency is quadratically increasing with the size of input $\beta_{k(t),l(t)}$. This assumption can capture the fact that the time-complexity of an application is $\mathcal{O}(n^2)$ where $n$ is the data size. Also, the latency can be scaled by the computational speed [45]. Thus, when input $\beta_{k(t),l(t)}$ is used, the computational latency at edge node $i$ is given by

$$C_i(\beta_{k(t),l(t)}) = \frac{\xi}{c} \left( \beta_{k(t),l(t)} \right)^2,$$ \hspace{1cm} (5.2)

\footnote{For instance, in signal processing or image processing, the discrete Fourier transform can have quadratic time complexity $\mathcal{O}(n^2)$.}
Our goal is to analyze how this system can exploit the IRs stored in different edge nodes so as to minimize its latency. 

Table 5.1: Summary of key notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_{k,l}$</td>
<td>Computational operation of command $l$ in application $k$.</td>
</tr>
<tr>
<td>$r_{k,l}$</td>
<td>IR obtained by computing $\alpha_{k,l-1}$.</td>
</tr>
<tr>
<td>$\beta_{k(t),l(t)}^{(t)}$</td>
<td>Input is IR $r_{k(t),l(t)}$ when computing operation at time $t$.</td>
</tr>
<tr>
<td>$u_{k(t),l(t)}^{(t)}$</td>
<td>Downloaded IR at time $t$ is IR $r_{k(t),l(t)}$.</td>
</tr>
<tr>
<td>$\mathcal{R}^{(t)}_i$</td>
<td>Set of stored IRs in the memory.</td>
</tr>
</tbody>
</table>

where $c$ is the computation speed of the edge node, e.g., value proportional to CPU frequency, and $\xi = 1$ is the unit computational price per bit. The computational latency can decrease as the data size of $\beta_{k(t),l(t)}^{(t)}$ decreases and $c$ increases. Therefore, to compute operation $\alpha_{k,l}$, instead of using other inputs such that $\beta_{k(t),l(t)}^{(t)}=r_{k,l'}$, $l'<l-1$, reusing IR $r_{k,l-1}$ can reduce the computational latency. Clearly, when $u_{k(t),l(t)}^{(t)}$ is downloaded from neighboring edge node $j$ and $\beta_{k(t),l(t)}^{(t)}$ is used to compute the given operation $\alpha_{k,l}$ by edge node $i$, the total latency can be defined by

$$f_i(u_{k(t),l(t)}^{(t)},\beta_{k(t),l(t)}^{(t)}) = T_{ji}(u_{k(t),l(t)}^{(t)}) + C_i(\beta_{k(t),l(t)}^{(t)}).$$

(5.3)

Our goal is to analyze how this system can exploit the IRs stored in different edge nodes so as to minimize its latency.

### 5.3 Problem Formulation

Our goal is to minimize the end-to-end latency by enabling an edge node to properly choose the IR that must be used for the computation of its operations. Since IRs stored in edge node $i$ can be reused to compute future operations,
storing IRs that will be used in the near future is beneficial to reduce the latency. However, in practice, the inputs of the operations can be dynamically determined by the user, and, thus, they cannot be anticipated. As a result, edge node \( i \) is generally unable to know a priori what the future inputs will be and, hence, optimizing the total latency becomes a challenging problem. Since the memory size of edge node \( i \) is limited, not all IRs can be stored. This limited memory has to be properly managed by erasing and storing IRs depending on the inputs. While considering unknown information on future input, downloading a new IR and managing the stored IRs at each time is essential to reduce the latency. Under such uncertainty, determining the IR needed to compute the current operation must also account for a prospective arrival of future operations. In consequence, we introduce a novel online optimization scheme that makes a sequential decision for the online arrival of operations to minimize the latency for computational caching.

We observe the system during a finite period of time \( t \in \mathcal{T} = \{1, \ldots, T\} \). At time \( t \in \mathcal{T} \), one operation \( \alpha_{k(t),l(t)} \) where \( k(t) \in \mathcal{K} \) and \( l(t) \in \mathcal{L} \), \( \forall t \in \mathcal{T} \), arrives to edge node \( i \). Then, the sequence of operations that have arrived during the observation period will be \( \sigma \triangleq (\sigma_t)_{t=1}^T = (\alpha_{k(1),l(1)}, \ldots, \alpha_{k(t),l(t)}, \ldots, \alpha_{k(T),l(T)}) \). For a sequence \( \sigma_t \), the latency minimization problem becomes:

\[
\min_{\mathbf{u}, \mathbf{\beta}} \sum_{t=1}^{T} f_s \left( u^{(t)}_{k(t),l(t)}, \beta^{(t)}_{k(t),l(t)} \right),
\]

\[
\text{s.t.} \quad \sum_{r_k,l \in \mathcal{R}^{(t)}_i} r_{k,l} \leq M_{\text{max}}, \forall t \in \mathcal{T},
\]

\[
\mathcal{R}^{(t)}_i \subseteq \mathcal{R}^{(t-1)}_i \cup \{ u^{(t)}_{k(t),l(t)} \}, \forall t \in \mathcal{T},
\]

\[
\beta^{(t)}_{k(t),l(t)} = r_{k(t),l(t)} \forall t \in \mathcal{T}, l'(t) < l(t), \forall t \in \mathcal{T},
\]

where the time-varying vector of downloaded IRs is \( \mathbf{u} = [u^{(1)}_{k(1),l(1)}, \ldots, u^{(t)}_{k(t),l(t)}, \ldots, u^{(T)}_{k(T),l(T)}] \) and the time-varying vector of the IRs used for computation is \( \mathbf{\beta} = [\beta^{(1)}_{k(1),l(1)}, \ldots, \beta^{(t)}_{k(t),l(t)}, \ldots, \beta^{(T)}_{k(T),l(T)}] \). The goal of (5.4) is to minimize the sum of transmission and computational latency during a finite time period. (5.5) constrains the sum of the stored IRs at time \( t \) within the limited memory size. (5.6) shows that \( u^{(t)}_{k(t),l(t)} \) is the downloaded IR from the network. (5.7) states that any IR \( r_{k(t),l'(t)} \) is also used to compute operation \( \alpha_{k(t),l(t)} \) at time \( t \). IR \( r_{k(t),l'(t)} \) must also be stored in the memory \( \mathcal{R}^{(t)}_i \).

In this problem, \( \sigma_t = \alpha_{k(t),l(t)} \) is revealed to edge node \( i \) at time \( t \), and, then, edge node \( i \) must determine \( \beta^{(t)}_{k(t),l(t)} = r_{k(t),l(t)} \). To make a decision at time \( t \), the previously stored IRs are used. Also, edge node \( i \) can download \( u^{(t)}_{k(t),l(t)} \) from neighboring node \( j \in \mathcal{J} \). For effective computational caching, node \( i \) can form a network in which all IRs can be downloaded from neighbors. To form a network, for example, edge nodes can exchange a beacon signal that contains the information about the stored IR. Then, edge node \( i \) can select the set of neighboring nodes such that each neighbor has a different IR. By doing so, the network of edge node \( i \) can be formed. Hereinafter, we assume that network formation is given using online approaches such as [80].

In this online computational caching problem, edge node \( i \) can reuse one of the stored IRs or download a new IR. While reusing a stored IR can reduce the transmission latency, the computational latency can increase unless IR \( r_{k(t),l(t)} \) was already stored. On the other hand, if edge node \( i \) decides to download IR \( r_{k(t),l(t)} \), transmission latency will be incurred, however, the computational latency will be minimized since \( r_{k(t),l(t)} \) is the IR having the smallest size among those can be used to compute input operation \( \alpha_{k(t),l(t)} \). Therefore, there is a tradeoff between computational latency and transmission latency, and, thus, choosing the appropriate IRs to compute the sequence under uncertainty, is not trivial. Under such incomplete information, finding the optimal solution of (5.4) in a conventional offline manner is clearly not feasible and, therefore, an online solution is needed.
Algorithm 5.3 Computational Caching Algorithm

1: while time $t \leq T$
2: Operation $\alpha_{k(t),l(t)}$ arrives.
3: if new application arrives,
4: mark all stored IRs as removable end if
5: if edge node $i$ has any IR that can be used,
6: Compute by using the best available stored IR.
7: elseif edge node $i$ does not have an available IR,
8: Download IR $r_{k(t),l^*}$ from edge node $j^*$ where
1: $\min_{j \in J, l \in L} T_{ij}(r_{k(t),l}) \ s.t. \ r_{k(t),l} \in \mathcal{R}_j, l < l(t)$.
9: Store the downloaded IR $r_{k(t),l^*} \in \mathcal{R}_i^{(t)}$.
10: Compute by using $r_{k(t),l^*}$.
11: end if
12: Store output $r_{k(t),l(t)}$ in $\mathcal{R}_i^{(t)}$.
13: end while

5.4 Online Computational Caching

We propose an online computational caching algorithm that schedules the IRs to minimize the total latency given by (5.3). To reduce the computational latency, the stored IRs are reused. However, since not all possible IRs can be stored within a limited memory size, the edge node may download IRs from neighboring nodes. In such a case, if the wireless data rate is low, the transmission latency can become a bottleneck in minimizing the total latency. Thus, to prevent a large latency over the wireless links, we introduce an online algorithm focusing on minimizing the transmission latency.

The online algorithm must also manage the limited size of memory by evicting outdated IRs. For storing IRs, if the memory does not have sufficient free space, then existing IRs must be removed, to include the new data. Therefore, the current decision to remove a certain IR can also affect the set of the stored IRs in the future. Then, to manage the stored IRs, we define two events in which all stored IRs are marked so that the marked items can be evicted from the memory.

The proposed online computational caching algorithm shown in Algorithm 5.3 is a transmission-centric algorithm that minimizes the use of wireless resources. Thus, if any IR, i.e., $r_{k(t),l}, l < l(t)$, is stored in $\mathcal{R}_i^{(t)}$, it is used to compute $\alpha_{k(t),l(t)}$ without downloading other IRs. However, if edge node $i$ does not have an IR to compute $\alpha_{k(t),l(t)}$, it downloads IR $r_{k(t),l^*}$ from neighboring node $j^*$ so as to minimize the transmission latency $T_{ij}(r_{k(t),l})$ such that $l < l(t)$. The downloaded IR is stored in $\mathcal{R}_i^{(t)}$ and used to compute the operation. Then, the output of this operation $r_{k(t),l(t)}$ is stored in the memory.

In Algorithm 5.3, after computing the incoming operation at each time or after downloading the IR, the output IR is stored for possible future computation. To store the output IR, edge node $i$ must have free memory space. If $\mathcal{R}_i^{(t)}$ does not have enough space, some of the previously stored IRs must be erased. To this end, we propose a modified marking algorithm. In the so-called paging problem, a marking algorithm is typically developed to evict a marked page from the memory when eviction is necessary [90]. To exploit the marking-and-eviction structure from marking algorithms, we define two events to mark all stored IRs. The motivation behind designing two events is to determine the stored IRs that are unlikely to be used in the near future. The first event is triggered when the application type is changed. For instance, if the application at time $t - 1$ is different from the current application at $t$, all IRs in the memory are marked as erasable. In the second event, all IRs are marked if the sum of all unmarked IRs in the memory reaches or exceeds the maximum memory capacity.
Once IRs are marked by the two events, the eviction scheme will follow a least-recently-used (LRU) algorithm \[90\]. LRU replaces an old page that is least recently used if one empty slot for a new page is required. We propose an IR eviction scheme (IRES) that gives priority to recently used IRs. Similar to LRU, IRES removes the least-recently-used IR from \( R_i^{(t)} \), but it only selects the IR that is marked by the two proposed events. IRES is different from LRU in that LRU replaces an old page that is least recently used if one empty slot for a new page is required. We propose an IR eviction scheme (IRES) that gives priority to recently used IRs. Similar to LRU, IRES removes the least-recently-used IR from \( R_i^{(t)} \), but it only selects the IR that is marked by the two proposed events. IRES is different from LRU in that LRU replaces one page to store a page, but IRES will repeatedly remove the marked IRs until enough free space for a new IR is available.

Next, we analyze the online computational caching problem by using competitive analysis \[90\]. Competitive analysis measures the performance of an online algorithm denoted by \( ALG(\sigma) \) by comparing it to the performance of an ideal, offline optimal algorithm represented by \( OPT(\sigma) \). Then, a competitive ratio can be defined by \( \Gamma(\sigma) = ALG(\sigma) / OPT(\sigma) \). While an online algorithm has information only on the current and past input sequence of operations, the offline optimal algorithm is ideal and knows the entire input sequence \( \sigma \). For analysis, we divide the whole input sequence \( \sigma \) into multiple partitioned sequences using two marking events. While the online algorithm processes input sequence \( \sigma \), if one of two marking events is triggered at time \( t' \) and time \( t'' \), respectively, the partitioned sequence \( \hat{\sigma} \) will be defined as \( (\sigma_{t'}, \ldots, \sigma_{t''}) \). By doing so, for an online algorithm, any input sequence can be divided into partitioned sequences. For notational simplicity, we omit the index for the partitioned sequences. Then, we respectively measure the minimum latency of the offline optimal algorithm and the maximum latency of an online algorithm during processing \( \hat{\sigma} \). If the input operations are given by \( \hat{\sigma} \), the minimum latency achieved by the optimal offline algorithm is denoted by \( OPT(\hat{\sigma}) \). To derive a lower bound for \( OPT(\hat{\sigma}) \), we find the lower bound of the transmission and computational latencies, respectively. The lower bound of the transmission latency can be zero if edge node \( i \) does not download any IRs. The minimum computational latency is achieved by using the smallest size of IR for each input operation in \( \hat{\sigma} \). Therefore, \( OPT(\hat{\sigma}) \) is at least larger than the minimal computational latency to compute operations \( \hat{\sigma} \), so \( OPT(\hat{\sigma}) \geq \sum_{t=t'}^{t''-1} C_i(r_{k(t),l(t)}-1) \). We denote the latency of the online algorithm by \( ALG(\hat{\sigma}) \). To find the upper bound of \( ALG(\hat{\sigma}) \), we consider the worst-case scenario in which node \( i \) does not have any IR. As an example of this worst case, if the requests of applications arrive in a round-robin fashion, any cached IR will be removed and can no longer be used. Here, \( ALG(\hat{\sigma}) \) becomes at most the sum of the transmission and computational latency for all operations in \( \hat{\sigma} \). Thus, \( ALG(\hat{\sigma}) \leq \sum_{t=t'}^{t''-1} T_{ij}(r_{k(t),l(t)}-1) + C_i(r_{k(t),l(t)}-1) \).

The competitive ratio with respect to \( \sigma \) is upper bounded by the largest competitive ratio of a partitioned sequence denoted by \( \hat{\sigma} \). Thus, the competitive ratio can be found as follows:

\[
\Gamma(\sigma) \leq \max_\sigma \{ \Gamma(\hat{\sigma}) \},
\]

\[
\leq \max_\sigma \left\{ \frac{\sum_{t=t'}^{t''-1} (\rho_t + 1) C_i(r_{k(t),l(t)}-1)}{\sum_{t=t'}^{t''-1} C_i(r_{k(t),l(t)}-1)} \right\},
\]

(5.8)

where \( \rho_t \) is defined by \( T_{ij}(r_{k(t),l(t)}-1) = \rho_t C_i(r_{k(t),l(t)}-1) \).

### 5.4.1 Bandwidth allocation

From (5.8), we propose a wireless bandwidth allocation scheme that yields a constant competitive ratio for problem (5.4). Suppose that a target performance is predetermined by \( \rho \). Then, when edge node \( j \) transmits an IR to edge node \( i \), the bandwidth used by neighboring node \( j \) is the solution of:

\[
B \log_2 \left( 1 + \frac{g_{ji} P_{ij}}{B N_0} \right) = \frac{c}{\rho \xi r_{k(t),l(t)}-1}.
\]

(5.9)

If the maximum bandwidth that edge nodes can access is \( B_{max} \), then \( 0 \leq B \leq B_{max} \). Since the left-hand side of (5.9) corresponds to the data rate \( P_{ij}^{(t)} \), the data rate is scaled by \( \rho \) in (5.9). Now, we find the competitive ratio by using \( \rho \).

**Proposition 2.** For a given \( \rho \), if there exists \( B \) satisfying (5.9), \( 0 \leq B \leq B_{max} \), then the competitive ratio becomes \( \rho+1 \).
Table 5.2: Simulation parameters.

<table>
<thead>
<tr>
<th>Figure</th>
<th>( c ) [bps]</th>
<th>( B_{\text{max}} )</th>
<th>( M_{\text{max}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 2</td>
<td>( 30 \times 10^{14} )</td>
<td>3 MHz</td>
<td>Variable</td>
</tr>
<tr>
<td>Fig. 3</td>
<td>( 30 \times 10^{14} )</td>
<td>Variable</td>
<td>90 Mbits</td>
</tr>
<tr>
<td>Fig. 4</td>
<td>( 5 \times 10^{14} )</td>
<td>10 MHz</td>
<td>90 Mbits</td>
</tr>
</tbody>
</table>

Table 5.3: Computational latency in the case of Fig. 5.2.

<table>
<thead>
<tr>
<th>No comp. caching</th>
<th>Proposed algorithm</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.33 sec</td>
<td>5.38 sec</td>
<td>5.19 sec</td>
</tr>
</tbody>
</table>

Proof. From (5.8), by replacing \( \rho_t, \forall t \in T \), with \( \rho \), the competitive ratio of all partitions becomes \( \rho + 1 \), and it provides the upper bound of the competitive ratio with respect to \( \sigma \).

This show how \( \rho \) can be a design parameter used to adjust the competitive ratio. For any given \( \rho \) satisfying Proposition 2, the competitive ratio is fixed as \( \rho + 1 \). Also, from (5.9), the bandwidth allocation scheme results in a higher data rate if computational speed \( c \) increases and the size of stored IR \( r_{k(t),l(t)} - 1 \) decreases. The competitive ratio (5.8) is derived from (5.4) without any assumption on the online algorithms. Hence, this competitive ratio can be applied to any online algorithm for our model.

5.5 Simulation Results

For our simulations, we consider that an edge node is connected to eight neighboring edge nodes uniformly distributed within 50 m. As in [80], we set \( P_{\text{tx},i} = 20 \text{ dBm} \) and the power spectral density of the noise is -174 dBm/Hz. Also, the channel gain is \( g_{ji} = \gamma_1 d_{ji}^{\gamma_2} \) where \( d_{ji} \) is the distance between edge nodes \( i \) and \( j \), \( \gamma_1 = 10^{-3} \), and \( \gamma_2 = 4 \). Each neighboring edge node stores a different IR. Two applications are used by the user with equal probability while the arrival of four operations at each time follows the Zipf distribution [77]. The size of IRs is given by \( [r_{k,0}, r_{k,1}, r_{k,2}, r_{k,3}] = [50, 30, 20, 10] \text{ Mbits}, \forall k \in K \). We set \( T = 10 \), and all statistical results are averaged over a large number of simulation runs. For comparison, an exhaustive search with complete knowledge about the inputs is used to find the optimal latency when IRES is used for managing the memory policy. Other parameters are shown in Table 5.2.

In Fig. 5.2, we show the total latency defined in (5.4) for different sizes of the edge node \( i \)'s memory with \( B = B_{\text{max}} \). Since the computational latency does not change significantly along with \( M_{\text{max}} \), the value averaged over different \( M_{\text{max}} \) for each case is presented in Table 5.3. From this figure, we can first see that the proposed computational caching algorithm decreases the latency. Compared to the case without computational caching, the proposed algorithm can reduce the latency by up to 26.8% at \( M_{\text{max}} = 90 \text{ Mbits} \). In the case of no computational caching, the latency is reduced at \( M_{\text{max}} = 100 \text{ Mbits} \), since all raw data can be stored in the memory. However, in computational caching, the latency decreases as the memory size increases since all possible IRs cannot be stored in \( M_{\text{max}} \). Also, Fig. 5.2 shows that the gap between the proposed online and offline solutions, in terms of the total latency, is 3% at \( M_{\text{max}} = 80 \text{ Mbits} \). From Fig. 5.2, we can also see that the total latency decreases as the memory size increases since the transmission latency can be reduced by caching more IRs in a larger size memory. For instance, the latency decreases by 13.6% if \( M_{\text{max}} \) increases from 80 Mbits to 130 Mbits.

Fig. 5.3 shows the total latency for different maximum accessible bandwidth when \( B = B_{\text{max}} \). We first see that the latency decreases with the increasing bandwidth used to transmit IRs. For example, the proposed algorithm reduces the latency by 57.4% by increasing \( B_{\text{max}} \) from 1.4 MHz to 10 MHz. Moreover, Fig. 5.3 shows that the total latency resulting from the proposed algorithm and the offline scenario are very close. This demonstrates the effectiveness of the proposed online algorithm. Also, in Fig. 5.3, the gap between the proposed online and offline solutions, in
Figure 5.2: Total latency for different memory sizes of edge node \( i \).

Figure 5.3: Total latency for different bandwidths \( B_{\text{max}} \).
terms of total latency, can be reduced by up to 1.7% at $B_{\max} = 10$ MHz. Moreover, compared to the case without computational caching, the total latency can be reduced by up to 32.1% at $B_{\max} = 10$ MHz.

Fig. 5.4 shows the empirical competitive ratio for problem (5.4) with $\rho = 1$. The bandwidth $B$ for each neighboring edge node is calculated from (5.9). We can first see that 32.7% of iterations achieve a competitive ratio of 1 which means that the result of the online algorithm coincides with the offline optimal solution of (5.4). Fig. 5.4 also shows that the empirical competitive ratio in the worst case is shown to be 1.157. From Fig. 5.4, we can also see that the empirical competitive ratio is less than the upper bound $\rho + 1 = 2$ from Proposition 2. Thus, the results from Fig. 5.4 show that Algorithm 5.3 can effectively select the input IR, in an online manner, while minimizing latency.

5.6 Summary

In this chapter, we have proposed a novel framework for online computational caching in an edge network that allows the optimization of the selection of the input IR under uncertainty on the arrival order of the user’s operation. We have formulated an online computational caching problem to minimize the transmission latency and computational latency. The proposed online algorithm schedules the IRs to compute each of the sequentially arriving operations while managing the stored IRs in the memory. We have also shown an upper bound of the competitive ratio for the formulated online problem. Simulation results have shown that the proposed online computational caching algorithm is effective in reducing latency.
Chapter 6

ON/OFF Scheduling of Energy Harvesting Base Stations

6.1 Background, Related Works, and Contributions

Despite their promising potential for enhancing the capacity and coverage of cellular systems, small cell networks (SCNs) can also increase the overall power consumption of a cellular system since the access network and edge facilities take up to 83% of mobiles’ operator power consumption [117]. To this end, enhancing the energy efficiency of dense SCNs has emerged as a major research challenge [118]. In particular, there has been a recent significant interest, not only in minimizing energy consumption, but also in maximizing the use of green energy by deploying energy harvesting, self-powered BSs that rely solely on renewable and clean energy for operation [119]. Thus, deploying self-powered BSs is currently being demonstrated by various network operators. For instance, LG Uplus deploys solar-powered LTE BSs in mountain areas of South Korea [120], and, also, a large solar-powered BS cluster is deployed in Tibet by China Mobile [121]. Clearly, one can realize the vision of truly green cellular networks by deploying self-powered, energy harvesting SBSs that rely solely on renewable energy for their operation [122].

Recently, numerous works have focused on the use of energy harvesting techniques in cellular networks [16, 16, 81, 123–134]. For instance, the work in [123] overviews key design issues for adopting energy harvesting into cellular networks and propose energy harvesting-aware user association and BS sleep mode optimization problems. With regards to the user association problem in energy harvesting scenarios, the authors in [124] consider a model in which wireless BSs are powered by both grid power and green energy in energy harvesting heterogeneous cellular networks. For this model, the authors propose a user association scheme that minimizes the average traffic delay while maximizing the use of green energy. Furthermore, the authors in [125] propose a probabilistic framework to model energy harvesting and energy consumptions of BSs and investigate a distributed user association problem when BSs is powered by energy harvesting. Also, to study the problem of user association, in [126], the authors considered a network in which the uncertainty of energy harvesting is modeled within a competitive market with the SBSs being the consumers who seek to maximize their utility function.

Reaping the benefits of self-powered SBSs mandates effective and self-organizing ways to optimize the ON and OFF schedules of such SBSs, depending on uncertain and intermittent energy arrivals. Therefore, several recent works have focused on optimizing energy efficiency in energy harvesting systems by intelligently turning BSs ON and OFF [81, 127–131]. For instance, the authors in [127] provide a model to measure the performance of heterogeneous networks with self-powered BSs. In [128], when BSs are powered by both a renewable source and the power grid, the authors propose an algorithm to maximize the utilization of green energy so that the grid power consumption can be
minimized. Moreover, the work in [129] develops a number of algorithms to minimize grid power consumption when considering hybrid-powered BSs. For solving a capital expenditure minimization problem, the authors in [130] propose an ON/OFF scheduling method for self-powered BSs. The work in [131] investigates the problem of minimizing grid power consumption and blocking probability by using statistical information for traffic and renewable energy. The authors in [81] study the optimal BS sleep policy based on dynamic programming with the statistical energy arrival information.

In this existing body of literature that addresses ON/OFF scheduling in energy harvesting networks [81, 128–131], it is generally assumed that statistical or complete information about the amount and arrival time of energy is perfectly known. However, in practice, energy arrivals are largely intermittent and uncertain since they can stem from multiple sources. Moreover, turning SBSs ON and OFF based on every single energy arrival instance can lead to significant handovers and network stoppage times. Further, the existing works [123, 124, 129], and [131] on energy harvesting networks often assume the presence of both smart grid and energy harvesting sources at every SBS. In contrast, here, we focus on cellular networks in which SBSs are completely self-powered and reliant on energy harvesting. In [135] and [136] the problem of ON/OFF scheduling of base stations is studied for a heterogeneous network using reinforcement learning. However, these works are focused on classical grid-powered networks and do not take into account the presence of energy harvesting in the system. Also, unlike the work in [127] which focuses on the global performance analysis of self-powered SBSs, our goal is to develop self-organizing and online algorithms for optimizing the ON/OFF schedule of self-powered SBSs.

The main contributions of this chapter is to develop a novel framework for optimizing the ON and OFF schedule of self-powered SBSs in a cellular network in which multiple SBSs coexist with a macrocell base station (MBS). In particular, an optimization problem is formulated that seeks to minimize the operational cost that captures both the power and delay of the system by appropriately determining the SBSs ON and OFF scheduling, in the presence of complete uncertainty on the energy harvesting process. We cast the problem as an online optimization and we analyze its properties. We show that, under an illustrative case, an algorithm achieves a competitive ratio, defined as the ratio of an online algorithm’s to the optimal cost of an offline algorithm, of 2. Then, to overcome the complexity of the original problem, an approximation is derived and shown to allow the decomposition of the original problem into a set of distributed online optimization problems that are run at each SBS. To solve the resulting per-SBS online optimization problem, a novel approach based on the ski rental problem, a powerful online optimization tool [92], is proposed. In particular, we present two schemes to solve the ski rental problem: a deterministic online algorithm (DOA) and a randomized online algorithm (ROA). On the one hand, the DOA is a benchmark scheme designed to turn each SBS OFF at a predetermined time so as to achieve a competitive ratio of 2. On the other hand, the ROA enables the SBSs to make a decision according to a probability distribution, and it can achieve an optimal competitive ratio of $e/(e-1)$ which provides an upper bound for the approximated problem. The proposed algorithms allow the SBSs to effectively decide on their ON/OFF schedule, without knowing any prior information on future energy arrivals. To the best of our knowledge, this is the first work that exploits the online ski rental problem for managing energy uncertainty in cellular systems with self-powered SBSs. Simulation results show that the empirical competitive ratio of using the ROA to solve the original problem is 1.86. This demonstrates that the ROA achieves a reasonable performance gap compared to the ideal, offline optimal solution found by exhaustive search. Also, our results show that the ROA can decrease the total operational cost compared to the DOA and a baseline approach. Moreover, the ROA can reduce total energy consumption of SBSs and per-SBS network delay compared to DOA or a baseline that turns SBSs ON during the same fixed period for all SBS. This performance advantage is shown to reach up to 15.6% and 11.4% in reducing the energy consumption of a network relative to a baseline and the DOA, respectively. The ROA also decreases the delay per SBS up to 20.6% and 8.4% relative to a baseline and the DOA, respectively. In particular, we observe that the ON time of each SBS is affected by various factors including the harvested energy and the power consumption of BSs.

The rest of this chapter is organized as follows. In Section 6.2, the system model is presented. In Section 6.3, we present the problem formulation. In Section 6.4, we propose online algorithms based on the ski rental framework. In Section 6.5, the performance of the proposed algorithm is demonstrated with using extensive simulations. Finally, conclusions are drawn in Section 6.6.
6.2 System Model

Consider the downlink of a two-tier heterogeneous small cell network in which an MBS is located at the center of a service area. In this network, a set $\mathcal{J}$ of $J$ self-powered SBSs are deployed. Moreover, we define the set of all BSs as $\mathcal{B} = \{0, 1, 2, \cdots, J\}$ where the MBS is indexed by 0. We assume that the SBSs and the MBS will use different frequency bands and, therefore, the MBS and the SBSs will not interfere. In contrast, within the SBS tier, frequency bands may be reused and, as such, the SBSs will interfere with one another. In this system, when activated, the SBSs can offload traffic from the MBS, thus reducing the overall network congestion. A set $\mathcal{I}$ of $I$ UEs is randomly distributed in the coverage of the MBS where each UE can access either an SBS or MBS. Each UE can be connected with only one of the BSs at a certain time $t$ within a period of $T$.

An illustration of our system model is shown in Fig. 6.1. In our considered system, while the MBS is connected to the conventional power grid, SBSs are self-powered and rely exclusively on energy harvesting sources. In such case, the self-powered SBSs will operate as a means to boost capacity and to complement the existing grid powered MBS. For example, SBSs can be equipped with solar panels to procure energy for their operation, or, alternatively, they can use wireless power transfer from MBS transmissions. Since the characteristics of the harvested energy can be highly dynamic, we do not make any specific assumption on the energy harvesting process. Thus, our model can accommodate any type of energy harvesting mechanism. To enhance the overall energy efficiency of the system, we assume that the SBSs can dynamically turn ON or OFF, depending on the network state, energy harvesting state, and other related parameters. To manage the intermittent and uncertain nature of energy harvesting, energy storage systems (ESS) can be used. Energy harvesting is assumed to be done irrespective on whether an SBS is turned ON or OFF. Thus, an SBS will store energy in its ESS when it is turned OFF, and this stored energy can be used when it is turned ON to service users. Also, when it is turned ON, an SBS can store the excess of harvested energy if instantaneous harvested energy is enough to operate an SBS.

At time $t$, the ON or OFF state of SBS $j$ is denoted by $\sigma_j(t)$ which is defined as follows:

$$
\sigma_j(t) = \begin{cases} 
1, & \text{if SBS } j \text{ is turned ON at time } t, \\
0, & \text{otherwise.}
\end{cases}
$$

(6.1)

For the MBS, $\sigma_0(t) = 1$ since the MBS is always turned ON. The set of switched-ON BSs at time $t$ is denoted by $\mathcal{B}^\text{on}(t) = \{j|\sigma_j(t) = 1, \forall j \in \mathcal{B}\}$. Similarly, the set of switched-OFF BSs can be shown as $\mathcal{B}^\text{off}(t) = \mathcal{B} \setminus \mathcal{B}^\text{on}(t)$. 

Figure 6.1: System model of a heterogeneous deployment with self-powered SBSs.
6.2.1 Network Performance

We model the network performance between BS and UE. In the downlink, the signal to interference and noise ratio (SINR) between UE $i$ and SBS $j \in J$ at time $t$ can be shown as

$$\gamma_{ij}(\sigma(t)) = \frac{P_{tx}^j \sigma_j(t) h_{ij}}{\sum_{j' \in B_0 \setminus \{j\}} P_{tx}^j \sigma_{j'}(t) h_{ij'} + \rho^2},$$

(6.2)

where $\sigma(t) = [\sigma_j(t) | \forall j \in J]$, $h_{ij}$ is the channel gain between UE $i$ and SBS $j$, $P_{tx}^j$ is the transmit power of the connected SBS $j$, and $\rho^2$ is the noise power. If an UE is associated with an SBS, the UE can receive interference from the other SBSs. On the other hand, when a UE is associated with the MBS, the UE does not experience any interference from the SBSs. Therefore, the user association can be given by:

$$j^*(i, \sigma(t)) = \arg\max_{j \in B_0} \gamma_{ij}(\sigma(t)).$$

(6.4)

By using the user association rule in (6.4), the user association of whole network is updated at each time $t$. Then, the set of UEs associated with the same BS $j$ can be defined by

$$\mathcal{I}_j(\sigma(t)) = \{i \mid j^*(i, \sigma(t)) = j, \forall i\}.$$  

(6.5)

The set $\mathcal{I}_j(\sigma(t))$ changes over time $t$ according to the user association results from (6.4). If $j \neq 0$, then $\mathcal{I}_j(\sigma(t))$ indicates the set of UEs associated with SBS $j$. Otherwise, when $j = 0$, then $\mathcal{I}_0(\sigma(t))$ indicates the set of UEs connected to the MBS. Subsequently, the set of all UEs $\mathcal{I}$ can be divided into $J + 1$ subsets at most, each of which is denoted by $\mathcal{I}_j(\sigma(t))$, $j \in J$. Thus, each UE should be associated with one of BSs at any time $0 \leq t \leq T$ from (6.4), and, thus, we have $\mathcal{I} = \bigcup_{j=0}^{J} \mathcal{I}_j(\sigma(t))$, $0 \leq t \leq T$.

When the user association is determined by (6.4), the achievable data rate of UE $i$ is given by

$$c_{ij}(\sigma(t)) = \frac{B}{|\mathcal{I}_j(t)|} \log_2(1 + \gamma_{ij}(\sigma(t))),$$

(6.6)

where $|\mathcal{I}_j(t)|$ is the number of UEs associated with SBS $j$ at time $t$, and $B$ is the bandwidth of an SBS ($B = B_s$) or MBS ($B = B_m$). When the MBS can transmit data to UEs using bandwidth $B_m$, time slots are scheduled for the $|\mathcal{I}_0(t)|$ UEs using a round robin scheduling. In the considered model, whenever a file of $K$ bits needs to be transmitted to each UE, we can define the total transmission delay between BS $j$ and all UE in $\mathcal{I}_j(t)$ at time $t$ as

$$\phi_j(\sigma(t)) = \sum_{i \in \mathcal{I}_j(\sigma(t))} \frac{K}{c_{ij}(\sigma(t))}.$$  

(6.7)

6.2.2 Power Consumption

Next, we define the power consumption models for the MBS and SBSs. When modeling the power consumption of BSs, the resource utilization of a BS monotonically increases as the number of UE connections increases. Thus, the
power consumption of a BS increase as the utilization become higher. The power consumption model for a BS includes two components: the utilization-proportional power consumption and the fixed power consumption. The utilization-proportional power consumption depends on the signal processing functions and, hence, it varies depending on the number of associated UEs at a BS. Meanwhile, the fixed power components pertain to the power consumed due to components such as the power amplifier or the cooler. Thus, a fixed amount of power is required to operate the BS regardless of the number of the associated UEs. The power consumption of a BS at time $t$ is therefore given by:

$$\psi_j(\sigma(t)) = \frac{|I_j(\sigma(t))|}{M} (1 - q) P_{j,\text{op}} + q P_{j,\text{op}}^\text{max},$$

(6.8)

where $q$ is a weighting parameter that captures the tradeoff between the utilization-proportional power consumption and the fixed power, $P_{j,\text{op}}^\text{max}$ is the maximum power consumption when the BS is fully utilized, and $M$ is the maximum number of UE connections. If the type of BS $j$ is a MBS, then we set $M = M_m$, and, if BS $j$ indicates an SBS, then $M = M_s$. The MBS can provide service to the larger number of UEs since the MBS has higher computing capability than an SBS; thus, the different service capabilities can be presented by $M_m \geq M_s$. Also, $P_{j,\text{op}} = \alpha P_{j,\text{op}}^\text{max}$ where the constant $\alpha$ denotes the fraction of the transmit power $P_{j,\text{op}}^\text{max}$ out of the total the maximum operational power $P_{j,\text{op}}^\text{max}$. For example, if $q = 1$, the BS consumes constant power regardless of the utilization level of the BS. On the other hand, if $q = 0$, the power consumption of the BS is proportional to the utilization, which is a more realistic BS power consumption model. Note that $\psi_j(\sigma(t))$ is the power required to turn ON SBS $j$ at time $t$, and it depends on the number of UEs associated with SBS $j$.

As mentioned, SBSs use energy harvesting as a primary energy source, so an ESS can be used to store the excess energy for future use. The available amount of energy at time $t$ is given by

$$E_j(t) = \min \left( \int_0^{t-\epsilon} \Omega_j(\tau) d\tau - \int_0^t \psi_j(\sigma(\tau)) d\tau, E_{\text{max}} \right), \forall j \in \mathcal{J},$$

(6.9)

where $E_j(t) \geq 0$ is the stored energy of SBS $j$ at time $t$, $\psi_j(\sigma(t))$ is the consumed power of SBS $j$, $\Omega_j(t)$ is the amount of energy arrival of SBS $j$, $\epsilon$ is a small number, and $E_{\text{max}}$ is the maximum capacity of ESS. $\Omega_j(t)$ captures the uncertainty of energy harvesting in the time domain. Since an SBS solely relies on the energy harvesting, if $E_j(t)$ becomes zero at a certain time $t$, SBS $j$ is turned OFF at time $t$, and the UEs connected to SBS $j$ are handed over to other SBSs or the MBS according to the user association rule (6.4).

### 6.2.3 Operational Expenditure of Base Stations

Given the defined network delay and power consumption models, we define operational costs incurred when using an SBS or MBS. First, we account for the operational cost of a given SBS per unit time when an SBS is turned ON. In the ON state, UEs associated with SBS $j$ experience the network delay given by $\phi_j(\sigma(t))$. Since higher delay is an unfavorable aspect, the operational cost has to increase with the network delay of UEs. Moreover, while an SBS is turned ON, it will incur a power consumption cost. Thus, to turn SBS $j$ ON at time $t$, the required cost of using SBS $j$ can be defined by

$$r_j(\sigma(t)) = \alpha_D \phi_j(\sigma(t)) + \alpha_P \psi_j(\sigma(t)),$$

(6.10)

where the constant $\alpha_D$ is the monetary cost per unit transmission delay, and the constant $\alpha_P$ is the monetary cost per unit power consumption. $\alpha_D$ and $\alpha_P$ can be used to change the weighting of delay and power consumption. The delay and energy are combined in (6.10) so as to balance the tradeoff between the two metrics. The cost $r_j(\sigma(t))$ of a given SBS $j$ can vary over time due to the fact that the user association of UEs can change between two different times $t$ and $t'$, i.e., $I_j(\sigma(t)) \neq I_j(\sigma(t'))$. Thus, different user associations can result in different $\phi_j(\sigma(t))$ and $\psi_j(\sigma(t))$ since the data rate of each UE and the number of connected UEs per SBS are different.

Next, we model the cost for using the MBS. When self-powered SBSs rely solely on the harvested energy that is highly uncertain and intermittent, they might need to turn OFF if they have no more energy. Therefore, to avoid the
risk of such energy depletion, the SBSs can go into an energy-saving OFF state to store additional energy for future use. Due to this energy storage need, the system can end up with a large number of OFF SBSs which, in turn, will degrade the network performance as it increases congestion at the MBS and the ON SBSs. Thus, to prevent such a network congestion, if SBS \( j \) decides to switch OFF, we assume that it will be charged a cost \( b_j \). By setting a flat-rate cost \( b_j \), the network can control how often the SBSs can turn OFF, particularly when they still have a sufficient amount of energy stored. Here, as \( b_j \) increases, the penalty of turning a given SBS \( j \) OFF becomes larger; thus, the SBSs will have an incentive to maintain the ON state as long as possible. In a dynamic network, the ON and OFF states of the SBSs can change over time thus also changing the user association. In such a dynamic network, finding an exact, flat rate \( b_j \) is difficult. Therefore, we propose to derive this cost based on a worst-case assumption. In particular, to define the cost \( b_j \), first we find the maximum cost of using the MBS which is then scaled by a parameter \( \alpha_B \in [0, 1] \). The cost \( b_j \) is the maximum cost that can be incurred by turning OFF and transferring traffic to the MBS. To find the maximum cost of using the MBS in the worst case, suppose that all UEs can be associated with the MBS so that the network delay and power consumption of the MBS are maximized. Here, when a portion of the maximum cost is incurred to an SBS, the incurred cost can depend on the UEs in the SBS denoted by the set \( I_j(\sigma(0)) \). By doing so, the maximum cost of using the MBS can be divided into the per-SBS costs. If UE \( i \in I_j(0) \) is connected to the MBS, the transmission delay of UE \( i \) will be \( \frac{K}{T\log_2(1 + \gamma_i(0))} \). By summing over all UEs in \( I_j(0) \), we obtain the network delay corresponding to the UEs in \( I_j(0) \), as shown as

\[
\Phi_{0}^{I_j(0)} = \sum_{i \in I_j(0)} \frac{K}{T\log_2(1 + \gamma_i(0))}.
\]

Also, the portion of the power consumption of the MBS that is consumed by the UEs in \( I_j(0) \) will be:

\[
\Psi_{0}^{I_j(\sigma(0))} = \frac{|I_j(\sigma(0))|}{M} (1 - q) P_{0}^{op} + q P_{0}^{op}.
\]

Consequently, whenever an SBS \( j \) decides to turn OFF, the accompanying cost, due to the handover to the MBS, will be given by:

\[
b_j = \alpha_B \left( \alpha_D \Phi_{0}^{I_j(\sigma(0))} + \alpha_P \Psi_{0}^{I_j(\sigma(0))} \right) T,
\]

where \( \alpha_B \in [0, 1] \) is the fraction of the maximum cost. For example, when we set \( \alpha_B = 0.10 \), then 10% of the maximum cost of using the MBS during time period \( T \) will be incurred to SBS \( j \). Thus, if the value of \( b_j \) is too high, being turned ON becomes an affordable option, so SBS \( j \) is turned ON until the whole harvested energy is used. On the other hand, if the value of \( b_j \) is low, SBSs tend to be turned OFF to keep the stored harvested energy due to a low penalty in switching SBSs OFF.

### 6.3 Problem Formulation

Given the operational costs, our goal is to analyze the optimal ON and OFF scheduling problem for the SBSs. In cellular networks consisting of self-powered SBSs, the amount of available energy is dynamically changing and very limited. To be able to operate using energy harvesting as a primary energy source of SBSs, self-powered SBSs should intelligently manage their ON and OFF states considering delay, power, and energy state. Moreover, since future energy arrivals can be highly unpredictable, optimizing the ON and OFF schedule of SBSs is a very challenging problem. By properly scheduling its OFF duration, an SBS can reduce its energy consumption while also storing more energy for future use. However, at the same time, the SBS must turn ON for a sufficient period of time to service users and offload MBS traffic. In our problem, information on energy arrival is unknown, so an online optimization approach is suitable. To cope with the inherent uncertainty of energy harvesting while balancing the tradeoff between energy consumption and network delay, we introduce a novel, self-organizing online optimization framework for optimizing the ON and OFF schedule of self-powered SBSs.
6.3.1 ON/OFF Scheduling as an Online Optimization Problem

We formulate the global ON and OFF scheduling problem with the goal of minimizing the sum of costs that encompass the costs of using an SBS and the MBS in (6.10) and (6.13), as follows:

\[
\min_{\sigma(t), x} \sum_{j=1}^{J} \left( \int_{0}^{u_j} r_j(\sigma(\tau))\sigma_j(\tau)d\tau + b_j x_j \right), \tag{6.14}
\]

\[\text{s.t.} \quad \sigma_j(t) + x_j \geq 1, \quad 0 \leq t \leq u_j, \quad \forall j, \tag{6.15}\]

\[\sigma_j(t) \in \{0, 1\}, \quad 0 \leq t \leq u_j, \quad \forall j, \tag{6.16}\]

\[x_j \in \{0, 1\}, \quad \forall j, \tag{6.17}\]

where \(x = [x_j | \forall j \in J]\), respectively. The ON and OFF states of SBS \(j\) at time \(t\) is denoted by \(\sigma_j(t)\) in (6.16). Also, \(x_j\) in (6.17) indicates whether SBS \(j\) is determined to be turned OFF before SBS \(j\)’s stored energy is depleted at time \(u_j\). In (6.15) and (6.16), time \(t > u_j\) is not considered since SBS \(j\) is turned OFF due to energy depletion. Note that \(u_j\) is the first moment when energy harvesting constraint (6.9) is not satisfied. Thus, each SBS can experience energy depletion at a different time \(u_j\) since the amount of energy arrival of SBS \(j\) denoted by \(\Omega_j(t)\) is unknown before time \(t\), and SBS \(j\) cannot know the future energy status, as observed in many real-world scenarios [137]. For example, when energy is harvested from the environment, the amount of harvested energy can quickly change due to factors such as weather conditions which can change rapidly during are changing in a short period of time. Not only the sudden weather, long-term seasonal changes also brings uncertainty into energy harvesting. Therefore, the uncertainty of the harvested energy at each moment can be captured by \(\Omega_j(t)\), and, thus, the energy depletion time \(u_j\) is unknown in our problem. In essence, our problem is online where energy harvesting brings in uncertainty about the future event. The period \(T\) can be defined in various ways. For example, \(T\) can be defined as a short period of time during which the SBS can stay ON using a fully charged battery.

Also, it is required to reduce the network congestion by increasing the use of the harvested energy, so the ON time of each SBS needs to be extended. In problem (6.14), if an SBS is turned OFF due to energy depletion, the cost of using the MBS is not incurred to the SBS so as to provide incentives for SBSs to maintain a longer ON period. However, if SBS \(j\) is turned OFF according to its decision, the cost of using the MBS is incurred to the SBS, as captured by setting \(x_j = 1\). Therefore, the ON and OFF scheduling solution given by \(\sigma_j(t)\) and \(x_j\) can be determined by SBS \(j\) during \(1 \leq t \leq u_j\) so that UEs in \(I_j(\sigma(t))\) can be connected to either SBS \(j\) \((\sigma_j(t) = 1)\) or the MBS \((x_j = 1)\) by satisfying constraint (6.15).

If the problem is offline, then it can be readily solved. For example, in the offline scenario, the optimal solution is either always ON strategy \((\sigma_j(t) = 1, 0 \leq t \leq u_j, x_j = 0)\) or OFF strategy \((\sigma_j(t) = 0, 0 \leq t \leq u_j, x_j = 1)\). When \(u_j\) is known in offline, it is possible to compute the total costs corresponding to a strategy that the SBS uses. Thus, since the SBS can compare the costs of all possible solutions, the optimal solution can be found. However, such offline scenario is not available in real environment due to the uncertainty of energy harvesting as mentioned above. Thus, the problem (6.14) needs to be considered in an online optimization framework.

To solve (6.14), one must develop a suitable online algorithm. To assess the effectiveness of such an algorithm, we need to use competitive analysis. Competitive analysis [138] is a method used to compare between the performance of online algorithms and that of an optimal offline algorithm. One key metric in competitive analysis is the so-called competitive ratio, defined next:

**Definition 1.** The competitive ratio of an online algorithm is defined by

\[
\kappa = \max_{u_j} \frac{\beta_{\text{ALG}}(u_j)}{\beta_{\text{OPT}}(u_j)}, \quad \forall u_j, \tag{6.18}
\]

where \(u_j\) is a random time instant when harvested energy is depleted, \(\beta_{\text{ALG}}(u_j)\) is the cost of an online algorithm that corresponds to the total cost of the problem (6.14), and \(\beta_{\text{OPT}}(u_j)\) is the optimal cost achieved by using an offline algorithm that knows all input information.
When we use an online algorithm, our goal is to find an algorithm that minimize the competitive ratio $\kappa$. Therefore, in competitive analysis, the competitive ratio is meaningful since it shows the performance of an online algorithm [90]. For this analysis, the competitive ratio of online algorithms is evaluated for a given arbitrary input sequence that corresponds to uncertain energy arrivals. In our model, the arbitrary input sequence is characterized by $u_j$ that is the moment of energy depletion. From the competitive analysis, even though an SBS does not know the input sequence, the use of online algorithms will give a solution that can at least achieve the cost of $\kappa\beta_{OPT}(u_j)$.

To analyze this problem, first, we consider two special cases in which: a) $r_j(\sigma(t))$ is decreasing over time or b) $r_j(\sigma(t))$ is increasing over time. If an SBS’s $r_j(\sigma(t))$ decreases, the SBS can have motivation to extend its ON time since the cost of using SBS becomes inexpensive. Thus, the SBS can simply extend the ON time. On the other hand, if $r_j(\sigma(t))$ increases, the SBS has less motivation of maintaining the ON state. Moreover, in this case, it is possible that the SBS could stay in the OFF state from the beginning if the SBS knew the increasing of $r_j(\sigma(t))$. Therefore, since the SBS cannot change its previous decisions in the case in which $r_j(\sigma(t))$ is increasing, it is difficult to minimize the total cost.

Thus, we present an example case where the cost of using an SBS $r_j(\sigma(t))$ decreases as the time $t$ increases. By doing so, we can propose an ON and OFF scheduling algorithm that achieves a finite competitive ratio. Note that the decreasing of $r_j(\sigma(t))$ can be physically observed when an SBS increases the transmission power so that it can decrease the delay cost of the SBS as shown in our simulations. In such case, we propose an online algorithm in which the SBS is turned OFF at a predetermined time $\bar{t}$. When the value of $r_j(\sigma(t))$ decreases, each achieved value for $r_j(\sigma(t))$ will be denoted by $r_v$. These values are then arranged in a descending order where $v$ indicates the order of a given value $r_v$, as follows:

$$r(1) > r(2) > \cdots > r(v-1) > r(v).$$

(6.19)

Here, we note that, $r_j(\sigma(t))$ changes from $r_{v-1}$ to $r_v$ at time $t_{v-1}$, and $r_v$ stays constant from $t_{v-1}$ to $t_v$ where $t_0 = 0 < t_1 < t_2 < \cdots < t_{v-1} < t_v$.

**Theorem 8.** When $r_j(\sigma(t))$ decreases over time $t$ in the problem (6.14), the initial SBS’s OFF time is given by $\bar{t} = b_j/r(1)$ at time $t_0$. Also, at time $t_{v-1}$, $v \geq 2$, the SBS’s OFF time is updated using the following equation:

$$\bar{t} = \bar{b}_j \frac{b_j}{r(v)} - \frac{1}{r(v)} \sum_{v' = 1}^{v-1} t_{v'} \left(r_{v'} - r_{v'+1}\right).$$

(6.20)

Then, the OFF time $\bar{t}$ increases when it is updated by (6.20). Also, an online OFF time scheduling algorithm that uses $\bar{t}$ can achieve a competitive ratio of 2.

**Proof.** See the Appendix. □

In Theorem 8, at the time in which the SBS’s cost $r_j(\sigma(t))$ is updated, the SBS update its ON time by setting a larger value for $\bar{t}$. Thus, the updated $\bar{t}$ effectively optimizes the problem.

To investigate more dynamically changing $r_j(\sigma(t))$ needs to be considered. However, since the value of $r_j(\sigma(t))$ depends on the ON/OFF state of SBSs in a network, the exact value of a future $r_j(\sigma(t))$ cannot be known and expected. For instance, if the neighboring SBSs are turned OFF, the interference at SBS $j$ will be reduced thus increasing the data rate of UEs that are associated with SBS $j$. This, in turn, results in a smaller delay cost and reduces $r_j(\sigma(t))$. At the same time, UEs associated with other, neighboring SBSs may be handed over to SBS $j$. Then, the number of UEs served by SBS $j$ increases thus increasing the delay cost. In addition, due to the increase of the number of UEs, the power consumption of SBS $j$ also increases thus yielding a higher $r_j(\sigma(t))$. As seen from these illustrative scenarios, the OFF scheduling of the various SBSs can either increase or decrease $r_j(\sigma(t))$. Therefore, the cost of using a given SBS will not always be monotonically increasing or decreasing thus making it very challenging to find a solution to the optimization problem in (6.14) by estimating the future variation of $r_j(\sigma(t))$ over time $t$. Moreover, to solve (6.14),
the ON and OFF states of all SBS must be collected by the network which can generate additional signaling overhead for information exchange. This can also require the use of a centralized controller. Naturally, in a dense SCN, such centralized control may not be possible or scalable.

Consequently, in essence, our goal is to devise a self-organizing approach in which the solution to (6.14) can be done locally at each SBS. Clearly, solving this problem for a generic, non-monotonically changing \( r_j(\sigma(t)) \) is challenging and, therefore, we need to use an approximation. One natural way is to assume that \( r_j(\sigma(t)) \) is not time-varying, which can simplify the problem because the interference and user association that change over time do not need to be considered, as discussed next.

### 6.3.2 Approximated Problem

To relax the time dependence from \( r_j(\sigma(t)) \), we assume that the cost will be equal to \( r_j = r_j(\sigma(0)) \). In other words, the initial cost, which is generally known to the network, will be used as a flat cost of using an SBS. This approximation can help simplify the problem by considering a worst-case assumption for the interference, as follows. As mentioned, the cost \( r_j(\sigma(t)) \) incurred to an SBS \( j \) is affected by interference when other SBSs are randomly turned OFF. However, by approximating \( r_j(\sigma(t)) \) using a constant value, the scheduling decisions will no longer be dependent and, thus, each SBS can make its own decision without having global knowledge about other SBSs’ ON and OFF states. Note that the largest value of the interference is captured in the approximated problem since all SBSs are turned ON at the beginning. Thus, the SBSs can compute the value of \( r_j \) even though all SBSs are not actually turned ON. One key advantage of the proposed approach is that an SBS can determine the solution at the beginning of each period \( T \). Thus, distributed optimization can be done by computing locally, and also it reduce network overhead since signaling is not required. Here, the approximated problem can be given by:

\[
\min_{\sigma(t), x_j} \sum_{j=1}^{J} \left( \int_0^{u_j} r_j \sigma_j(\tau) d\tau + b_j x_j \right), \tag{6.21}
\]

s.t. \( (6.15), (6.16), \) and \( (6.17) \).

To solve problem (6.21), we decompose it into smaller, per SBS subproblems. As shown next, each SBS can solve an individual optimization subproblem, so the approximated problem in (6.21) can be solved in a distributed way.

**Proposition 3.** The problem in (6.21) can be decomposed into \(|\mathcal{J}|\) subproblems.

**Proof.** The objective function of the problem (6.21) can be shown to be a sum of functions of \( \sigma_j(t) \) and \( x_j \) as shown as (6.21). Thus, changing of \( \sigma_j(t) \) and \( x_j \) does not affect \( \sigma_{j'}(t) \) and \( x_{j'} \), \( j' \neq j \). Therefore, the objective function of (6.21) can be separated into \(|\mathcal{J}|\) functions. Also, each SBS’s energy storage is not connected to other SBSs’ energy source. Thus, due to the isolated energy harvesting system of each SBS, the amount of stored energy shown as (6.9) is managed independently by each SBS. Hence, the problem (6.21) can be decomposed into \(|\mathcal{J}|\) subproblems.

Now, we have \(|\mathcal{J}|\) subproblems derived from the approximated problem in (6.21). The ON or OFF decision of an SBS does not affect the decision of another SBS, so we can solve \(|\mathcal{J}|\) subproblems in parallel. By solving each of the per-SBS problems, we can significantly reduce complexity and overhead while allowing for a self-organizing implementation. Consequently, each SBS will solve its local version of (6.21) that seeks to minimize its individual cost function given by

\[
\min_{\sigma_j(t), x_j} \int_0^{u_j} r_j \sigma_j(\tau) d\tau + b_j x_j, \tag{6.22}
\]

s.t. \( (6.15), (6.16), \) and \( (6.17) \).
Since SBS \( j \) does not know the whole input sequence (e.g., uncertain energy arrivals), the SBS cannot know the optimal schedule of ON and OFF before time elapses. Thus, (6.22) is still formulated as an online optimization problem, for which an online algorithm is needed to make a decision in real time under an uncertain future. Remarkably, the problem in (6.22) is analogous to the so-called ski rental problem [92], an online optimization framework that enables such decision making in face of uncertainty, as discussed next.

6.4 On/Off Scheduling as an Online Ski Rental Problem

First, we will explicitly define the analogy between ski rental and self-powered BS scheduling. In the classical online ski rental problem, an individual is going skiing for an unknown number of days [138]. The uncertainty on the skiing period is due to factors such as nature or whether this individual will enjoy skiing or not. Here, the individual must decide on whether to rent skis over a short period of time or, alternatively, buy them for a long period of time, depending on the costs of renting and buying, the number of days that he/she will end up skiing, and on whether the skiing activity will be enjoyable. The online ski rental framework provides online optimization techniques that allows one to understand how an individual will make a “rent” or “buy” decision in such a scenario while facing uncertainty due to nature and while accounting for the tradeoff between the costs of rental and purchase and the benefits of skiing.

In this regard, our problem in (6.22) is similar to the ski rental decision making process. In our model, each SBS is an individual that must rent its resources (turn ON) to the network under the uncertainty of energy harvesting or alternatively buy more reliable MBS resources (and turn OFF). From (6.10) and (6.13), we can see that \( r_j \) and \( b_j \) will represent the prices for rent and buy, respectively. Thus, the decision of an SBS on how long to turn ON is essentially a decision on how long to rent its resources which require paying \( r_j \) per unit time. Once the SBS turns OFF, the network must buy the more expensive but more reliable MBS resources at a price \( b_j \). Given this analogy, we can develop efficient online algorithms to solve (6.21) [139]. An online algorithm can solve the problem at each present time without having whole information about future energy harvesting results.

To solve the BS ON/OFF scheduling problem, one may consider other methods such as Markov decision processes, dynamic programming, reinforcement learning, or convex online optimization. However, those are not suitable frameworks for studying the problem considered in this work since additional assumption or information on energy harvesting process would be required to model the environment.

We use online algorithms to solve the optimization problem, and competitive analysis is used to study the performance of the online algorithms. We first analyze the optimal offline strategy when assuming energy arrival information over the entire period is given. The offline optimal cost can be shown as

\[
\beta_{\text{OPT}}(u_j) = \begin{cases} 
 r_j u_j, & 0 \leq u_j \leq \frac{b_j}{r_j}, \\
 b_j, & \frac{b_j}{r_j} \leq u_j \leq T. 
\end{cases} 
\]  

(6.23)

The optimal solution is using the rent option until \( \frac{b_j}{r_j} \) if energy is depleted earlier than \( \frac{b_j}{r_j} \). Otherwise, the buy option should be chosen with one time payment \( b_j \) at time 0.

6.4.1 Deterministic Online Algorithm

To design an online algorithm that can achieve a close performance to optimal, we first investigate how close performance a deterministic online algorithm can yield. A deterministic approach is mainly operated by a predetermined parameter when making decision of ON/OFF scheduling. In a deterministic online algorithm (DOA), SBS \( j \) is turned OFF at a predetermined time \( t_j \), \( 0 \leq t_j \leq T \). This flowchart in Fig. 6.2 shows the structure of Algorithm 1 where the OFF time is determined at the beginning of the period. From time 0 to \( t_j \), the rent option is used, and the cost
START a new ON/OFF scheduling period

BSs and UEs exchange the network information.
SBS \( j \) determines the OFF time \( t_j \).

Is now in the scheduling period?
( check \( t \leq T \) )

Has SBS \( j \) enough energy?
( check whether satisfying \( (9) \) )
Or
Is now the predetermined OFF time?
( check \( t = t_j \) )

SBS \( j \) stays turned ON.

SBS \( j \) is turned OFF.

END of the period

Figure 6.2: Flowchart of Algorithm 1.

increases along with the rental cost \( r_j \) per time. Then, at time \( t_j \), the buy option is purchased for the one time cost \( b_j \).

DOA can be shown as Algorithm 6.4. The competitive ratio \( \kappa \) of DOA is given by

\[
\frac{\beta_{\text{DOA}}(u_j)}{\beta_{\text{OPT}}(u_j)} = \begin{cases} \frac{r_j u_j}{\min(r_j u_j, b_j)}, & 0 \leq u_j \leq t_j, \\ \frac{r_j t_j + b_j}{\min(r_j t_j, b_j)}, & t_j \leq u_j \leq T, \end{cases}
\]

where \( \beta_{\text{DOA}} \) is the cost of DOA.

We want to minimize \( \kappa \) subject to \( \beta_{\text{DOA}}(u_j) \leq \kappa \beta_{\text{OPT}}(u_j) \) for every \( u_j \) from 0 to \( T \). Therefore, when \( u_j = t_j = b_j/r_j \), the competitive ratio becomes 2 known as the best possible competitive ratio of a deterministic, online algorithm [92].

### 6.4.2 Randomized Online Algorithm

To handle uncertainty, a rent or buy decision will be made by using a randomized online algorithm (ROA) by means of a probability distribution for ON/OFF scheduling designed to solve our cost-minimization problem. For instance, it is known that, when a randomized approach is used to address a ski rental problem, it is possible to achieve a lower competitive ratio of \( e/e - 1 \) [91, 92], while DOA achieves the competitive ratio of 2.

To develop an ROA for our problem, a competitive analysis analogous to the one done in [92] will be followed. For an arbitrary input, ROA computes an output (i.e., the turn OFF time, \( t_j \)) based on a probability distribution. We want
Algorithm 6.4 Deterministic Online Algorithm (DOA)

1: Initialization: SBS $j \in \mathcal{J}$ has a predetermined value $t_j = b_j/r_j$.
2: while $t \leq T$
3: Update $t \leftarrow t + \epsilon$.
4: If ((6.9) is unsatisfied) or ($t = t_j$),
5: then SBS $j$ is turned OFF.
6: else SBS $j$ maintains its ON state.
7: end while
8: At $t = T$, update $P_j^{\text{op}}, P_j^{\text{on}}, \forall j \in \mathcal{J}$, and user association.

To design an ROA that satisfies $\mathbb{E}[F_j(t_j)] < \kappa \beta_{\text{OPT}}(u_j)$ where $\mathbb{E}[F_j(t_j)]$ is the expected cost of the problem (6.22) redefined by $F_j(t_j) = \{ r_ju_j, \frac{r_j}{r_j + b_j}, \text{if } u_j < t_j, \text{if } u_j \geq t_j \}$, provided that unknown time of energy depletion is given by $u_j$. This will be adequate for our problem in that the input sequence is the unknown and uncertain energy arrivals at a given SBS. Even though an SBS does not know the input sequence, the use of an ROA will give a solution that can at least achieve the expected cost of $\kappa \beta_{\text{OPT}}$.

In this section, when the rental price $r_j$ and the buying price $b_j$ are values related to the cost of using an SBS and the MBS, respectively, we will compute the expected cost of ROA. At time $t_j$, the state of the SBS can be either ON or OFF with probability distribution $p_j^{\text{on}}(t_j)$ or $p_j^{\text{off}}(t_j) = 1 - p_j^{\text{on}}(t_j)$. When an SBS decides to turn OFF at $t_j$, we have

$$\mathbb{E}[F_j(t_j)] = \int_0^{u_j} (r_j t_j + b_j) p_j^{\text{off}}(t_j) dt + \int_{u_j}^T r_j u_j p_j^{\text{off}}(t_j) dt, \quad (6.25)$$

where $p_j^{\text{off}}(t_j)$ is the first-order derivative of $p_j^{\text{on}}(t_j)$. Then, from $\frac{d}{du_j} \mathbb{E}[F_j(t_j)] = R_j(u_j)$, the rate of increase of the cost will be expressed by

$$R_j(u_j) = r_j p_j^{\text{on}}(u_j) + r_j u_j p_j^{\text{on}}(u_j) + (r_j u_j + b_j) p_j^{\text{off}}(u_j),$$

where $p_j^{\text{on}} = -p_j^{\text{off}}$. To find an upper bound on $F_j(t_j)$, we focus on the case in which the expected cost is at its largest value. Naturally, this is the same as finding the worst case in the online ski rental problem which corresponds to the case in which the individual buys the skis on one day, but is unable to use them in the next day. In our model, this corresponds to the case in which the SBS pays for the MBS resources at a price $b_j$ at $u_j$ due to the uncertainty of energy. However, at $u_j = t_j$, the SBS does not need to turn OFF if new energy arrives suddenly at that moment. In this worst case, the cost-increasing rate $R_j(u_j)$ becomes

$$R_j(t_j) = r_j p_j^{\text{on}}(t_j) + r_j t_j p_j^{\text{on}}(t_j) + (r_j t_j + b_j) p_j^{\text{off}}(t_j) = r_j p_j^{\text{on}}(t_j) - b_j p_j^{\text{off}}(t_j).$$

By using the relationship $\mathbb{E}[F_j(t_j)] < \kappa \beta_{\text{OPT}}$, the cost-increasing rate of $\mathbb{E}[F_j(t_j)]$ cannot be larger than the cost-increasing rate of $\kappa \beta_{\text{OPT}}$. The cost-increasing rate of $\beta_{\text{OPT}}$ with respect to $u_j$ can be readily derived by choosing the rent or buy option that yields smaller cost. Now, we divide the range of $u_j, t_j$ into two cases.

First, if $0 < u_j < b_j/r_j$ and $0 < t_j < b_j/r_j$, then the optimal cost-increasing rate is $r_j$ which means that an SBS should be turned ON during $t_j$. Thus, the cost-increasing rate of ROA cannot be lower than $\kappa$ times the optimal cost-increasing rate, we have

$$r_j \kappa = r_j p_j^{\text{on}}(t_j) - b_j p_j^{\text{on}}(t_j).$$

Since this is a first-order linear ordinary differential equation, the solution $p_j^{\text{on}}(t_j)$ is given by:

$$p_j^{\text{on}}(t_j) = ce^{\frac{r_j}{b_j}t_j} + \kappa, \quad (6.26)$$
Algorithm 6.5 Randomized Online Algorithm (ROA)

1: Initialization: SBS \( j \in \mathcal{J} \) determines \( r_j \) and \( b_j \).
2: Find \( t_j \) s.t. \( p_j^{\text{off}}(t_j) = \mu_j, \mu_j \sim U(0, 1), \forall j \in \mathcal{J} \).
3: \textbf{while} \( t \leq T \)
4: \quad Update \( t \leftarrow t + \epsilon \).
5: \quad \textbf{if} \((6.9) \) is unsatisfied or \((t = t_j)\),
6: \quad \qquad then SBS \( j \) is turned OFF.
7: \quad \textbf{else} SBS \( j \) maintains its ON state.
8: \textbf{end while}
9: \textbf{At} \( t = T \), update \( P_j^{\text{op}}, P_j^{\text{on}}, \forall j \in \mathcal{J} \), and user association.

where \( c \) is a constant that can be found by using two boundary conditions. If an SBS starts with the ON state, then \( p_j^{\text{on}}(0) = \kappa + c = 1 \), and then \( c = 1 - \kappa \).

Second, if \( b_j/r_j < u_j \) and \( b_j/r_j < t_j \), then using the MBS is the optimal choice. In this case, an SBS should buy the MBS resource before \( b_j/r_j \). Thus, the SBS should remain in the OFF state at \( b_j/r_j \). This fact leads us to find
\[
p_j^{\text{on}}(b_j/r_j) = (1 - \kappa) e + \kappa = 0, \quad \text{and we find } \kappa = \frac{e}{e - 1}.
\]
Therefore, we have the ON probability \( p_j^{\text{on}}(t_j) = \frac{e}{e - 1} \).

Remark 1. At \( t_j \), SBS \( j \) will turn OFF according to the following probability distribution,
\[
p_j^{\text{off}}(t_j) = \begin{cases} \frac{e}{e - 1}, & 0 \leq t_j \leq \frac{b_j}{r_j}, \\ 1, & \frac{b_j}{r_j} \leq t_j \leq T. \end{cases}
\]

The proposed online ski rental algorithm is summarized in Algorithm 6.5. From (6.27), we observe the tradeoff between rent and buy. As mentioned, the rental price is a cost related to using an SBS while the buying price reflects the cost of using the MBS. For example, the rental price is reduced if using an SBS yields lower delay cost, or the power consumption of an SBS is reduced. Also, the buying price increases if the delay from using the MBS increases, or the power consumption of the MBS increases. Therefore, if \( r_j \) is low and \( b_j \) is high, then it implies that using SBS will reap benefits in terms of delay cost or power consumption, so the rent time becomes longer. In contrast, the rent time becomes shorter if \( r_j \) is high and \( b_j \) is low. The short rent time means an SBS turns OFF early because buying the MBS resource would be more beneficial than using the SBS resource with the rent price. Each SBS will now run Algorithm 6.5 and decide at time \( t = 0 \) when to turn OFF, without knowing any information on energy arrivals, by using the distribution in (6.27). From (6.27), we can observe that the OFF time can be adjusted by changing the value of \( T \). For example, if \( b_j/r_j \) increases by having a longer period of \( T \), the ON time can be extended, so it can prevent the frequent ON/OFF switching. Also, it can be helpful to reduce the frequent handovers.

When using the Algorithm 6.5, we can verify that the expected competitive ratio of ROA is \( \frac{e}{e - 1} \) if \( r_j T \geq b_j \) is satisfied. When the rental option is chosen during the whole period \( T \), the total cost is \( r_j T \). If the total cost is smaller than selecting the buy option such that \( r_j T < b_j \), then this leads to a special case. For such a case, since the optimal solution is always choosing the rental option, the SBS is not turned OFF until the energy is exhausted. Therefore, to find a solution of our interest, we should consider the case in which \( r_j T \geq b_j \).

Then, to show the expected competitive ratio, we calculate the expected cost of ROA. First, let us consider when \( 0 \leq u_j < b_j/r_j \) and \( b_j/r_j < T \). By using (6.25), the expected cost is
\[
\mathbb{E}[F_j(t_j)] = \int_{0}^{u_j} (r_j t_j + b_j) p_j^{\text{off}}(t_j) dt_j + \int_{u_j}^{b_j} r_j t_j u_j p_j^{\text{off}}(t_j) dt_j + \int_{b_j}^{T} r_j u_j p_j^{\text{off}}(t_j) dt_j = \frac{r_j u_j e}{e - 1}, \quad (6.28)
\]
where \( p^\text{off}_j(t_j) = \left\{ \begin{array}{ll} \frac{r_j}{b_j} t_j, & 0 \leq t_j < b_j / r_j, \\ 0, & b_j / r_j \leq t_j \leq T. \end{array} \right. \) The third integration in (6.28) becomes zero since \( p^\text{off}_j(t_j) = 0 \) in \( b_j / r_j \leq t_j \leq T. \)

Second, by letting \( b_j / r_j \leq u_j < T \), we have the expected cost shown as

\[
\mathbb{E}[F_j(t_j)] = \int_0^{b_j} (r_j t_j + b_j) p^\text{off}_j(t_j) dt_j + \int_{b_j}^{u_j} (r_j t_j + b_j) p^\text{off}_j(t_j) dt_j + \int_{u_j}^{T} r_j u_j p^\text{off}_j(t_j) dt_j = \frac{b_j c e}{e-1}.
\]

The second and third terms in (6.29) become zero since \( p^\text{off}_j(t_j) = 0 \) in \( b_j / r_j \leq t_j \leq T. \) By using Definition 1 and the optimal cost given by (6.23), the expected competitive ratio of ROA is \( \kappa = e / (e - 1) \). As a result, for an arbitrary energy arrival, an ROA provides the OFF time of SBS that can have the expected cost of \( e / (e - 1) \) times of the minimum cost of the problem (6.22). Also, while the ROA has the optimal competitive ratio, the solutions found by the online algorithms are suboptimal as shown in the definition of competitive ratio [91, 139]. In fact, given uncertainty of energy harvesting, it is challenging to find the optimal solution of problems.

Then, we can derive the average OFF time period of each SBS when the ROA is used to solve problem (6.21) in the worst case that yields the optimal competitive ratio.

**Theorem 9.** The expected OFF time period of the SBS is \( T - \frac{1}{e-1} \frac{b_j}{r_j} \).

**Proof.** SBS \( j \) is turned OFF at time \( t_j \), so the OFF time period becomes \( T - t_j \). Therefore, the expected OFF time period within period \( T \) is given by

\[
\mathbb{E}[(T - t_j) p^\text{off}_j(t_j) dt_j] = \int_0^{b_j} (T - t_j) \frac{b_j}{r_j} e^{-r_j t_j / b_j} dt_j = \frac{b_j}{e-1}.
\]

In the classical ski-rental problem, the skiing period is not determined by \( T \); thus, the average buying time period cannot be derived. However, in our problem, by using a given period \( T \), the average OFF time period can be derived. The result shows how \( b_j \) and \( r_j \) affect the OFF time period. From the result, if the cost of using the MBS, \( b_j \), becomes inexpensive, the OFF time period is longer. Also, if the cost of using SBS, \( r_j \), is decreasing, then the OFF time is reduced, and the SBSs can be turned ON for a longer time.

Next, we discuss the case that the ROA solves the original problem (6.14). Due to the difficulty of theoretical analysis in problem (6.14), we numerically evaluate the empirical competitive ratio of the ROA with respect to the problem (6.14) through simulations. Furthermore, we carry out simulations to evaluate the OFF time when the ROA is used to solve problem (6.14) in Section 6.5.

### 6.5 Simulation Results and Analysis

For our simulations, the SBSs and UEs are randomly distributed in a \( 0.5 \times 0.5 \text{ km}^2 \) area with one MBS located at the center of the area as shown in Fig 6.3. Statistical results are averaged over a large number of independent simulation runs during time period \( 2T \) with the parameters in Table 1. Simulations during \( 2T \) allow a clear observation of the impact of the unused energy in the first period which can be exploited in the next period. In the simulation, all values are updated with the time resolution of \( \epsilon = 0.1 \text{ sec} \). Without loss of generality, during \( T = 10 \text{ sec} \), we assume that energy arrivals per second follow a Poisson process in which energy arrival rate is \( 20 \), and each arrived energy is \( 0.2 \text{ J} \); for example, it can model a 4 W solar panel or wind generation having power density of 4 W/m\(^2\) [140]. Also it is assumed that initially stored energy of SBS \( j \) is set to \( E_j(0) = 60 \text{ J} \) where the maximum capacity of ESS is \( E_{\text{max}} = 100 \text{ J} \). We use \( q = 0.9 \) and \( K = 10^8 \text{ bits} \). We compare our online ski rental approach ROA and DOA to the baseline approach that turns all SBSs OFF at a certain, pre-determined time \( t_j \).
Table 6.1: Simulation parameters

<table>
<thead>
<tr>
<th>Notation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F_0^{op}$, $F_j^{op}$</td>
<td>20 W, 10 W</td>
</tr>
<tr>
<td>$F_0^{rx}$, $F_j^{rx}$</td>
<td>33 dBm, 23 dBm</td>
</tr>
<tr>
<td>$B_s$, $B_m$</td>
<td>10 MHz, 10 MHz</td>
</tr>
<tr>
<td>$M_s$, $M_m$</td>
<td>10 users, 50 users</td>
</tr>
<tr>
<td>$\rho^2$</td>
<td>-104 dBm</td>
</tr>
<tr>
<td>carrier frequencies</td>
<td>2.1 GHz bands</td>
</tr>
</tbody>
</table>

Figure 6.3: Snapshot example of network resulting from the proposed ROA approach.

Fig. 6.3 shows a snapshot example for 15 SBSs, and 30 UEs at $t = 2$ when ROA is used. In Fig. 6.3, 9 SBSs are turned ON while 6 SBSs are turned OFF. Here, user association is shown as dotted lines between ON SBSs and UEs. From the beginning, four OFF SBSs out of the 6 OFF SBSs initially stay in the OFF state since they do not have any associated UE as shown in Fig. 6.3. We can observe that the other two SBSs are turned OFF by the ROA scheduling since the UEs of two OFF SBSs are located near the MBS. In contrast, most of the ON SBSs are located far from the MBS. In Fig. 6.3, as UEs in $I_j(0)$ are located closer to the MBS, the delay cost of using the MBS, $\phi_0$, decreases. Therefore, the buy price in (6.13) becomes lower. Thus, as the use of the MBS becomes inexpensive, the SBS tends to buy the MBS resource earlier. Also, as the UEs are located farther from any given SBS $j$, the delay cost of using this SBS, $\phi_j(0)$, will increase. Thus, the rental price in (6.10) becomes higher. Since the use of the SBS becomes more expensive, the SBS will buy the MBS resource earlier.

Fig. 6.4 shows, jointly, the total energy consumption of SBSs and the average network delay per SBS, for various numbers of SBSs with 15 UEs, $E_j(0) = 30$ J, $\alpha_D = 0.05$, $\alpha_P = 0.0001$, and $\alpha_B = 0.05$. From Fig. 6.4, we can see that, for all algorithms, as the network size increases, the delay per SBS will decrease, but the total energy consumption will increase. This is due to the fact that having more SBSs turned ON will enable the network to service users more efficiently, however, this comes with an increase in energy consumption. From Fig. 6.4, we can clearly see that ROA reduces both the delay and the energy consumption as compared to the baseline. It is because ROA results the different turned-OFF time of SBSs while all SBSs are turned OFF at the same designated time in the baseline. Thus, it is possible to mitigate interference and enhance network performance when ROA is used. This performance
advantage, reaches up to 20.6% reduction in the delay relative to the baseline $t_j = 7$ for a network with 4 SBSs and 15.6% reduction in energy consumption relative to the baseline for a network with 8 SBSs. Finally, compared to the DOA scheme, Fig. 6.4 shows that ROA will reduce the delay of up to 8.4% (for 4 SBSs) and the energy by up to 11.4% (for 8 SBSs).

In Fig. 6.5, we show the total cost of the network as the network size varies for 30 UEs, $\alpha_D = 0.05$, $\alpha_P = 0.05$, and $\alpha_B = 0.05$. From Fig. 6.5, we can first see that the overall cost of the network given by (6.14) will increase as the number of SBSs increases. This is mainly due to the fact that increasing the number of SBSs will increase the overall power consumption of the network. Also, the sum of delay of SBSs increases along with the number of SBSs in the network. Fig. 6.5 shows that the cost increase of the proposed ROA is much slower than the increase of the DOA and the baseline approach. This demonstrates the effectiveness of the proposed approach in maintaining a low network cost. In particular, Fig. 6.5 shows that, at all network sizes, the proposed online ski rental approach yields reduction in the overall cost of the network. This performance advantage of ROA reaches up to 69.9% reduction of the average cost for 8 SBSs compared to the baseline with $t_j = 7$.

In Fig. 6.6, the total cost of the network is shown when the number of UEs varies for a network with 6 SBSs, $\alpha_D = 0.05$, $\alpha_P = 0.05$, and $\alpha_B = 0.05$. Fig. 6.6 shows that the total cost of the network increases along with the number of UEs. This is because of the fact that increasing the number of UEs will naturally lead to a higher network delay. Nonetheless, we can clearly see that the cost increase of the proposed ROA is slower than that of the DOA and the baseline approach. This shows that the increase of the overall cost is limited by using the proposed ROA. Fig. 6.6 shows that the performance advantage of ROA can yield a reduction of up to 65.4% of the average cost for 40 UEs compared to the baseline $t_j = 7$.

Fig. 6.7 shows the empirical competitive ratio for a network consisting of 3 SBSs and 15 UEs with $\alpha_D = 0.05$, $\alpha_P = 0.05$, and $\alpha_B = 0.05$. To compute empirical competitive ratio, the total cost of the solution resulting from the ROA is divided by the total cost of the offline optimal solution. The optimal cost of each network realization is found by running exhaustive search where all possible OFF times of SBSs are computed. Since the time complexity of the exhaustive search is $O\left(\left(\frac{T}{\epsilon}\right)^{4}\right)$, we reduce the time resolution to $\epsilon = 0.2$ sec and run the simulation for one period.
Figure 6.5: Comparison of the total network cost when using ROA, DOA, and a baseline.

T. We can see that, in 50% of all iterations, the ROA can yield a total cost that is $1.36$ times that of the offline optimal cost. Also, over a total of 800 simulation runs, the empirical competitive ratio in the worst case is shown to be of $1.86$. Thus, the results show that ROA can effectively choose the OFF time in an online manner.

In Fig. 6.8, the average ON time per SBS within time period $T$ is shown for different transmit powers of an SBS and the MBS with 6 SBSs, 16 UEs, $\alpha_D = 0.05$, $\alpha_P = 0$, and $\alpha_B = 0.05$. We compare three different values for the transmit power of an SBS, $P_{tx}^j$: 22, 23, and 26 dBm. If an SBS uses a high $P_{tx}^j$, then the rent price becomes smaller. As the use of the SBS resource becomes less expensive, the SBS tends to maintain the ON state. This, in turn, results in a longer ON time as shown in Fig. 6.8. For example, the average ON time increase by $16.9\%$ if $P_{tx}^j$ increases from 22 dBm to 26 dBm when the MBS uses the transmit power of 37 dBm. Moreover, if the MBS uses a high $P_{tx}^0$, then the buy price becomes smaller. As the cost of using the MBS becomes lower, the SBS tends to use the MBS resource. For example, the average ON time per SBS is reduced by $19.2\%$ if $P_{tx}^0$ increases from 33 dBm to 37 dBm when the transmit power of 22 dBm is used by an SBS in a network.

In Fig. 6.9, we show the total number of ON/OFF operations within time period $T$ for 16 UEs, $\alpha_D = 0.05$, $\alpha_P = 0.05$, and $\alpha_B = 0.05$. Here, we consider another baseline approach that turns an SBS ON if and only if the percentage of charged energy in storage is greater than a threshold $K$. For example, we set $K = 40$ or 50 such that an SBS maintains its ESS half-charged. We first present two baselines in which an SBS is turned ON if $K = 40$ and $K = 50$, respectively. The ROA and DOA clearly yield a lower number of SBS ON/OFF switchings whereas the baseline ($K = 40$ or 50) turns SBSs ON and OFF more frequently. This is mainly due to the fact that the algorithm based on the stored energy will turn ON SBSs that have more than a certain predetermined level of energy. However, the ROA and DOA switch SBSs OFF only once in period $T$. The baseline ($t_j = 7$) also shows the similar number of ON/OFF switchings compared to DOA. Hence, Fig. 6.9 shows that the performance advantage of ROA reaches up to $97.9\%$ of reduction in the number of ON/OFF switchings when compared to the baseline ($K = 50$) in the network consisted of 8 SBSs.

In Fig. 6.10, we show the percentage of unused SBSs for different network sizes. We compare three different values for $P_{tx}^j$: 22, 23, and 26 dBm for 16 UEs, $\alpha_D = 0.05$, and $\alpha_B = 0.05$ while the transmission power of the MBS is fixed to 33 dBm. We set $\alpha_P = 0$ to observe the changes related to network performance. In Fig. 6.10, the percentage of unused SBS decreases as the transmission power of an SBS increases in the network. When the transmission power
of an SBS become higher, UEs can receive higher SINR value in (6.2) than SNR from the MBS in (6.3). Thus, larger number of UEs is connected to SBSs, so it can reduce the number of unused SBSs. For example, Fig. 6.10 shows that the percentage of unused SBSs is reduced by 33% if the transmission power of an SBS increases from 22 dBm to 26 dBm. Also, as the number of SBSs increases, we observe that a higher fraction of SBSs is not used in the network. This is because, as the number of SBSs increases, higher interference will occur thus reducing the SINR at the UEs. In essence, it leads to more UEs that associate with the MBS thus increasing the number of unused SBSs. Indeed, in Fig. 6.10, we can see that the percentage of unused SBSs increases by 47.2% if the number of SBSs changes from 4 to 8 in the network.

Figs. 6.11 and 6.12 show the average ON time per SBS for the different operational power of an SBS $P_{op}^j$ and the MBS $P_0^{op}$, respectively, with 6 SBSs, 16 UEs, and $\alpha_P = 0.05$. We set $\alpha_D = 0$ to observe the effects from different power consumptions. In Fig. 6.11, we observe that the ON time per SBS becomes shorter if an SBS consumes a higher operational power. This is due to the fact that the cost of using an SBS increases with the power consumption of an SBS. As a result, the rent price becomes higher. This means that choosing the rent option becomes less affordable, thus resulting in a shorter average ON time per SBS. From Fig. 6.11, the average ON time per SBS is shortened by 45% when the operational power of an SBS is changed from 10 W to 16 W when the MBS uses 20 W. Also, in Fig. 6.12, we observe that the ON time per SBS can be prolonged if the MBS consumes high $P_0^{op}$. This can be explained as follows: if $P_0^{op}$ is high, then the buying price (6.13) becomes higher, so the ON time per SBS becomes longer. The simulation result shows that the average ON time increases 2 times if $P_0^{op}$ increases from 20 W to 40 W when an SBS consumes 10 W.

Furthermore, in Figs. 6.11 and 6.12, the average ON time per SBS within time period $T$ is shown for different $\alpha_B$. As $\alpha_B$ becomes larger, a higher buy price will be incurred when an SBS is turned OFF, so the SBS tends to stay in the ON state without buying the MBS resource. This, in turn, results in a longer ON time as shown in Figs. 6.11 and 6.12. Also, the increase in the ON time is proportional to the increase of $\alpha_B$. For example, in Fig. 6.11, the average ON time increases three folds if $\alpha_B$ increases from 0.05 to 0.15 when $P_{op}^j$ is 10 W. The same effect can be seen in Fig. 6.12 where the average ON time is extended three folds if $\alpha_B$ increases from 0.05 to 0.15 when $P_{op}^j$ is 40 W.

In Fig. 6.13, we show the effect of the initial energy levels on the average ON time for 6 SBSs, 16 UEs, $\alpha_D = 0.05$,
α_P = 0.05, and α_B = 0.15. We compare three different values for E_j(0): 20, 40, and 60 J while other parameters related to energy arrival is given equally. As an SBS has high E_j(0), an increase in the average ON time is observed. The result is due to the fact that a high E_j(0) can help an SBS maintain in ON state for a longer period. For instance, the average ON time per SBS increases by 5.1% if E_j(0) increases from 20 J to 60 J when P_{tx}^j is 22 dBm. Furthermore, we observe that the utilization-proportional power consumption of the MBS is reduced when the ON time per SBS becomes longer. This is because the SBSs will offload UEs from the MBS. Clearly, the use of self-powered SBSs can reduce the power consumption of the MBS as shown in the case of E_j^L(0) = 60 J.

In Fig. 6.14, we investigate the effect of using more information about the dynamics of the rental cost on minimizing the total cost. We compare the update rule (6.20) in Theorem 8, DOA, and ROA under an illustrative network example in which the rental price is monotonically decreasing over time. The considered network here consists of 1 SBS, 1 MBS, and 10 UEs, for α_D = 0.05, α_P = 0.0001, and α_B = 0.05. To satisfy (6.19), we set the transmission power P_{tx}^j to 23, 25, 27, and 29 dBm, at the following time instants t = 0, 1, 3, and 5, respectively. Therefore, when P_{tx}^j increases at t = 1, 3, and 5, the delay cost of the SBS can be reduced; thus, the rental price decreases. In this environment, we can observe that the derived update rule in (6.20) can reduce the total cost when it is compared to DOA or ROA. This is due to the fact that by using (6.20), the SBS can use more information on the updated P_{tx}^j to make a better decision as opposed to DOA and ROA which rely solely on only information. The SBS following (6.20) can dynamically update its decisions based on the decreasing rental cost, so it is possible to have a longer ON time than the DOA as shown in Fig. 6.14. For the considered network example, by using (6.20), SBS will not need to buy the MBS resource whereas the DOA uses the SBS resource and also buy the MBS resource. Thus, Theorem 8 results in the smaller total cost compared to the DOA in the example. Also, ROA can reduce the total cost than the DOA since ROA uses the SBS resource for a short period and chooses to buy the MBS resource earlier. Our example illustrates that the ROA yields a lower cost than the DOA but a higher cost than Theorem 8. However, clearly, by using the ROA, the approximation yields a reasonably good solution, which does not require any full information on the dynamic parameters of the system.
Figure 6.8: Average ON time per SBS with respect to the transmit power of an SBS and the MBS during period $T$.

6.6 Summary

In this chapter, we have proposed a novel approach to optimize the ON/OFF schedule of self-powered SBSs. We have formulated the problem that minimizing network operational costs during a period. Also, the problem is approximated as an online ski rental problem which enables the network to operate effectively in the presence of energy harvesting uncertainty. To solve this online problem, we have proposed deterministic and randomized online algorithm that is shown to achieve the optimal competitive ratio for the approximated problem. Indeed, we have shown that by using the proposed ROA, each SBS can autonomously decide on its ON time without knowing any prior information on future energy arrivals. Simulation results have shown that the proposed ROA can achieve an empirical competitive ratio of 1.86, thus showing that ROA can effectively choose the OFF time in an online manner. The results have also shown that both delay and the ON/OFF switching overhead are significantly reduced when one adopts the online ski rental approach.

6.7 Appendix

6.7.1 Proof of Theorem 8

Given definitions of $r_{(v)}$ and $t_{(v)}$, we determine $\bar{t}$ such that the accumulated cost up to time $\bar{t}$ equals to the cost of using the MBS $b_j$; thus, $\bar{t}$ satisfies

$$r_{(1)}t_{(1)} + r_{(2)}(t_{(2)} - t_{(1)}) + \cdots + r_{(v-1)}(t_{(v-1)} - t_{(v-2)}) + r_{(v)}(\bar{t} - t_{(v-1)}) = b_j.$$  \hspace{1cm} (6.30)
However, the offline optimal cost is given by
\[ \beta_{\text{OPT}}(u) = \sum_{v'=1}^{u-1} t_{(v')}(r_{(v')} - r_{(v'+1)}) + r_{(v)}u. \]  

The optimal cost \( \beta_{\text{OPT}}(u) \) can be calculated by assuming an offline scenario where energy arrival information over the entire period is given. Thus, the amount of stored energy at each moment becomes known information. In this case, we can find that \( \beta_{\text{OPT}}(u) \) is the same as (6.31). Also, if \( \bar{t} \leq u \), then the total cost is given by
\[ \beta_{\text{ALG}}(u) = \sum_{v'=1}^{u-1} t_{(v')}(r_{(v')} - r_{(v'+1)}) + r_{(v)}\bar{t} + b_j. \]

However, the offline optimal cost is given by \( \beta_{\text{OPT}}(u) = b_j \). Therefore, the worst-case competitive ratio given by (6.18) becomes 2 in the case of \( \bar{t} \leq u \) since \( \beta_{\text{ALG}}(u) \) can be two times greater than \( \beta_{\text{OPT}}(u) \) due to (6.30).
Figure 6.10: Fraction of unused SBSs in the network for the different transmit power.

Figure 6.11: Average ON time per SBS for different operational power of an SBS
Figure 6.12: Average ON time per SBS for different operational power of the MBS

Figure 6.13: Average ON time per SBS for different harvested energy.
Figure 6.14: Total cost and ON time of an SBS when comparing Theorem 8, ROA, and DOA.
Chapter 7

Energy-Efficient Networking with Reconfigurable Intelligent Surfaces

7.1 Background, Related Works, and Contributions

The RISs, mounted on walls and buildings, have emerged as an effective approach to enhance the ever increasing need for capacity [103]. The advantage using an RIS as a reflector that assists existing cellular BSs stems from the ability of RISs to provide near-field communications while having a very low carbon footprint relative to conventional BSs [141]. If properly deployed in an urban environment, RISs can provide nearly line-of-sight (LOS) communication channels [142]. Indeed, the users in RIS environment will be able to maintain reliable wireless connections and low-latency data transmission compared to conventional antenna-array systems [143]. However, to reap the benefits of RISs, architectural and operational challenges must be addressed [144–149].

To develop RISs, the authors in [144] study the use of a field programmable gate array (FPGA) made with tunable metasurfaces electrically controllable via software. When using an RIS in wireless communications, radio resource allocation to optimize the network performance is a prime concern. For instance, the work in [145] studies an RIS-assisted downlink system design that minimizes the BS transmit power by optimizing the continuous transmit beamforming of the BS and the discrete phase shifter of the RIS. Moreover, the authors in [146] develop a joint active and passive beamforming design to maximize the received signal power of RIS users. Also, the authors in [147] investigate the problem of maximizing the downlink capacity to design the optimal RIS phase shift by exploiting statistical channel state information. In [148], a passive beamformer is proposed to achieve an asymptotic optimal performance by controlling the incident wave properties while considering a limited RIS control link and practical reflection coefficients. Moreover, the work in [149] studies the energy efficiency maximization problem in an RIS environment when all involved channels are perfectly known at BS to use zero forcing transmission.

In all of these existing RIS works [144–149], it is generally assumed that information on the environment such as wireless channels and power consumption is completely known. However, in practice, the wireless channel gains change in a fading environment, and the wireless channel can be uncertain if the RIS configuration is dynamically updated. Hence, the BS cannot know the exact channel gain. Indeed, it is challenging for a cellular BS to perform precoding with incomplete channel information. Thus, there exists an inherent uncertainty stemming from the unknown RIS configuration and the spatio-temporal dynamics of the channel. Further, most of the existing works [144–149] typically assume that an RIS system is operated by using a power grid. However, in practice, the use of energy harvesting at an RIS can be necessary to reduce the reliance on the conventional power grid and enable the vision of green communications. Moreover, since no amplifier is used in an RIS, it will consume very low energy, and, therefore, it can be suitable
for an RIS to adopt energy harvesting technologies. Consequently, unlike the existing literature [144–149] which assumes full knowledge about the network environment, uses power grid as energy source, and relies on model-based optimization techniques, our goal is to design a deep reinforcement learning (RL) approach to make a decision using on-the-fly information on the cellular networks, under channel uncertainty and energy harvesting, while maximizing the average energy efficiency.

The main contribution of this chapter is a novel framework for RIS-assisted cellular communications using energy harvesting technologies at the RIS. This framework allows a BS to dynamically adapt to wireless environment in the presence of uncertainty on the wireless channel gains and future available energy of the RIS (acting as a reflector). Therefore, this downlink system can jointly use both the direct and indirect wireless paths between the BS and users. We formulate an optimization problem whose objective is to maximize the average energy efficiency of the BS by properly allocating the downlink transmit power while also properly determining the phase of the RIS elements. To solve this optimization problem, we propose a novel approach based on deep RL by defining the state, action, reward, and policy. In the proposed framework, the BS decides the action about the power allocation, phase shift, and RIS ON/OFF states. Then, the environment nodes including the RIS and users send the feedback information consisting of the state about the wireless channel and energy and reward about energy efficiency. Throughout the proposed learning process, the policy of the BS can select the best possible actions depending on different states. Finally, a case study is conducted to analyze the performance of the studied RIS-assisted downlink system by asymptotically deriving an upper bound of the energy efficiency. Simulation results show that the energy efficiency can increase by up to 24.6% by increasing RIS phase shifter resolution from 3 to 5 bits.

The rest of this chapter is organized as follows. In Section 7.2, the system model is presented. In Section 7.3.1, we formulate the proposed problem. Section 7.3.2 presents our approach to solve the problem, and Section 7.3.3 includes a case study for a performance analysis. Simulation results are analyzed in Section 7.4 while conclusions are drawn in Section 7.5.
7.2 System Model

7.2.1 Wireless Network Model

We consider the downlink of a wireless network with a single BS with $M$ antennas. The BS is assisted by an RIS reflecting surface that serves a set of $K$ single-antenna user devices, as shown in Fig. 7.2. In this system model, the RIS is a reflecting surface that includes an antenna array with $N$ passive phase shifters used to reflect the received signal while changing the phase of the signal. Fig. 7.1 illustrates our model in which one of the sides of a building is equipped with an RIS having a large number of antennas, i.e., $N > M$.

A user can receive signal from the BS via direct and reflected wireless links, respectively. The direct path between the BS and the user is a non-LOS (NLOS) channel. The channel of the direct link between the BS and user $k$ is given by

$$h_B^k = \left[ h_{1k}^B, \dotsc, h_{Mk}^B \right]^T \in \mathbb{C}^{1 \times M},$$

where $g_{Bk} = d_B^{-\alpha_{NL} / 2}$ is the path loss between the BS and user $k$ at a distance $d_B$ and path loss exponent $\alpha_{NL}$, and $h_{mk}^B$ is the small-scale fading between BS antenna $m$ and user $k$. Therefore, the channel between the BS and the users will be:

$$H_{BU} = \left[ (h_1^B)^T, \dotsc, (h_K^B)^T \right]^T \in \mathbb{C}^{K \times M}.$$

The RIS-reflected path includes two links: an NLOS link between the BS and the RIS and an LOS link between the RIS and the users. The NLOS channel between the BS and the RIS system is given by

$$H_{BL} = g_{BL} \{ h_{nm} \}_{n \in [1,N], m \in [1,M]} \in \mathbb{C}^{N \times M},$$

where $g_{BL}$ is the path loss between the BS and the RIS system, and $h_{nm}$ is the complex Gaussian random variable, $\mathcal{CN}(0,1)$. Therefore, $H_{BL}$ is a matrix where $g_{BL}h_{nm}$ is the element at row $n$ and column $m$. When the BS transmits a signal to the RIS through $H_{BL}$, the RIS is used to reflect the signal to the users with the following vector of phase shifts:

$$\Phi = [\phi_1, \dotsc, \phi_N] \in \mathbb{C}^{1 \times N}.$$

Each RIS element $n$ can select a phase shift value $\phi_n$ from the feasible set of phase shifting values:

$$\phi_n \in \mathcal{F}_{\text{RIS}} \triangleq \left\{ \exp \left( \frac{j2\pi m}{2^b} \right) \right\}^{2^b-1}_{m=0},$$

where $b$ is the resolution of the RIS element’s phase shifter [149]. Also, the ON or OFF state of RIS element $n$ is denoted by $\sigma_n$ and defined as follows:

$$\sigma_n = \begin{cases} 1, & \text{if RIS element } n \text{ is turned ON,} \\ 0, & \text{otherwise.} \end{cases} \quad (7.1)$$

Then, the vector of the ON/OFF states of the RIS elements is defined as $\sigma = [\sigma_1, \dotsc, \sigma_N] \in \mathbb{R}^{1 \times N}$. If an RIS element $n$ is turned OFF, the signal from the BS will not be reflected by this RIS element $n$. Therefore, considering the ON/OFF states of the RIS elements, the phase shifting matrix can be defined as:

$$\Lambda = \text{diag}(\Phi \odot \sigma) \in \mathbb{C}^{N \times N},$$

where $\text{diag}(\cdot)$ is a block-diagonal matrix, and notation $\odot$ is an element-wise vector multiplication.

When the BS’s signals are reflected to the users, the channel between the RIS and user $k$ becomes

$$h_k^L = [g_{1k}h_{1k}^L, \dotsc, g_{Nk}h_{Nk}^L] \in \mathbb{C}^{1 \times N},$$
where \( g_{nk} = 1/\sqrt{4\pi d_{nk}^2} \) is a free space path loss attenuation between RIS element \( n \) and user \( k \), and \( h_{nk}^L \) is the LOS channel state between RIS element \( n \) and user \( k \). In Fig. 7.1, the users are located in front of the RIS within a two-dimensional space in the \( xy \)-plane at \( z = 0 \) in Cartesian coordinates. We define the location of user \( k \in [1, K] \) as \( (x_k, y_k, z_k) \). The RIS element \( n \in [1, N] \) is located at \( (x_n, y_n, 0) \). Then, the channels between the RIS system and user \( k \) are assumed to be LOS links as in [143], and, thus, channel \( h_{nk}^L \) becomes:

\[
h_{nk}^L = \exp \left( -j \frac{2\pi d_{nk}}{\lambda} \right), \quad \forall n \in [1, N],
\]

where \( \lambda \) is the signal wavelength, and \( d_{nk} \) is the distance between user \( k \) and RIS element \( n \) given by \( d_{nk} = \sqrt{(x_k - x_n)^2 + (y_k - y_n)^2 + z_k^2} \). Therefore, the channel between the RIS and its \( K \) users is given by

\[
H_{LU} = [(h_1^L)^T, \ldots, (h_K^L)^T]^T \in \mathbb{C}^{K \times N}.
\]

Thus, through the channels \( H_{BL} \) and \( H_{LU} \), the reflected path is used to transmit a signal from the BS to the users.

Given the wireless channel model, the downlink signal received by user \( k \) is expressed by:

\[
y_k = (h_k^L \Lambda H_{BL} + h_k^B) x + n_k,
\]

where \( n_k \sim \mathcal{CN}(0, \sigma_n^2) \) is the zero-mean complex white Gaussian noise with variance \( \sigma_n^2 \). The BS transmits the signal

\[
x = \sum_{k=1}^K \sqrt{p_k} f_k^H s_k \in \mathbb{C}^{M \times 1},
\]

where \( p_k \) is the transmit power of user \( k \)'s signal, \( f_k^H \) is the precoding vector for user \( k \), and \( s_k \) is the unit power information symbol. The sum of the transmit power of all users is smaller than the BS maximum transmit power \( P_{\text{max}} \). The transmit power for the users are captured by the vector \( p = [p_1, \ldots, p_K] \). At the BS, precoding \( f_k \) is applied to obtain the transmitted signal \( x_k \). Particularly, each user selects a precoding vector from a pre-defined codebook that is known to both the users and the BS [150]. Then, the user sends the precoding matrix indicator (PMI) of the selected precoding vector to the BS. When selecting a precoding vector, the users decide the precoding vector that is most suitable to maximize their data rate after measuring their channel state [150]. Since \( h_k \triangleq h_k^L \Lambda H_{BL} + h_k^B \in \mathbb{C}^{1 \times M} \) is the effective channel of user \( k \), user \( k \) selects its channel direction \( \hat{h}_k \) to the closest \( w_i \), according to

\[
\hat{h}_k = \arg \max_{w_n \in \mathcal{C}_{\text{BS}}} |h_k w_n^H|,
\]

where \( w_n \) is a precoding vector. The BS uses a discrete Fourier transform (DFT)-based codebook [151] given as:

\[
\mathcal{C}_{\text{BS}} = \left\{ \exp \left( j \frac{2\pi n}{2c} \right) \right\}_{n=0}^{2^c-1},
\]

where \( c \) is the number of feedback bits for PMI. Based on the feedback information from user \( k \), the BS selects the precoding vector \( f_k = \hat{h}_k \), and, therefore, the received SINR at user \( k \) becomes:

\[
\gamma_k = \frac{p_k |h_k \hat{h}_k^H|^2}{\sum_{j \neq k}^K p_j |h_k \hat{h}_j^H|^2 + \sigma_n^2}.
\]

The achievable sum rate of the RIS-assisted multi-user MISO system is given by \( R = \sum_{k=1}^K \log_2(1 + \gamma_k) \).

### 7.2.2 Energy Consumption Model

In our considered system, the RIS reflector is self-powered and relies exclusively on energy harvesting sources. For example, the RIS reflector can be equipped with solar panels to procure energy for its operation. Since the characteristics of the harvested energy can be highly dynamic, we do not make any specific assumption on the energy harvesting
process. Thus, our model can accommodate any type of energy harvesting mechanism. To enhance the overall energy efficiency of the system, the RIS elements can be turned ON or OFF, depending on the network performance and harvested energy status. Therefore, the RIS system is equipped with a battery or energy storage systems (ESS) to manage the intermittent and uncertain energy harvesting process. Also, the RIS system can harvest energy irrespective of the ON or OFF status of its RIS elements. Moreover, when the RIS elements are turned ON, the RIS system can still store the excess of harvested energy if the instantaneous harvested energy is enough to operate the RIS elements.

When the transmission power of the BS for user $k$ at time $t$ is $p_k(t)$, the power consumption for the wireless communication link between the BS and $K$ users becomes $P(t) = \sum_{k=1}^{K} \mu p_k(t)$ where $1/\mu$ is the efficiency of the transmit power amplifier [149]. When $\sigma(t)$ is the vector of the ON/OFF states of the RIS elements at time $t$, the power consumption of the RIS system at time $t$ is modeled as $P_{\text{RIS}}(t) = \sum_{n=1}^{N} \sigma(t) P_b$ where $P_b$ is the power consumption of a phase shifter with $b$-bit resolution. When all RIS elements are configured to use $b$-bit resolution, the power consumption to turn an RIS element ON will be identical.

We consider a time-slotted system with a time slot duration $\Delta$. The energy consumption of the RIS system during time slot $t$ becomes $S(t) = P_{\text{RIS}}(t) \Delta$. When the RIS elements consume the harvested energy, the available amount of energy stored in the ESS at time $t$ is given by

$$E(t) = \min \left( E(t-1) - S(t-1) + \Omega(t-1), E_{\text{max}} \right),$$  

where $E(t-1) \geq 0$ is the stored energy of the RIS system at time $t-1$, $S(t-1)$ is the energy used by the RIS elements, $\Omega(t)$ is the amount of harvested energy at the RIS system, and $E_{\text{max}}$ is the maximum energy storage capacity of the ESS [79, 152]. Since $\Omega(t)$ is randomly generated because the RIS system is unable to know the harvested energy in the future, and, therefore, randomness captures the uncertainty of the energy harvesting process over time. When the RIS reflectors are turned ON by only using the harvested energy, all RIS elements can be turned OFF at time $t$ if $E(t)$ becomes zero at a certain time $t$. In this case, the users will only receive the signal from the BS without any reflected signal from the RIS. Also, if the RIS system does not have enough energy to all of its RIS elements ON at time $t$, i.e., $S(t) > E(t)$, then, a partial set of RIS elements must be turned OFF, and the users’ data rate can change over the ON/OFF configuration of the RIS elements, i.e., $\sigma(t)$.

### 7.3 Machine Learning Framework for Energy-Efficient Networking

Given the defined system model, our goal is to analyze the joint problem for allocating transmit power to the BS antennas and optimizing the operation of the RIS elements by deciding the ON/OFF status and phase shifting. Particularly, since the future energy status of the RIS elements is unpredictable, the ON/OFF states of the RIS elements dynamically change. Also, when the ON/OFF status of the RIS elements is determined depending on the harvested energy at the RIS system, the wireless channel can dynamically change over time. Therefore, there is uncertainty on wireless channels when a self-powered RIS is used. Thus, it is highly challenging for the BS to optimize the transmit power each time the ON/OFF status of the RIS elements is updated. In fact, even if turning ON more RIS elements can improve the data rate, it may have a detrimental effect on future data rates due to the lack of the stored energy at an RIS. To cope with the uncertainty of the harvested energy and channel states, we introduce an artificial intelligence framework that uses machine learning techniques to maximize the energy efficiency of the cellular system while properly managing the harvested energy at the RIS.
### 7.3.1 Problem Formulation

First, we formulate the following optimization problem whose goal is to maximize the energy efficiency of the RIS-assisted communication system:

\[
\begin{align*}
\max_{\sigma(t), \Phi(t), p(t)} & \quad \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T} R(t) \frac{1}{P(t)}, \\
\text{s.t.} & \quad S(t) \leq E(t), \forall t, \\
& \quad \sum_{k=1}^{K} p_k(t) \leq P_{\text{max}}, \forall t.
\end{align*}
\]  

(7.4)

The objective function (7.4) is the average energy efficiency achieved by the decision of the BS. Constraint (7.4a) is an energy causality condition. Constraint (7.4b) means that the sum of transmit power is smaller than or equal to the maximum transmit power. In the formulated problem, the BS is assumed to optimize the transmit power, RIS phase shifts, and ON/OFF states. To determine the transmit power and RIS phase shifts, the BS needs to calculate the objective function (7.4). However, since the channel information required to calculate the SINR in (7.2) is unknown to the BS, the value of (7.4) is not directly known by the BS. In particular, when the users observe the channel through pilot signaling from the BS, the users send channel feedback information to the BS by using the PMI, thus resulting in quantization errors. Therefore, the BS can only know the estimated channel [153]. Due to the uncertainty on the channel between the BS and users, the BS is unable to calculate the data rate function that includes both original channel and estimated channel term. Thus, when the estimation error is unknown to the BS, conventional optimization techniques such as convex and combinatorial programming cannot be applied to solve problem (7.4) [154]. To this end, a deep RL technique can be used to solve problem (7.4) by using a deep neural network function to approximate the relationship between the optimization variables and achieved performance. Next, we propose an RL-based framework used to seek a solution to maximize the time-averaged energy efficiency of the system when the performance metric is not directly accessible for the optimization algorithm running on the BS.

### 7.3.2 Deep Reinforcement Learning Approach

As aforementioned, the BS is unable to directly evaluate the objective function with under channel uncertainty. Also, the wireless resource optimization and RIS ON/OFF scheduling problem in (7.4) involves a large state space, i.e., \(O(2^K)\). Therefore, a closed-form solution does not exist without any knowledge about the state transition probabilities, and an exhaustive search, i.e., a brute-force algorithm, is impractical. Thus, we propose a deep RL framework to seek a solution that maximizes the energy efficiency so that the data rate of the users are improved while consuming a lower energy. A deep RL framework can handle a control problem with a large state space since deep RL uses a deep neural network for approximation of the RL’s action-value function for a RIS system [155]. Beyond being able to maximize the energy efficiency, the key advantage of the deep RL framework is that the BS can learn the performance outcome of the system through a trial-and-error process while updating the weights of a deep neural network [156]. In doing so, the BS can make an immediate decision to allocate the radio resource while knowing only estimated channel without having a knowledge of the exact information on channel.

The proposed framework includes the agent implemented on the BS and the rest of environment nodes including an RIS and the users as shown in Fig. 7.2. To develop an RL framework, we must define the state, action, policy, and reward. For our model, the states consist of the precoding vectors \(\hat{h}_k\) from the users and the energy level of our RIS \(E(t)\). Meanwhile, the actions are the optimization variables in (7.4) such as transmit power \(p(t)\), phase shifting \(\Phi(t)\), and ON/OFF status \(\sigma(t)\). The policy denotes the strategy that maps a state to an action. Therefore, the policy is used to determine the values of the optimization variables. Since the actions are state-dependent, similar actions can yield different outcomes contingent upon the current states.
The goal of RL is to pick the best known action for any given state. To this end, the BS will observe and use feedback. Particularly, in Fig. 7.2, when the BS decides the action, the environment nodes send a feedback to the BS. The feedback message includes the current state and reward. In the proposed framework, the reward returned by the environment is defined as:

$$ r(t) = \sum_{k=1}^{K} R(t)/P(t). $$

(7.5)

When the users send feedback about the data rate $R(t)$, the BS knows its energy consumption $P(t)$. Therefore, the definition of the reward in (7.5) can be readily calculated by the BS. The reward value can be affected by various unknown aspects such as uncertainty on exact channel of the users. Therefore, by using the feedback returned from the uncertain environment, the deep neural network uses the difference between the expected reward and the actual reward, and the BS improves the weights of the deep neural network that shows the expected reward of state-action pairs. In Fig. 7.2, the decision making block can be implemented by using different deep RL methods such as deep Q-network [156]. Thus, throughout the learning process, the BS is able to learn the policy that is used to select the best possible actions depending on different states. As a result, the proposed framework shown in Fig. 7.2 has key benefits. First, the agent running on the BS only need to interact with the environment by exchanging a small size of bits. Also, any prior information on the environment is not required to deploy the proposed framework. Moreover, the proposed framework is compatible with the existing cellular standard in that PMI and reward in (7.5) are calculated by the users and those parameters can be reported to the BS through PUCCH on the control plane.

### 7.3.3 Performance Analysis: Case Study

Due to the uncertainty on the wireless channels and harvested energy of the studied system, the energy efficiency and data rate are not deterministic, and the exact outcomes of an algorithmic solution become mathematically intractable. Therefore, a case study is carried out to examine and analyze the performance of an RIS-assisted downlink system. To this end, we asymptotically analyze the data rate and energy efficiency. This analysis will provide the upper bound of the data rate and energy efficiency that can be asymptotically achieved by using the RL methods.
Theorem 10. For the studied RIS-assisted downlink system, the upper bound of energy efficiency is given by \( \frac{1}{\ln 2} N^2 \) when \( N, K \to \infty \), and \( p_k = P_{\text{max}}/K, \forall k \).

Proof. We derive the upper bound of the given system model’s energy efficiency. The SINR of user \( k \), \( \gamma_k \), has an upper bound given by \( \gamma_k^{\text{SNR}} \triangleq \frac{p_k | h_k^H |^2}{\sigma_k^2} \) that is the SNR of the single user case without inter-user interference. Also, \( \gamma_k^{\text{SNR}} \) increases with \( O(N^2) \) as \( N \to \infty \), as proved in [146]. Therefore, we have the following relationship:

\[
\gamma_k \leq \gamma_k^{\text{SNR}} \sim O(N^2).
\]

From (7.6), an upper bound of the sum data rate can be derived as:

\[
\sum_{k=1}^K \log_2(1 + \gamma_k) \leq \sum_{k=1}^K \log_2(1 + \gamma_k^{\text{SNR}}),
\]

where \( \sum_{k=1}^K \log_2(1 + \gamma_k^{\text{SNR}}) \sim \sum_{k=1}^K O(\log_2(1 + \frac{P_{\text{max}}}{K} N^2)) \). Now, when \( K \to \infty \) and the transmit power is equally allocated to each user, the sum rate can be written as:

\[
\lim_{K \to \infty} \sum_k \log_2(1 + \gamma_k^{\text{SNR}}) \sim O\left(K \log_2\left(1 + \frac{P_{\text{max}}}{K} N^2\right)\right).
\]

Thus, we have the result:

\[
\lim_{K \to \infty} K \log_2\left(1 + \frac{P_{\text{max}} N^2}{K}\right) \overset{(a)}{=} \frac{1}{\ln 2} P_{\text{max}} N^2,
\]

where (a) results from the exponential definition \( e^x = \lim_{n \to \infty} (1 + x/n)^n \). Hence, by dividing the derived upper bound of the sum rate by the total transmit power \( P_{\text{max}} \), the upper bound of EE is given by \( \frac{1}{\ln 2} N^2 \).

In the single user case, the user’s SNR increases with \( O(N^2) \), and, thus, the data rate will asymptotically follow \( O(\log_2(N)) \), as \( N \to \infty \). However, in a multi-user case, we can observe from Theorem 1 that the derived upper bound of multi-user sum rate increases with \( O(N^2) \), as \( N, K \to \infty \).

In a multi-user case, since the upper bound of the sum rate is finite, i.e., \( \frac{1}{\ln 2} P_{\text{max}} N^2 \), the data rate per user approaches zero as \( K \to \infty \). Therefore, it is beneficial to schedule a finite number of users in an RIS-assisted cellular network. From Theorem 1, for a finite \( K \), the upper bound of multi-user sum rate is asymptotically derived as \( O\left(K \log_2\left(1 + \frac{P_{\text{max}}}{K} N^2\right)\right) \sim O(\log_2(N)) \). Therefore, we conclude that the data rate of a single user case asymptotically achieves the upper bound of the multi-users’ sum rate. Next, we evaluate the performance of the proposed framework throughout simulation experiments in Section 7.4.

7.4 Simulation Results

For our simulations, we consider an RIS-assisted downlink system where the distance between the BS and the RIS is 300 m. The width and height of an RIS are 30 m, respectively. When four users are located in front of the RIS, the distance between each user and the RIS surface follows a uniform distribution in range between 0 and 30 m. The BS has 16 antennas while each user has a single antenna. The bandwidth is 10 MHz, and the power spectral density of the noise is -174 dBm/Hz. We use \( 1/\mu = 22.6 \) and \( P_{\text{max}} = 43 \) dBm. Also, the NLOS path loss exponent \( \alpha_{\text{NL}} \) is 3.7. We assume that energy arrivals per second follow a Poisson process with an energy arrival rate of 10. The RIS energy consumption is modeled with \( \Delta = 1 \) and \( P_b \Delta / E_{\text{max}} = 100 \). Finally, deep Q-network method is implemented as the decision making block in Fig. 7.2 where the action space includes the discrete transmit power levels quantized with an interval of 1 W. Thus, the performance of the proposed framework is evaluated in the defined environment.
Figure 7.3: Energy efficiency and sum rate for different number of RIS elements.

Figure 7.4: Energy efficiency and sum rate for different RIS phase shifter’s resolution bits.
In Fig. 7.3, we show the energy efficiency and the sum rate of the users for different numbers of RIS elements. Fig. 7.3 first shows that the sum rate of the users also increases when the number of RIS elements increases. This is due to the fact each user’s data rate increases with respect to the number of RIS elements $N$ as shown in Theorem 1. At the same time, in Fig. 7.3, we can see that the energy efficiency increases when the number of RIS elements increases since the energy efficiency is proportional to the sum rate. From these observations, a larger number of RIS elements can be deployed to maximize the sum rate and energy efficiency of an RIS-assisted downlink system. For instance, if the number of RIS elements $N$ increases from 9 to 25, then the energy efficiency can be improved by 77.3%, and the sum rate can increase up to 2.4 times.

Fig. 7.4 shows the energy efficiency and sum rate for different energy arrival rates when the resolution of RIS elements changes from 3 to 5 bits. We can first see that the energy efficiency and sum rate increase with respect to the resolution bits. As the RIS uses phase shifters with a higher resolution, it is possible to precisely control the RIS beamformer, thus improving the wireless channel gains. For instance, by increasing the resolution from 3 bits to 5 bits, the energy efficiency can increase up to 24.6% when the energy arrival rate is 20. Also, Fig. 7.4 shows that the energy efficiency and sum rate increase when more energy is harvested at an RIS. Since the RIS phase shifters are solely operated by harvested energy, harvesting more energy enable the RIS to turn on additional RIS phase shifters. Therefore, additional harvested energy will reduce the impact of the energy harvesting constraint when determining the ON/OFF states of the RIS elements. For example, if the energy arrival rate increases from 10 to 20 with an RIS resolution of 5 bits, then the energy efficiency and sum rate are improved up to 21.1% and 32.1%, respectively.

Fig. 7.5 shows the cumulative average energy efficiency defined in (7.4) over episodes. If the energy efficiency reaches or is greater than $10^6$, a current episode is assumed to be done. Then, the environment is reset to generate new wireless channels gains and user locations so that a new episode is started. From Fig. 7.5, we can see that, as the number of episodes increases, the cumulative average of final energy efficiency at each episode tends to converge in a certain range. For instance, when the number of episodes is 150, the value of cumulative average energy efficiency becomes at least $2.5 \times 10^6$, achieving a higher value than the preset threshold of $10^6$. 

![Figure 7.5: The cumulative average of energy efficiency over episodes.](image)
7.5 Conclusion

In this chapter, we have proposed a novel framework to optimize the energy efficiency of the BS assisted by an RIS using energy harvesting. We have formulated the problem of maximizing the average energy efficiency which enables the BS to jointly optimize the transmit power allocation, RIS phase shifter, and RIS reflector’s ON/OFF state effectively in the presence of uncertainty about wireless channel and available energy of an RIS. We have shown that by using the deep RL approach, the network parameters are suitably determined by the BS without knowing any prior information on wireless environment. The simulation results show that having two times of the harvested energy improves energy efficiency up to 21.1%.
Chapter 8

Channel Allocation for Device-to-Device Communications

8.1 Background, Related Works, and Contributions

To cope with emerging mobile data traffic, D2D communication has been proposed as a key technology for improving wireless capacity and coverage [157]. D2D over cellular networks allows wireless users to directly communicate with each other without using the cellular infrastructure thus reducing power consumption and improving data rates [158]. As D2D user pairs are typically within a short distance of one another, one can exploit the potential of full-duplex D2D communications to further improve the system performance. However, to reap the benefits of full-duplex D2D communications, one must address a number of challenges ranging from interference management to network modeling and resource allocation [84, 159–163].

In D2D communications, D2D users coexist with cellular users and share their resources. Such resource sharing, if not properly managed, can lead to harmful mutual interference. Thus, a careful interference management scheme is required under the coordination of a BS [159]. To mitigate interference and enhance spectral efficiency, the works in [84, 160, 161] study several resource allocation strategies. For instance, the authors in [160] propose a resource allocation scheme to maximize the network throughput with quality-of-service (QoS) constraints. The authors in [161] propose a channel assignment algorithm based on dynamic programming. Then, a suboptimal clustering algorithm is proposed to form groups of users over a bipartite graph, and, then, a queueing-based algorithm is used to determine channel assignments for clusters. In [84], the authors develop a game-theoretic model to address the problem of D2D sum-rate maximization under QoS constraints.

Furthermore, the authors in [162] and [163] study spectrum resource sharing for full-duplex radios and D2D communications. The work in [162] investigates the problem of maximizing user connectivity by proposing a two-stage approach. In the first stage, a bipartite matching problem is used to assign half-duplex users to the channels of a full-duplex BS, and, then the remaining users are offloaded to the unlicensed bands by using D2D mode. In [163], the authors study the system performance when full-duplex or half-duplex D2D users share the cellular channels to maximize the system throughput.

In all of these existing D2D and full-duplex communication resource management works [84,159–163], it is generally assumed that information on the D2D users such as the total number and locations of the D2D users is completely known. However, in practice, D2D communication can be spontaneously initiated by users that dynamically join and leave the network and, as such, the presence of a D2D link can be uncertain. Indeed, it is challenging for the
Figure 8.1: System model of the underlaid D2D network in the cellular system.

network to know when and where a D2D pair will be available. This is particularly important in dense networks in which users can join and leave at a high rate. Thus, there exists an inherent uncertainty stemming from the unknown locations or number of D2D users. Further, most of the existing works [84, 160, 161] on channel assignment problems typically assume half-duplex communications for D2D users. In contrast, the use of in-band full-duplex D2D users which can transmit and receive information simultaneously over the same channel can significantly improve D2D performance, if properly deployed. Consequently, unlike the existing literature [84, 160, 161] which assumes full information knowledge for half-duplex D2D, our goal is to assign, using an online approach, the best channel for each full-duplex D2D users under uncertainty.

The main contribution of this chapter is to develop a novel framework for online channel allocation in full-duplex D2D networks. This framework allows the network to dynamically allocate the most suitable channel to newly arriving D2D users in the presence of uncertainty on the arrival order of D2D pairs. In particular, we formulate an online optimization problem whose objective is to maximize the sum data rate of all D2D users by properly assigning channels. To solve this problem, we propose novel approach based on the tools of online weighted bipartite matching. We solve the proposed online matching problem using a practical greedy algorithm that enables each D2D pair to smartly share the channels that are already used by cellular users without any prior information on future D2D arrivals and their locations. For an illustrative case study, we derive upper and lower bounds on the competitive ratio to compare the performance of the proposed online algorithm to that of an offline algorithm. Simulation results show that the proposed online algorithm can maximize the sum data rate of the D2D network and achieve a performance that is near-optimal compared to an offline solution that has full information on D2D arrivals.

The rest of this chapter is organized as follows. In Section 8.2, the system model is presented. In Section 8.3, we formulate the proposed online problem. Section 8.4 presents our proposed online solution. Simulation results are analyzed in Section 8.5 while conclusions are drawn in Section 8.6.

8.2 System Model

Consider a cellular network in which cellular users share resources with an underlaid, full-duplex D2D network. As shown in Fig. 8.1, $M$ cellular users are present within the coverage area of a single BS. Each cellular user $m \in \mathcal{M} \triangleq \{1, \cdots, M\}$ uses a channel of bandwidth $B$. We consider an OFDMA system in which the total number of channels
in the system is equal to $M$. Since each channel is allocated to each user, we use the same index $m$ for cellular user and the user’s channel. We assume that the cellular network is fully loaded [160] such that all $M$ cellular users occupy the $M$ orthogonal channels without leaving any spare spectrum. Without loss of generality, we consider that this spectrum sharing occurs during the uplink of cellular transmissions [164–166]. Under this single-cell OFDMA model, cellular users will not interfere with each other. Moreover, the BS and non-D2D cellular users are assumed to operate in half-duplex mode as typically done in the literature [163].

In our network, $M$ cellular users coexist with $I$ D2D pairs. The pairing of D2D users is assumed to be pre-determined. The total number of the D2D users is $2I$. Each pair of D2D, $i \in \mathcal{I} \triangleq \{1, \cdots , I\}$, consists of two users denoted by $2i−1$ and $2i$, respectively. While the cellular users use half-duplex communications, all D2D users exploit full-duplex communications [163]. Since the D2D-paired UEs can transmit and receive information over the same channel, if properly optimized, full-duplex communication can increase the data rate of the D2D network. In our model, given their proximity, the D2D users can use a lower transmit power than the cellular users to mitigate the interference to other cellular users and D2D pairs.

First, we define the uplink data rate of a cellular user $m$ where a cellular user and a D2D pair $i,m$ share channel $m$. The transmit powers of a cellular user and a D2D user are, respectively, denoted by $P^C$ and $P^D$. In the uplink of cellular user $m$, the BS experiences interference over channel $m$ due to the transmission of the D2D pair that is sharing channel $m$. Therefore, when cellular user $m$ transmits information to the BS, the signal-to-interference-plus-noise ratio (SINR) over channel $m$ at the BS will be:

$$
\Gamma_i^C(m) = \frac{g_m P^C}{BN_0 + \sum_{k=(2i−1,2i)} g_k P^D},
$$

where $N_0$ is the noise spectral density, $g_m = \beta d_m^{-\alpha}$ is the channel gain with $d_m$ being the distance between cellular user $m$ and the BS, and $g_k = \beta d_k^{-\alpha}$ is the channel gain between D2D user $k$ and the BS with $d_k$ being the distance between them. $\alpha$ and $\beta$ are, respectively, the path loss exponent and path loss constant. We assume perfect channel state information is available. Also, a time-invariant block fading channel is considered. Then, the uplink data rate of a cellular user $m$ will be:

$$
R_i^C(m) = B \log_2(1 + \Gamma_i^C(m)).
$$

Thus, if the received signal power from the D2D users increases, the cellular user experiences a higher co-channel interference during the uplink. To effectively share the cellular channels, the D2D users must ensure that the cellular users’ QoS does not go below a certain threshold. When assigning the channels for D2D users, the data rate of existing cellular users must be maintained at a minimum threshold $\gamma^C$ so that the QoS is guaranteed. To share a channel $m$ with a D2D pair $i$, the data rate of the cellular user has to be greater than the threshold $\gamma^C$ such that $R_i^C(m) \geq \gamma^C$. We define a variable $\omega_{i,m}$ to indicate whether a D2D pair $i$ can be assigned to a channel $m$ as follows:

$$
\omega_{i,m} = \begin{cases} 
1, & \text{if D2D pair } i \text{ is admissible to channel } m, \\
0, & \text{otherwise}. 
\end{cases}
$$

(8.3)

To measure the performance of the D2D network, we define the data rate of D2D pair $i$. When two D2D-paired users $k = 2i−1$ and $k' = 2i$ share a channel $m$ with cellular user $m$, the SINR of a D2D link from user $k$ to $k'$ will be:

$$
\Gamma_{k\rightarrow k'}(m) = \frac{g_{k,k'} P^D}{BN_0 + g_{m,k'} P^C + g_0 g_{k,k'} P^D},
$$

where $g_{k,k'} = \beta d_{k,k'}^{-\alpha}$ is the channel gain with $d_{k,k'}$ being the distance between D2D users $k$ and $k'$, $g_{m,k} = \beta d_{m,k}^{-\alpha}$ is the channel gain between D2D user $k$ and the cellular user $m$ with $d_{m,k}$ being the distance between them, $g_0$ is the self-interference cancellation at the analog components, and $g_{k,k'}$ shows the self interference. Similarly, we define $\Gamma_{k'\rightarrow k}(m)$ as the SINR of a D2D link from user $k'$ to $k$. Due to channel reciprocity, we assume that $g_{k,k'}$ equals to $g_{k',k}$. Then, the data rate in D2D mode will be given by:

$$
R_i^D(m) = B \left( \log_2(1 + \Gamma_{k\rightarrow k'}(m)) + \log_2(1 + \Gamma_{k'\rightarrow k}(m)) \right).
$$

(8.5)
The sum data rate of the uplink of users $k$ and $k'$ captures the data rate of the D2D pair $i$ due to the full-duplex communications. In (8.5), we observe that the uplink of a cellular user will create interference at the corresponding D2D thus affecting its data rate.

8.3 Problem Formulation

Given the defined system model, our goal is to analyze the optimal channel assignment problem for D2D users. In the D2D network, D2D links can be created in a dynamic manner as users join and leave the network. As such, the BS is unable to know a priori whether new D2D pairs will be formed in the network or not. Moreover, since the total number of D2D pairs as well as the location of each such pair are unknown and highly unpredictable, optimizing channel assignment becomes quite challenging. Under such uncertainty, assigning channels to existing D2D pairs must also account for potential arrival of new D2D pairs. In fact, even if a given channel allocation can improve the performance of an existing D2D pair, it may have a detrimental effect on an incoming pair. To cope with the uncertainty of D2D arrivals while considering the data rate of current and future D2D pairs, we introduce an online optimization scheme that can handle channel assignment under uncertainty.

First, we formulate the following online channel assignment problem whose goal is to maximize the sum data rate of all D2D users:

$$\max \sum_{i=1}^{I} \sum_{m=1}^{M} x_{i,m} R_{i,m}^{D}(m),$$

subject to

$$\sum_{m=1}^{M} x_{i,m} \leq 1, \forall i \in I,$$

$$\sum_{i=1}^{I} x_{i,m} \leq 1, \forall m \in M,$$

$$R_{i,m}^{C}(m) \geq \gamma_{C},$$

where $x$ is a vector whose elements $x_{i,a}, \forall i \in I, m \in M$, indicate the channel assignment. In (8.6), the objective function is the sum of data rate of all current D2D users. We determine the channel assignment $x$ so that the sum data rate is maximized. To guarantee the QoS of cellular users, if $\omega_{i,m} = 0$, then $x_{i,m} = 0$.

In (8.6), while the number of channels and information about cellular users are known, we assume that D2D pairs arrive in an online and arbitrary manner. This implies that the information about each D2D pair is collected sequentially. For example, a couple of non-D2D users can be paired spontaneously to initiate a D2D link. Similarly, D2D-paired users that were idle can suddenly re-initiate a D2D link. Both such cases show that the BS is unable to know any information on future D2D pairs. Therefore, in our problem, the arrival order is represented by an index $i$. At each arrival event, the arrival order $i$ increases by one. Also, index $i$ can be seen as the time order of arrival. Thus, the first arriving D2D pair is indexed by $i = 1$, and similarly, the D2D pair that arrives at order $n$ has index $i = n$. The number of D2D pairs is unpredictable, and, thus, $I$ is an unknown value. When D2D pair $i = n$ arrives, we know the information of only the D2D pairs $i \leq n$. Under such incomplete information, finding the optimal solution of (8.6) is challenging and, as such, one has to seek an online, sub-optimal solution that is robust to uncertainty.

When a D2D pair appears in the network, we must assign it one of the cellular channels. Since each channel has a limited capacity, the channel allocated to a given D2D pair may be re-allocated to another, incoming D2D pair if this newly arriving pair can yield a higher rate. The first constraint of problem (8.6) shows that one or no channel can be assigned to the D2D pair $i$. This also implies that, if a D2D pair $i$ does not acquire a channel, then this pair cannot communicate. The second constraint indicates the a cellular user’s channel $m$ can be shared with at most one D2D pair. Note that, if the number of D2D pairs $I$ is greater than the number of channels $M$, the proposed solution can still...
be applied as follows. When the first $M$ D2D pairs are assigned to $M$ channels, the next $K$ arriving D2D pairs can be assigned by considering the interference from not only cellular users but also newly arriving D2D users. In this case, constraint (8.7) can be substituted by $\sum_{m=1}^{M} x_{i,m} \leq \lceil I/M \rceil$.

The dual of problem (8.6) will be:

$$
\min_{\mu, \lambda} \quad \sum_{m=1}^{M} \mu_m + \sum_{i=1}^{I} \lambda_i, \quad (8.10)
$$

subject to:

$$
\mu_m + \lambda_i \geq R^D_i(m), \forall m \in M, \forall i \in I, \quad (8.11)
$$

$$
\mu_m \geq 0, \lambda_i \geq 0, \forall m \in M, \forall i \in I, \quad (8.12)
$$

$$
R^C_i(m) \geq \gamma^C, \quad (8.13)
$$

where vectors $\mu$ and $\lambda$ consist of elements $\mu_m, \forall m \in M$, and $\lambda_i, \forall i \in I$, respectively. In (8.10), the dual solutions are $\mu$ and $\lambda$. We use an online algorithm to solve (8.6) and (8.10) to optimize the channel assignment in Section 8.4.

### 8.4 Online Channel Assignment: Weighted Bipartite Matching Approach

#### 8.4.1 Construction of a Weighted Bipartite Graph

To represent the online problem on a graph, we build a weighted bipartite graph $G = (U, V, E)$ in which the cellular channels are denoted by the vertex set $U$, the D2D users are captured by the vertex set $V$, and the achievable data rates of each D2D pair are represented by the set of edges $E$. For example, in Fig. 8.2, a total of $M$ channels are shown as vertices in the upper half, and the D2D pairs are shown in the lower half. Also, the edges show the data rate of D2D pair $i$ using channel $m$, i.e., $R^D_i(m)$. If $\omega_{i,m} = 0$, the two vertices $i$ and $m$ are not connected. Every edge indicates that a matching between D2D pair $i$ and channel $m$ is possible. Note that the D2D pairs arrive online. Thus, for example, when D2D pair 1 arrives in Fig. (8.2), we do not have any information on any D2D pair $i \geq 2$. When a D2D arrival event happens, the related values including $\omega_{i,m}$ and $R^D_i(m), \forall i \in I, \forall m \in M$ are calculated, and the graph is updated.
Algorithm 8.6 Online Greedy Algorithm

1: Initialization: $\mu_m = 0, \forall m \in \{1, \cdots, M\}$.
2: Repeat:
3:  If D2D pair $i$ arrives then
4:    Find $m^* = \arg\max_{m' \in \mathcal{M}} R^D_i(m') - \mu_{m'}$
5:       s.t. $\mathcal{M} = \{m|\omega_{i,m} = 1\}$.
6:  If $R^D_i(m^*) - \mu_{m^*} \geq 0$ then
7:    Assign the D2D pair $i$ to channel $m^*$.
8:    If $\sum_{i'=1}^{i-1} x_{i',m^*} = 1$, then
9:      Find $\beta_{i\text{old}}$ s.t. $x_{i\text{old},m^*} = 1$, 
10:     Update $x_{i\text{old},m^*} = 0$.
11:    Update $\mu_m$ and $\lambda_i$.

8.4.2 Greedy Online Algorithm: Procedure and Analysis

To find the channel assignment vector $x$ that maximizes the sum rate, we propose a greedy online algorithm shown as Algorithm 8.6. The BS can use Algorithm 8.6 to assign channels to each D2D pair. To develop the proposed algorithm, we use the structure of the dual problem (8.10). Our proposed greedy algorithm builds on the primal-dual algorithm with greedy update rule that is introduced in [167].

In Algorithm 8.6, we first initialize the dual variable $\mu_m$ to 0 for all channels $m$. When a D2D pair $i$ arrives online, we calculate the gain that is defined by $R^D_i(m) - \mu_m$ for all channels. After that, the channel $m^*$ that has the largest gain is assigned to the D2D pair $i$, and we set $x_{i,m^*} = 1$. If the gain for each channel is negative, then we do not assign a channel and leave the D2D pair unassigned. Also, if channel $m^*$ has been already assigned to another D2D pair $i'$, then the previous channel assignment of $i'$ is canceled; thus, we set $x_{i',m^*} = 0$. Finally, we can have the dual solutions, $\lambda_i$ and $\mu_{m^*}$. We update $\lambda_i = R^D_i(m^*) - \mu_{m^*}$. Here, we set $\beta_{i,m^*}$ to the value of $R^D_i(m^*)$. Essentially, for each $i$, we find the solution by increasing $\mu_m$ and calculating the corresponding $\lambda_i$ while $\mu_m$ and $\lambda_i$ should satisfy the constraint (8.11). The complexity of Algorithm 1 experienced by a D2D pair at each time pertains to the process of finding the maximum value in an array, so the worst-case complexity is $O(M)$. Since the size of the search space is limited by the number of channels, Algorithm 1 can be executed in a reasonably short time. The online algorithm significantly reduce the complexity compared to the offline, exhaustive search approach which has a complexity of $O(2^M)$.

To illustrate the effect of reassigning a channel, we consider the case in which a new D2D pair $i'\text{new}$ arrives, and the BS reassigns channel $m$ from $i'\text{old}$ to $i'\text{new}$. This will only happen if the data rate of $i'\text{new}$ is greater than $\mu_m$. It is due to the fact that $\lambda_{i'\text{new}} = R^D_{i'\text{new}}(m) - \mu_m \geq 0$ where $\mu_m = R^D_{i'\text{old}}$. Thus, we have $R^D_{i'\text{new}}(m) \geq \mu_m = R^D_{i'\text{old}}$. Consequently, the proposed algorithm always find the channel assignment that has incremental, marginal gain; the sum data rate of the D2D network increases at each D2D arrival event unless the D2D pair that arrived is assigned to a channel.

While for the general case with $M$ D2D users, analytical results are challenging to derive, we can still gain an insight on the algorithm performance for a special case with two channels and two D2D pairs arriving online. For this two D2D pairs, two channels case, we assume that D2D pair 1 arrives first where D2D pair 1 can be assigned to channel 1 and 2. Then, D2D pair 2 arrives, but it can be assigned to channel 1 only. This example is illustrated in Fig. 8.2 if we account for D2D pair 1 and 2 only. Then, for the given example, we provide an analysis to measure the benefit of the proposed online algorithm. For the analysis, we define $ALG$ and $OPT$. $ALG$ is the value of the objective function when using the proposed algorithm while $OPT$ is the optimal value of the problem (8.6) when using an offline solution. The offline scenario means that we already know all the information on the total number, the locations, and achievable data rates of D2D pairs. To measure the performance of the proposed online algorithm, we compare the ratio between $ALG$ and $OPT$. This is known as competitive analysis, and a ratio $\frac{ALG}{OPT}$ is called a competitive ratio. A
competitive ratio can be a metric showing how close the performance of an online algorithm is compared to an offline optimal solution.

For our case study, we derive the upper and lower bounds on the competitive ratio to compare the performance of the proposed algorithm compared to the optimal solution as follows.

**Theorem 11.** We show that a ratio between $OPT$ and $ALG$ satisfies $\frac{1}{2} < \frac{ALG}{OPT} < 1$ for the given example.

**Proof.** When D2D pair 1 arrives, D2D pair 1 can be assigned to channel 1 or 2 where channel 1 and 2 yields different data rates, $R_1^D(1)$ and $R_1^D(2)$, respectively. First, if we consider $R_1^D(1) < R_1^D(2)$, then D2D pair 2 is assigned to channel 1, thus having data rate $R_2^D(1)$. In this case, the channel assignment by the online algorithm is also optimal. Thus, we have $ALG = OPT = R_1^D(2) + R_2^D(1)$, and $\frac{ALG}{OPT} = 1$.

Next, we consider the other case $R_1^D(1) > R_1^D(2)$. Then, the online algorithm assigns D2D pair 1 to channel 1. After that, D2D pair 2 arrives where it can use channel 1 only. However, channel 1 has been assigned to D2D pair 1 due to $R_1^D(1) > R_1^D(2)$. If $R_1^D(1) - R_1^D(1) > 0$, then channel 1 is reassigned to D2D pair 2, and D2D pair 1 loses assignment. In this case, the results are $ALG = R_2^D(1)$ and $OPT = R_1^D(2) + R_2^D(1)$. Since $R_1^D(1) > R_1^D(1)$ and $R_1^D(1) > R_1^D(2)$, we have inequalities given by $\frac{1}{2} = \frac{R_2^D(1)}{R_1^D(2) + R_2^D(1)} < \frac{R_2^D(1)}{R_2^D(1) + R_2^D(2)} < \frac{R_2^D(1)}{R_2^D(1) + R_2^D(2)}$.

Moreover, due to $R_1^D(1) < R_1^D(2) < R_2^D(1) + R_1^D(2)$, we additionally know $\frac{R_2^D(1)}{R_2^D(1) + R_2^D(2)} < 1$. Therefore, the boundaries of the ratio can be shown as

$$\frac{1}{2} < \frac{ALG}{OPT} = \frac{R_2^D(1)}{R_1^D(2) + R_2^D(1)} < 1. \quad (8.14)$$

Otherwise, channel 1 is still assigned to D2D pair 1, and D2D pair 2 do not acquire a channel. Then, the result are $ALG = R_1^D(1)$; $OPT = R_1^D(1)$ if $R_1^D(1) > R_1^D(2) + R_2^D(1)$, and otherwise, $OPT = R_1^D(2) + R_2^D(1)$. If $OPT = ALG = R_1^D(1)$, then $\frac{ALG}{OPT} = 1$. Thus, we focus on the later case $OPT = R_1^D(2) + R_2^D(1)$ when $R_1^D(1) < R_1^D(2) + R_2^D(1)$. In this case, we have two conditions, $R_1^D(1) > R_1^D(2)$ and $R_1^D(1) > R_2^D(1)$, so the two inequalities result in $2R_1^D(1) > R_1^D(2) + R_2^D(1)$ that is $\frac{1}{2} < \frac{R_1^D(1)}{R_1^D(2) + R_2^D(1)}$. Therefore, the competitive ratio can be bounded a follows:

$$\frac{1}{2} < \frac{ALG}{OPT} = \frac{R_1^D(1)}{R_1^D(2) + R_2^D(1)} < 1. \quad (8.15)$$

Hence, in the given example of the online maximization problem, the value achieved by the online algorithm is at least half of the offline optimal value.

For the more general case, we will provide thorough analysis via simulations in Section 8.5.

### 8.5 Simulation Results

For our simulations, we consider a single-cell environment where the cellular users are uniformly distributed within a 50 m × 50 m area. We consider a network with 10 channels each of which is allocated to a cellular user. Also, the sequence of D2D users’ arrival follows a uniform distribution. The bandwidth of each channel is 200 kHz, such that the total bandwidth of the single cell is 2 MHz. The power spectral density of the thermal noise is -174 dBm/Hz. We set $\alpha = 2$ and $\beta = 10^{-3}$ to model the channel gain. Also, at the reference distance of 1 m, the channel gain is $-30$ dB [168]. For the D2D users, $g_{k,k}$ is set to 0.03, $\forall k \in \{2i - 1, 2i\} \forall i \in I$ assuming that the circulator provides 15 dB of isolation of self-interference [168], and we assume that an analog cancellation additionally provides
Figure 8.3: The total data rate of the D2D network compared with optimal, offline algorithm for the different maximum D2D distances.

$g_0 = -60$ dB of self-interference cancellation. Also, the location of D2D users are also uniformly distributed, and the distance between the two paired D2D users are less than the given maximum D2D distance. All statistical results are averaged over a large number of simulation runs. For comparison, we use the offline, optimal algorithm that has complete knowledge of the system.

In Fig. 8.3, we show the total data rate of the D2D network as the number of D2D pairs varies for various maximum D2D distances with $\gamma^C = 1 \times 10^5$ bps. From this figure, we can first see that the total data rate of our proposed algorithm achieves a performance that is quite close to the optimal solution derived using exhaustive search. This demonstrates the effectiveness of the proposed algorithm. For instance, Fig. 8.3 shows that the optimality gap is at most 8.3% at 20 m of the maximum D2D distance for a system with 10 D2D pairs.

In Fig. 8.4, we show the total data rate of the D2D network resulting from our proposed approach for different thresholds $\gamma^C$. From Fig. 8.4, we can see that, as the threshold value decreases, the overall data rate of the D2D network will increase. This is due to the fact that having a smaller threshold value will enable a D2D pair to have a larger number of channel options that can be assigned. This, in turn, prevents cases in which a D2D pair is not assigned to any channel. This can be commonly observed at all different D2D distances. For example, if the threshold $\gamma^C$ decreases from $2 \times 10^5$ to $1 \times 10^5$, the system exhibits a performance improvement of up to 31.4% in the total data rate at 10 D2D pairs and 20 m of the maximum D2D distance.

In Fig. 8.5, we show the average data rate of a user with $\gamma^C = 1 \times 10^5$ bps and a maximum D2D distance of 10 m, for two different transmit powers for the cellular users, 24 and 28 dBm. The results show that the D2D and total data rates increase when a lower transmit power is used by the cellular users. From Fig. 8.5, we also observe that the data rate of a user in the network decreases as the number of D2D pairs increases. This is due to the fact that an increase in the number of D2D pairs will naturally increase the interference on the cellular users. However, if a lower transmit power is used, then we can mitigate the effect of this interference at the expense of a lower rate for the cellular users. For instance, at 10 D2D pairs, the case using 24 dBm of transmit power results in 34.3% higher average data rate for D2D user compared to one using 28 dBm.
Figure 8.4: The total data rate of the D2D network for the different QoS thresholds of cellular users.

Figure 8.5: The average data rate per user when cellular users use different transmit power.
Figure 8.6: The number of non-assigned D2D pairs for different maximum D2D distances.

Fig. 8.6 shows the total number of the non-assigned D2D pairs for various maximum D2D distances. The proposed online algorithm results in a lower number of non-assigned D2D pairs when the maximum D2D distance decreases. This illustrates that as the distance between two D2D users becomes closer, the D2D pair may have larger number of channels where the D2D pair can be assigned. For example, Fig. 8.6 shows that decreasing the maximum D2D distance from 20 m to 5 m yields 19.6% reduction in the number of non-assigned D2D users.

8.6 Summary

In this chapter, we have proposed a novel approach to optimize the channel assignment for the D2D network as an underlay of the cellular network. We have formulated the problem as an online weighted bipartite matching which enables the BS to assign channels to the D2D network effectively in the presence of uncertainty about D2D arrivals. We have shown that by using the greedy online algorithm, the suitable channel can be assigned to each D2D pair without knowing any prior information on future D2D arrivals. Simulation results have shown that the proposed online algorithm can achieve a total sum-rate that is no less than 8.3% below the optimal, offline solution found via exhaustive search.
Chapter 9

Conclusions and Open Problems

The main body of our work in this dissertation can be summarized as follows. In the first chapter, we have presented the main benefits, applications, and challenges of edge computing over wireless networks. Particularly, in edge computing, the communication latency can be reduced since the cloud’s functionalities are moved to the edge nodes. Therefore, edge computing can be applied to many areas including cellular systems, IoT networks, vehicular network, and smart factories. Despite the major benefits and practical applications of adopting edge computing architecture, we need to address the key challenges such as radio and computing resource allocation when the future information is unknown.

In the following sections, we have investigated a number of problems related to these technical challenges as well as potential applications of edge computing. In particular, in Chapter 2, we have studied a joint problem of edge network formation and computational task distribution in a hybrid edge-cloud system. To this end, a novel online framework is proposed to form an edge network by selecting a set of edge nodes, distribute the computational tasks, and update a target competitive ratio defined as the ratio between the latency achieved by the proposed online algorithm and the optimal latency that can be achieved by an offline algorithm. The results show that the proposed framework achieves the target competitive ratio that is affected by the wireless data rate and computing speeds of edge nodes. Also, in Chapter 3, we have investigated the problem of ephemeral edge computing where the total time period of edge computing is limited. We first have formulated a new problem of maximizing the number of computed tasks under the uncertainty on future task arrivals. Then, a novel online framework is proposed to enable a source edge node to offload computing tasks from sensors and allocate them to neighboring edge nodes for distributed task computing, within the limited total time period. The proposed framework is shown to effectively maximize the number of computed tasks by jointly considering the communication and computation latency.

Next, Sections 4 and 5 investigate that edge computing can be used to implement the applications using an intense computation. In Chapter 4, we have proposed a mobile blockchain architecture where edge nodes are used as mobile miners who performs computation to maintain a blockchain. We have analyzed the performance of a mobile blockchain system in terms of the probability of a forking event and the average energy consumption of mobile miner. In Chapter 5, we have developed a novel framework for online computational caching in an edge network that allows the optimization of the selection of the input intermediate computed results under uncertainty on the arrival order of the user’s operation. To reduce the communication and computing latency, the proposed caching framework enables each edge node to store IRs from previous computations and download IRs from neighboring nodes under uncertainty on future computation. Simulation results show that the total latency can be significantly reduced by leveraging the computational caching method.

Furthermore, Chapters 6 and 7 of this dissertation, we have studied how online optimization can effectively handle the uncertainty innate in energy harvesting, and also shown that energy harvesting can be used to design an energy-efficient cellular network that is essential for edge computing. In Chapter 6, we have proposed a novel approach to optimize the
ON/OFF schedule of self-powered SBSs which enables the network to operate effectively in the presence of energy harvesting uncertainty. To this end, the problem of online ON/OFF scheduling of self-powered SBSs is studied, in the presence of energy harvesting uncertainty with the goal of minimizing the operational costs that consist of energy consumption and transmission delay of a network. The results show that the ON time of SBSs is affected by the factors such as the harvested energy and the power consumption of the base stations, while demonstrating that the proposed algorithm can reduce the total operational cost of a cellular network. By leveraging the proposed framework, the self-powered base stations can be widely deployed and those base stations can be used for edge computing. Additionally, in Chapter 7, we have studied the deployment of a self-powered RIS system that assists the downlink transmission of the BS. Therefore, we have proposed a novel framework based on a deep reinforcement learning to optimize the energy efficiency of the downlink cellular system when an RIS system solely relies on using energy harvesting techniques. Finally, in Chapter 8, we have proposed a novel approach to optimize the channel assignment for the D2D network as an underlay of the cellular network in the presence of uncertainty about D2D arrivals. Such a framework can be readily used to improve the connectivity among a large number of edge nodes.

While we have addressed several key challenges in edge computing over wireless networks, some of our results can be further extended. Therefore, in this chapter, we next present an overview of the research work which have been performed in this dissertation, and we introduce a number of new directions that can be explored.

9.1 Summary

9.1.1 Edge Computing over Wireless Networks: Applications and Challenges

In Chapter 1, a comprehensive overview has been provided on the benefits and potential applications of edge computing over wireless networks. We have explained various applications drawn from a variety of scenarios and use cases. In particular, we have shown an overview of edge computing architecture focusing on a wireless radio access network. Also, we have investigated the use cases such as edge computing in the IoT sensor networks where the application handling a large volume of data uses edge computing. Moreover, the important challenges of edge computing for vehicular networks have been investigated to emphasize the importance of the latency requirement for safety applications. Finally, we have shown that edge computing is beneficial to operate a smart factory using UAVs, drones, and robots to offload and process the sensory data.

9.1.2 Distributed Edge Network Formation with Minimal Latency

In Chapter 2, we have studied a joint problem of edge network formation and computational task distribution in a hybrid edge-cloud system. We have addressed an online optimization problem whose objective is to minimize the maximum communication and computational latency of all edge nodes by suitably selecting the set of edge nodes in presence of uncertainty about edge nodes’ arrivals. To this end, we have proposed online optimization algorithms whose target competitive ratio is achieved by suitably selecting the neighboring nodes while effectively offloading the tasks to the neighboring edge nodes and the cloud. The theoretical analysis and simulation results have shown that the proposed framework achieves a low target competitive ratio while successfully minimizing the maximum latency in edge computing.
9.1.3 Ephemeral Edge Computing for Internet of Things

In Chapter 3, we have investigated the problem of ephemeral edge computing where the total time period of edge computing is limited. We have provided that the concept of ephemeral edge computing is applicable to different scenarios including a smart factory, intelligent transportation system, and smart home. By modeling a generalized scenario of ephemeral edge computing, we have proposed a novel framework to maximize the number of successful computations over an edge computing network within a limited time period. To this end, we have formulated online optimization problem that jointly optimizes the communication and computation latency and introduced an online greedy algorithm to solve the problem. Then, by using the structure of the primal-dual problem formulation, we have derived a feasible competitive ratio as a function of the task sizes and the data rates of the edge nodes. Simulation results show that the proposed online algorithm can maximize the number of computed tasks and achieve a performance that is near-optimal compared to an offline solution that has full information on tasks.

9.1.4 Blockchain Systems with Wireless Mobile Miners

In Chapter 4, we have proposed a novel framework using mobile miners to operate a blockchain over a wireless mobile network. We have derived an exact closed-form expression for the probability of occurrence of a forking event and the average mobile miner’s energy consumption required to earn a reward. Simulation results have shown that the wireless transmission power, SNR threshold, and the number of MMs significantly impact the energy consumption of the mobile miners.

9.1.5 Computational Caching in Edge Networks

In Chapter 5, we have proposed a novel framework for online computational caching in an edge network that allows the optimization of the selection of the input intermediate computed results under uncertainty on the arrival order of the user’s operation. We have formulated an online computational caching problem to minimize the transmission latency and computational latency. To solve this problem, we have proposed an algorithm scheduling the intermediate results to compute each of the sequentially arriving operations while managing the stored intermediate results in the memory. The analysis shows an upper bound of the competitive ratio for the formulated online problem.

9.1.6 ON/OFF Scheduling of Energy Harvesting Base Stations

In Chapter 6, we have proposed a novel approach to optimize the ON/OFF schedule of self-powered SBSs. We have formulated the problem that minimizing network operational costs during a period. To solve this online problem, we have approximated the problem as an online ski rental problem which enables the network to operate effectively in the presence of energy harvesting uncertainty. Then, we have proposed deterministic and randomized online algorithm that is shown to achieve the optimal competitive ratio for the approximated problem. Simulation results confirm that both delay and the ON/OFF switching overhead are significantly reduced when one adopts the online ski rental approach.

9.1.7 Energy-Efficient Networking with Reconfigurable Intelligent Surfaces

In Chapter 7, we have proposed a novel framework to optimize the energy efficiency of the BS assisted by an reconfigurable intelligent surface using energy harvesting. We have formulated the problem of maximizing the average energy efficiency which enables the base station to jointly optimize the transmit power allocation, phase shifter, and
RIS reflector’s ON/OFF state effectively. To solve this optimization problem, we propose a novel approach based on deep reinforcement learning by defining the state, action, reward, and policy. This framework allows a base station to dynamically adapt to wireless environment in the presence of uncertainty about wireless channel and available energy of an reconfigurable intelligent surface. We have shown that by using the deep reinforcement learning approach, the network parameters are suitably determined by the base station without knowing any prior information on wireless environment. Moreover, we have conducted a case study to analyze the performance of the studied downlink system by asymptotically deriving an upper bound of the energy efficiency.

9.1.8 Channel Allocation for Device-to-Device Communications

In Chapter 8, we have proposed a novel approach to optimize the channel assignment for the D2D network as an underlay of the cellular network. We have formulated the problem as an online weighted bipartite matching which enables the BS to assign channels to the D2D network effectively in the presence of uncertainty about D2D arrivals. We have shown that by using the greedy online algorithm, the suitable channel can be assigned to each D2D pair without knowing any prior information on future D2D arrivals. Simulation results have shown that the proposed online algorithm can achieve a total sum-rate that is no less than 8.3% below the optimal, offline solution found via exhaustive search.

9.2 Open Problems

Despite a considerable number of studies on edge computing, there are still important open questions and problems that must be investigated. Also, while we have addressed several key challenges in edge computing over wireless networks, some of our results need to be further extended and a number of new directions can be explored. Therefore, we present an overview of the future directions of this dissertation.

9.2.1 Edge Communications in the IoT

To cater for the IoT, the goal of edge communications is enabling edge computing to be seamlessly operated by accessing distributed data stored at IoT nodes over a wireless IoT network. Therefore, wireless IoT networks need to provide high reliability and low latency in transmission. However, in a massive IoT scenario with billions of heterogeneous devices, wearables, sensors, and physical objects, it is challenging to manage and optimize large IoT networks while achieving both high reliability and low latency. Therefore, in terms of open problems for edge communications, there is a need for development of: a) defining a performance metric that characterizes edge computing’s reliability, b) self-organizing resource allocation among the devices in a local area to support various IoT services with different requirements of reliability, latency, and security, c) medium access control protocol to provide massive connectivity of IoT devices using orthogonal multiple access (OMA) and non-orthogonal multiple access (NOMA) designs to improve reliability, and d) self-healing IoT system that reconstruct radio access network.

9.2.2 Edge Machine Learning over Wireless Networks

For network intelligent, it is essential to use edge computing’s real-time information processing capability and edge communication’s achievable performance boundary. Therefore, there is a need to study how computing and communications resources can be leveraged to support many emerging services requiring effective computations of various tasks using machine learning techniques. In particular, there is a need for new advanced ideas from various machine
learning tools to develop novel online, distributed machine learning algorithms to optimize the use of IoT device resources, such as time, energy, and computation. Moreover, fundamental performance analysis is needed to address: a) the tradeoffs involved between the use of each device’s training data size, predictive accuracy, and computational power, under limited IoT device capabilities, and b) the optimal wireless resources needed for learning, under the inherent uncertainty of wireless links.

9.2.3 Joint Design of Cyber-Physical System

An efficient joint design of cyber-physical system (CPS) is an important but overlooked area in edge computing. For example, the IoT system includes physical functions such as controlling and sensing of every object, including actuators, sensors, vehicles, and manufacturing robots. The physical function is coupled with cyber functions such as computation and communications. Therefore, due to highly varying nature of IoT systems, there is a need for development of scalable approaches to cope with the dynamics of the CPS and handle its massive scale of heterogeneous devices. For joint design of CPS, there are numerous problems that must be studied. In particular, an example of CPS is intelligence transportation systems including physical components such as vehicle’s dynamics controller and cyber components such as V2X communications functionalities. Remarkably, in recent years, V2X communications technologies have been studied to enable communications among vehicles and everything around, such as infrastructure and pedestrians. V2X communications is indeed a key component to manage and control physical mobility of future autonomous connected vehicles in intelligent transportation systems. In this example, one of important open problems is development of holistic analytical frameworks incorporating radio and computing resource allocation problems based on edge computing frameworks and large-scale urban traffic management problems to achieve: a) Vehicular ultra-reliable low-latency communication for enhanced active safety, b) vehicular enhanced mobile broadband with high data rate transmission to support vehicle’s infotainment systems, and c) vehicular NOMA for massive connectivity of vehicle nodes in various road scenarios.

9.2.4 Energy-Efficient Green Edge Networking for Massive IoT

As deployment of a trillion new IoT devices is expected, the total energy consumption of massive IoT systems will increase in proportion to the number of IoT nodes, thus resulting in a huge energy consumption on the networks. Development and implementation of mobile edge computing in IoT systems to monitor and control smart grid systems are expected to consume more energy in the future networking systems. In light of these illustrations, it is indispensable to enhance energy efficiency of IoT networks. Designing energy-efficient IoT networks is another key research problem in edge computing over wireless networks. In particular, there is a need for an extended study for: a) evaluation and modeling of IoT networks exploiting energy-harvesting technologies, b) development of decentralized online algorithms based on machine learning techniques to improve energy efficiency by turning off underutilized networking nodes, i.e., base stations or IoT sensors, c) adoption of machine learning to forecast the availability of harvested energy (e.g., photovoltaic generation), and d) experimental implementation and evaluations of an edge computing system.
Bibliography


