Project Title	Virtual Presence in Moving Objects through 5G
Project Acronym	PriMO-5G
Grant Agreement No	815191
Instrument	Research and Innovation Action
Topic	The PriMO-5G project addresses the area of “a) Focus on mmWave and super broadband services” in the call “EUK-02-2018: 5G” of the Horizon 2020 Work Program 2018-2020.
Start Date of Project	01.07.2018
Duration of Project	36 Months
Project Website	https://primo-5g.eu/

D4.1 INTERMEDIATE REPORT ON AI-ASSISTED NETWORKING AND EDGE COMPUTING

Work Package	WP4, Intermediate report on AI-assisted networking and edge computing
Lead Author (Org)	Y. Yi (KAIST), R. Delos Reyes (KAIST), D. E. Hostallero (KAIST), Y. P. Koo (KAIST)
Contributing Author(s) (Org)	K. W. Sung (KTH), W. H. Lee (KTH), U. Challita (EAB), Z. Ghebretensae (EAB), H. Cha (YU), E. J. Jeong (YU), S. H. Kim (YU), S. L. Kim (YU), J. H. Kim (CAU), M. S. Choi (CAU), J. H. Jeon (CAU), D. H. Kim (CAU), G. Destino (KCL), Y. Deng (KCL), T. Mahmoodi (KCL), J. Costa-Requena (CMC), A. Mohammedadem (CMC), Z. Chao (AALTO), X. Yu (AALTO)
Due Date	30.04.2019, M10
Date	29.04.2019
Version	4.0 (Submitted)

Dissemination Level

X	PU: Public
	PP: Restricted to other programme participants (including the Commission)
	RE: Restricted to a group specified by the consortium (including the Commission)
	CO: Confidential, only for members of the consortium (including the Commission)

The work described in this document has been conducted within the project PriMO-5G. This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 815191. The project is also supported by the Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00170, Virtual Presence in Moving Objects through 5G). The dissemination of results herein reflects only the author’s view, and the European Commission, IITP and MSIT are not responsible for any use that may be made of the information it contains.

Versioning and contribution history

Version	Date	Authors	Notes
1.0	20.01.2019	Y. Yi (KAIST), R. Delos Reyes (KAIST), D. E. Hostallero (KAIST), Y. P. Koo (KAIST)	Draft ToC
2.0	21.03.2019	Y. Yi (KAIST), R. Delos Reyes (KAIST), D. E. Hostallero (KAIST), Y. P. Koo (KAIST), K. W. Sung (KTH), W. H. Lee (KTH), U. Challita (EAB), Z. Ghebretensae (EAB), H. Cha (YU), E. J. Jeong (YU), S. H. Kim (YU), S. L. Kim (YU), J. H. Kim (CAU), M. S. Choi (CAU), J. H. Jeon (CAU), D. H. Kim (CAU), G. Destino (KCL), Y. Deng (KCL), T. Mahmoodi (KCL), J. Costa-Requena (CMC), A. Mohammedadem (CMC), Z. Chao (AALTO), X. Yu (AALTO)	Introduction, contributions from different partners in Sections 2 & 3
3.0	09.04.2019	Y. Yi (KAIST), R. Delos Reyes (KAIST), D. E. Hostallero (KAIST), Y. P. Koo (KAIST), K. W. Sung (KTH), W. H. Lee (KTH), U. Challita (EAB), Z. Ghebretensae (EAB), H. Cha (YU), E. J. Jeong (YU), S. H. Kim (YU), S. L. Kim (YU), J. H. Kim (CAU), M. S. Choi (CAU), J. H. Jeon (CAU), D. H. Kim (CAU), G. Destino (KCL), Y. Deng (KCL), T. Mahmoodi (KCL), J. Costa-Requena (CMC), A. Mohammedadem (CMC), Z. Chao (AALTO), X. Yu (AALTO)	Consolidated draft complete with conclusions
4.0	29.04.2019	Y. Yi (KAIST), R. Delos Reyes (KAIST), D. E. Hostallero (KAIST), Y. P. Koo (KAIST), K. W. Sung (KTH), W. H. Lee (KTH), U. Challita (EAB), Z. Ghebretensae (EAB), H. Cha (YU), E. J. Jeong (YU), S. H. Kim (YU), S. L. Kim (YU), J. H. Kim (CAU), M. S. Choi (CAU), J. H. Jeon (CAU), D. H. Kim (CAU), G. Destino (KCL), Y. Deng (KCL), T. Mahmoodi (KCL), J. Costa-Requena (CMC), A. Mohammedadem (CMC), Z. Chao (AALTO), X. Yu (AALTO)	Final version of the document

Disclaimer

PriMO-5G has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 815191. The project is also supported by the Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00170, Virtual Presence in Moving Objects through 5G). The dissemination of results herein reflects only the author's view, and the European Commission, IITP and MSIT are not

responsible for any use that may be made of the information it contains.

Table of Contents

List of Tables

List of Figures

Executive Summary

This document aims to show that the aim of the PriMO-5G is attainable. We demonstrate the possibility of the project's goal providing immersive video services for moving object by showcasing application-driven algorithms that incorporate artificial intelligence to different network communication problems. This deliverable presents selected studies regarding AI-assisted communications that are being conducted by the project members.

Under this work package, we further categorized the research into AI-assisted networking and AI-assisted edge computing. In the AI-assisted networking, we present general networking problems and how they are solved using machine learning and AI. Similar content is presented in the AI-assisted edge computing, where we elaborate on applications that are closer to edge computing.

The document is organized as follows. Section 3 presents AI-assisted networking. It starts with a brief background on AI and machine learning, followed by the literature review and challenges in this area. We then present the current research efforts by each partner. Studies in this section include spectrum access, distributed machine learning, communication scheduling, caching, and UAV networking simulation. Section 4 contains a similar content but for the case of AI-assisted edge computing. This section investigates topics including network slicing, super-resolution, drone battery charging allocation, and quality-aware transmission. Finally, conclusions and outlook are given in Section 5.

List of Acronyms

Acronym	Definition
5G	Fifth-Generation Mobile Network
AI	Aritificial Intelligence
ANN	Artificial Neural Network
ASE	Area Spectral Efiiciency
BBU	BaseBand Units
BE	Best Effort
BPSO	Binary Particle Swarm Optimization
CDF	Cumulative Distribution Function
CNN	Convolutional Neural Network
cGAN	Conditional Generative Adversarial Network
CRAN	Cloud Radio Access Networks
CSMA/CA	Carrier Sense Multiple Access with Collision Avoidance
CTDE	Centralized Training and Distributed Execution
D2D	Drone to drone
DCVDSR	Depth-Controllable Very Deep Super Resolution
DLB	Dynamic Load Balancing
DNN	Deep Neural Networks
DoA	Description of Action
DQN	Deep Q-Network
DSA	Dynamic Spectrum Access
DSRC	Dedicated Short Range Communications
EC	European Commission
EDC	Edge Data Center
ER	Erdos Renyi
ESN	Echo State Network
EU	European Union
FA	Federated Augmentation
FC	Full Communication
FD	Federated Distillation
FNN	Feed-forward Neural Network
GA	Genetic Algorithms
GAN	Generative Adversarial Network
GCS	Ground Control Station
GPU	Graphics Processing Unit
GPS	GIbal Positioning System
HR	High Resolution
ICT	Information and Communication Technologies
IDQN	Independent Deep Q-Networks
IITP	Institute for Information and communication Technology Promotion
ILP	Integer Linear Programming
IoT	Internet of Things
IoV	Internet-of-Vehicle
IPC	Inter-Process Communication

Acronym	Definition
LR	Low Resolution
LSTM	Long Short Term Memory
M2M	Machine to Machine
MAC	Medium Access Control
MARL	Multi-Agent Reinforcement Learning
MBO	Mobile Backhaul Orchestrator
MEC	Mobile Edge Computing
ML	Machine Learning
MLP	Multi Layer Perceptron
MM	Mobility Management
Mxx	Month
NLP	Natural Language Processing
NLOS	Non-Line-Of-Sight
NN	Neural Network
NNE	Neural Network Ensemble
OP	Opportunistic Probability
PTX	Primary Transmitter
PU	Public (deliverable)
QAR	QoS aware Adaptive Routing
QoE	Quality of Experience
QoR	Quality-of-Result
QoS	Quality of Service
RAN	Radio Access Network
ReLU	Rectified Linear Unit
RIA	Research & Innovation Action
RL	Reinforcement Learning
RLP	Repeated Leaning and Placement
RNN	Recurrent Neural Network
RRA	Round Robin Algorithm
SaP	Sense-and-Predict
SGD	Stochastic Gradient Descent
SISR	Single Image Super Resolution
SLAM	Simultaneous Localization and Mapping
SR	Super Resolution
SRCNN	Super Resolution Convolutional Neural Network
STX	Secondary Transmitter
SVM	Support Vector Machine
TC	Tilted learn popularity with Cutting
TCP	Transport Control Protocol
ToC	Table of Contents
ToS	Type of Service
TSF	Time Series Forecasting
UAV	Unmanned Aerial Vehicle
UPF	User Plane Function
URLLC	Ultra Reliable Low Latency Communication
VDSR	Very Deep Super-Resolution

Acronym	Definition
VDN	Value Decomposition Network
VFC	Vehicular Fog Computing
WAVE	Wireless Access in Vehicular Environment
WP	Work Package
WPL	Work Package Leader
WSA	Weight-based Scheduling Algorithm

1 Introduction

1.1 Scope of the document

To show that the goal of the PriMO-5G project is attainable, this document provides a summary of current research in the project's application domains. In particular, it showcases application-driven algorithms for autonomous networking with distributed learning-based computations from AI-assisted networking and edge computing research.

An overview of existing literature and key challenges in the use of AI in the fields of networking and edge computing is provided. This document presents current works in these areas that were able to address these challenges and/or produced state-of-the-art results.

1.2 Structure of the document

This document is organized as follows. Section 2 provides an overview of how AI has been used in solving networking problems and presents the current research in this area. Section 3 shows the similar content but for the case of AI-assisted edge computing. Finally, conclusions and outlooks are given in Section 4.

1.3 Relationship to other project outcomes

This document concretizes the use cases provided in WP1, i.e. Task 1.1, and the concepts presented in WP2, i.e. Task 2.3. With the use of AI, this document demonstrates promising solution approaches that can help in achieving the project's goal.

2 AI-assisted Networking

2.1 Overview

Machine learning (ML) techniques have already transformed multiple fields - e.g., computer vision, natural language processing, speech recognition, and optimal control. The success of these techniques is mainly attributed to many factors, but the important ones include the significant advances in ML techniques such as deep learning, the availability of huge raw data, and the progress in computing and graphics processing unit (GPU) and other hardware enhancements [1].

ML plays a vital role in the networking ecosystem, realizing a future vision of cognitive networks, in which networks will be able to self-organize and autonomously implement intelligent network-wide behavior to solve problems such as routing, congestion management, traffic prediction, traffic classification, and anomaly detection. Machine learning is also an ideal tool to automate many of the traditional infrastructure management processes that are performed manually. Today, the application of ML in network operation and management is at its very early stage and the solutions that exist are mainly focused on cost reduction, routine work simplification, and efficient operation of the network.

In the following subsections, we present a brief overview of artificial intelligence and machine learning, followed by a literature survey and some of the challenges of ML in addressing networking problems.

2.1.1 A Brief Overview of AI and ML

Artificial intelligence (AI) is a broad term which refers to machine's capability to mimic natural intelligence of humans and animals, i.e. to perceive an environment, learn from a data, and solve problems to achieve specific goals. Machine learning (ML), which is a fundamental concept of AI, means the study of algorithms which improve automatically to perform a specific task without explicit instructions. ML has made tremendous improvement in recent years, and it is often considered the most important tool of AI.

ML has three learning paradigms, namely supervised learning, unsupervised learning, and reinforcement learning. Each of the concepts is explained below.

- **Supervised learning** is the most common form of machine learning [2]. Here, the machine learns a relationship between input and output from training data consisting of a set of labelled training examples. Then, the established relationship is used to make an inference to a data.
- **Unsupervised learning** does not have labelled data when a machine performs the learning. In unsupervised learning, the major objective is to identify commonalities in the data and to react to a new piece of data based on the identification.
- **Reinforcement learning** lies between supervised and unsupervised learning. With reinforcement learning, a machine learns the best policy of actions in an environment to maximize the long term reward.

ML can be implemented by various models such as artificial neural network (ANN), support vector machine (SVM), and genetic algorithms (GA). Among those, ANN is arguably the most popular model for both academia and industry mainly owing to the breakthrough that deep learning or deep neural network (DNN) has made.

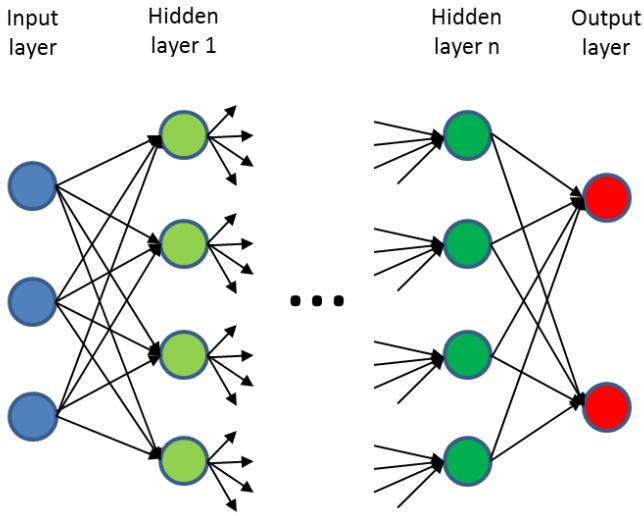


Figure 2-1: an illustration of the typical structure of DNN

ANN is a framework to accommodate many different machine learning algorithms. It is inspired by the biological neural networks that constitute human brains. This means that ANN is an interconnection of simple processing elements, which are called neurons. In ANN, a neuron is a simple non-linear function such as sigmoid or rectified linear unit (ReLU). DNN differs from traditional ANN in that it has multiple hidden layers between the input and output layer. Figure 2-1 illustrates the typical structure of ANN, particularly DNN.

With the soaring interest in the research of ANN, various classes of ANN have been developed and being actively proposed. Some of the popular ANN classes are listed below.

- **Convolutional neural network** (CNN) is a type of feed-forward neural networks (FNN), and it is known to be good at dealing with data in the form of multiple arrays, e.g. images, audio spectrograms, and videos. The hidden layers of CNN consist of convolutional layer, pooling layer, and full connected layer. Convolutional layers perform convolution of input, and passes the result to the next layer. Pooling layers combine the results from the convolutional layers, and fully connected layers does the same job as typical ANN, i.e. connecting every neuron in one layer to every neuron in another layer. By allowing local connectivity and letting each layer perform different operations, CNN is able to deal with high-dimensional data [2].
- **Recurrent neural network** (RNN) is distinguished from the general FNN in that it has connections from one layer to previous layers. This change makes the output of a RNN determined by not only the current input but also the historical input in the past. Thus, RNN has an advantage in processing sequential data such as speech and mobility prediction. An elaborated variant of RNN is long short-term memory (LSTM) where a special unit called the memory cell has gates to decide which data to keep longer and which to discard.
- **Autoencoder** is a type of ANN which is designed for learning a representation (encoding) of a set of data. The aim of autoencoder is to reduce the dimensionality of the data by ignoring noise. Autoencoder may look similar to typical FNN from an architectural perspective. However, the output layer in autoencoders has the same number of nodes as the input layer because its purpose is to reconstruct its own input. Thus, autoendoers can be classified into unsupervised learning.

- **Generative adversarial network** (GAN) is a system of two ANNs contesting with each other in a zero-sum game. GAN is mainly used to create an artificial data, e.g. photograph and voice clip that look or sound like real ones. In GAN, one ANN generates candidate images, voices, or videos, and then the other evaluates the candidates generated by another. Although the evaluating ANN, namely discriminator, is trained by initial training set, the two networks contest with each other without training data for such tasks. Thus, GAN is classified into unsupervised machine learning.

2.1.2 Literature Review

2.1.2.1 AI in General Networking

In this subsection, we identify typical networking problems and discuss how they are previously addressed using machine learning methodologies.

Traffic Routing. Dedicated devices called routers steer traffic across the network from one host to another. The path selection criteria primarily depends on the operation policies and objectives, such as cost minimization, maximization of link utilization, and QoS provisioning. Traffic routing presents some challenging tasks to ML models, such as the ability to cope and scale with complex and dynamic network topologies, to learn the correlation between the selected path and the perceived QoS, and to predict the consequences of routing decisions [3]. Because of their low computational and communication requirements and their ability to perform well at finding an optimal solution and adapting to changes in the environment, most of the ML related work to routing in the existing literature is based on Reinforcement Learning (RL). S.C. Lin et al. [4] propose quality of service (QoS) aware adaptive routing (QAR) in a multi-layer hierarchical SDN networks. Using RL and QoS aware reward function, the proposed algorithm could achieve a time-efficient, adaptive, and QoS provisioning. Simulation results show that QAR outperforms the existing learning solution and provides fast convergence with QoS provisioning, facilitating the practical implementations in large-scale software service-defined networks. In a similar work, A. Bhorkar et al. [5] propose a distributed adaptive opportunistic routing scheme for multi-hop wireless ad hoc networks using an RL framework to opportunistically route the packets even in the absence of reliable knowledge about channel statistics and network model. This scheme is shown to be optimal with respect to an expected average per-packet reward criterion

Congestion control. Congestion refers to a network state when the network nodes or links carry so much data that it may deteriorate network service quality, resulting in queuing delay, frame or data packet loss and the blocking of new connections. The most well-known congestion control mechanisms are those implemented in TCP which operate in the end-systems of the network to limit the packet sending rate when congestion is detected. Another congestion control scheme is the queue management that is used in switches and routers used to complement TCP. One of the main challenges of applying RL to TCP congestion control is the problem's continuous, high-dimensional state space [3]. The size of the state space can grow exponentially with the dimension of the state space, causing a significant increase in the size of the table needed to store the state-action values. It is also very time consuming to update entries in such a large table, which results in unacceptably long training time. To improve the limitations of the core congestion control methods used in TCP based networks, W. Li et al. [6] propose to integrate reinforcement-based machine learning with TCP design called QTCP, which enables senders to gradually learn the optimal congestion control policy and show that it outperforms the traditional rule-based TCP while maintaining low transmission latency. Testing the robustness of using deep reinforcement-based congestion control approach, N. Jay et al. [7] present improved results in terms of link capacity, latency, and buffer size.

Traffic prediction. Traffic prediction plays a fundamental role in network design, management, control, and optimization. It is important for network planning and routing configurations that are implemented to improve the QoS for users and entails forecasting future traffic and traditionally has been addressed via

time series forecasting (TSF) [3]. The objective in TSF is to construct a regression model that is capable of drawing accurate correlation between future traffic volume and previously observed traffic volumes. Supervised neural networks using TSF approaches have been successfully applied to traffic prediction where neural networks are used to infer traffic volumes from past measured volumes, show high long-term and short-term prediction accuracy. E.g., P. Cortex et al. [8] present a neural network ensemble (NNE) for the prediction of TCP/IP traffic by employing a TSF using real-world data from two large Internet Service Providers. The collected data reflected different time scales and forecasting horizons and results show that NNE approach is competitive when compared with other TSF methods. S. Chabaa et al. [9] employ an ANN model based on the multi-layer perceptron (MLP) for analyzing internet traffic data over IP networks and show results that the developed models, can be successfully used for analyzing internet traffic over IP networks. TSF approaches are however restrictive, e.g., in order to predict the traffic for a particular flow f on link l, there must be a counter on link l actively measuring the traffic for that particular flow f, which can be challenging to conduct such measurements at very high speed links at the required speed or granularity. Non-TSF approaches have been investigated in [10], [11], [12] to infer traffic volumes from flow count and packet header fields, Simulation results have, however, shown higher prediction error rates.

Traffic Classification. Different traffic classes can be treated differently to provide the appropriate services. Traffic classification is a vital tool for network operation and management, which enables QoS and service differentiation, performance monitoring, resource provisioning etc. by associating network traffic to pre-defined classes of application or service. Generally, network traffic classification is based on port number, packet payload, host behavior or flow features. Port-based techniques are largely unreliable, while payload-based traffic classification searches through the payload for known application signatures, therefore it incurs higher computation and storage costs. In contrast to these, the flow-based traffic classification techniques inspect the complete communication session and is widely studied using both supervised and unsupervised machine learning. For instance, Y. Jin et al. [13] explore supervised machine learning based flow-feature traffic classification system that combines a series of simple linear binary classifiers. Evaluation of the model using real traffic data from multiple locations in a large ISP show that the system accurately reproduces the labels of the packet level classifier. To address the challenges related to encrypted traffic, W. M. Shbair et al. [14] define dedicated features for HTTPS traffic that are used as input for a multi-level identification framework based on machine learning algorithms and show that they can identify encrypted web services with high accuracy. Traffic optimizations (To) techniques such as flow scheduling, load balancing, etc. in datacenters, which are computed using handcrafted heuristics for varying traffic load, flow size distribution, traffic concentration are typical difficult online decision-making problems. These heuristics are, however, prone to suffer from mismatch between the calculated and the actual flow size distribution in run-time and long turnaround design time since they require operator insight, application knowledge, and traffic statistics collected over a long period of time. Leveraging the long-tail distribution of datacenter traffic, Li Chen et al. in [15] developed a two-level Deep Reinforcement Learning (DRL)system, AuTO, automatic, end-to-end TO system that can collect network information, learn from past decisions, and perform actions to achieve operator-defined goals. Compared to existing approaches, AuTO reduces the TO turn-around time from weeks to ~ 100 milliseconds while achieving superior performance. For example, it demonstrates up to 48:14% reduction in average flow completion time (FCT) over existing solutions.

Fault management. This is concerned with detecting, isolating, and resolving problems in the network. Traditional fault management is reactive while fault prediction is proactive and aims to prevent faults or failures in the future by predicting them and initiating mitigation procedures to minimize performance degradation. Most of the ML based fault management approaches use different supervised learning techniques that depend on training data to predict/detect/locate faults in the network. However, a common challenge faced by these techniques is the scarcity of fault data generated in a production network. While both normal and fault data are easily available for a test or simulated network, only normal data with infrequent faults are routinely available for a production network. Injecting faults can help produce the

required data [16] but it may not be realistic in a production network, while synthetic data generated in a test or simulated network may not perfectly mimic the behavior of a production network. On the other hand, employing unsupervised techniques can take longer time to converge, potentially missing any fault occurring before the convergence. Trying to maintain reliable operation in a cost-efficient manner, Kumar et al. [17] evaluate a proactive failure prediction approach using Support Vector Machine (SVM) Regression and multiple Neural Network variants. The results show that the most promising technique is a Deep neural networks (DNN) utilizing autoencoders. A. John and C. Meirosu [18], the authors describe a novel, computationally efficient, autonomic network fault localization algorithm based on active network measurements and probabilistic inference. The probabilistic inference, which is based on a discrete state-space particle filter, provides probability mass distribution that indicates the fault location. Using Recurrent neural networks (RNNs) to model a sensor node and its interconnections to other sensor network nodes A. Moustapha and R. Selmic [19] present sensor node identification and fault detection model in a dynamic a dynamic wireless sensor networks (WSNs). Simulation results in a 15 node WSN shows successful detection of sensor drift.

2.1.2.2 AI in Wireless Networking

The upcoming 5G system is expected to achieve a considerable increase in data rates, coverage and the number of connected devices with reduced latency and energy consumption. 5G will also open the door for the introduction of a wide range of devices and sensors, in the form of IoT, that can be connected to the mobile network. This will enviably lead to data traffic explosion, which, together with the stringent requirements of the new applications will create unpresented requirement on the network. Fulfilling these tasks in an increasingly complex, heterogeneous, and evolving network using legacy algorithms to the different networking functionalities will be extremely challenging. One potential solution is to apply machine learning algorithms to cope with the complexity and the scalability of the mobile network. Because of their ability to learn from input data, machine learning models are anticipated to cope with the increased complexity and traffic volume in the network and therefore there is a growing interest in applying machine learning to different networking problems including resource management [20] [21] [22], spectrum management [23] [24], mobility management [25] [26] and backhaul management [27].

Resource Allocation. The RL technique has been applied to various fields of wireless communications and networks. G. Alnwaimi et al. [28] and F. Bernardo et al. [29] consider the resource allocation for mobile cellular networks. They introduce the structure of cell association system based on RL technique, which provide global optimization of the reward signal, allowing RL technique to escape from local maxima. O. Onireti et al. [30] introduces the method for selecting reward function that is enabler for convergence of RL based resource allocation algorithm. They also provided extensive comparison of dynamic solution using methods from reinforcement learning: gradient follower, the modified Roth-Erev, and the modified Bush and Mosteller learning algorithms. The authors consider cell outage management for dense heterogeneous networks with split control and data planes. When the cell outages of control plane and data plane is detected, RL based cell outage compensation algorithm arranges the proper neighboring base stations by finding suitable beamforming pattern and downlink transmission power. RL based cell outage compensation algorithm runs in real-time for serving pedestrian and vehicular wireless communications. In S. Maghsudi et al. [31] show the channel selection problem for D2D communications reverts to a multi-player multi-armed bandit game with side information, which is the one of representative RL techniques. They propose a no-regret learning based channel selection rule for selfish D2D users to utilizing spectrum vacancies of cellular network. They show that the ergodic performance of single user is equivalent to optimal spectrum selection algorithm.

In the dynamic spectrum access area, the papers [32] and [33] consider the dynamic multichannel access scenario for users with RL technique. The authors point out the divergence property of existing optimization

techniques when the number of channel is large and apply the RL technique to handle the problem. However, [33] and [34] propose the RL based cognitive radio system for a single user.

Content Caching. Addressing the content caching problem, M. Chen et al. [24] explore a proactive caching solution for cloud radio access networks (CRANs). Assuming that the baseband units (BBUs) can predict the content request distribution and mobility pattern of each user, the problem of determining which content to cache at remote radio heads and the BBUs can be formulated as an optimization problem that incorporates backhaul and fronthaul loads and content caching. They propose a solution based on an algorithm that combines the machine learning framework of echo state networks (ESNs) with sublinear algorithms. Simulation results show that the proposed approach yields significant gains when compared to random caching with clustering and random caching without clustering algorithms. Handover degradation in small cells is another task where machine learning can make a difference. In an effort to improve the handover performance of fast moving UEs in heterogeneous networks, M. Simsek et al. [35] propose context-aware mobility management (MM) procedure for small cell networks, which uses reinforcement learning techniques and inter-cell coordination for improving the handover and throughput performance of UE. Using the proposed MM approaches, they show that the UE throughput performance can be improved substantially while reducing the handover failure probability.

2.1.2.3 Convergence of AI and Networking

The data generated by mobile environments is increasingly heterogeneous, as it is collected from various sources, have different formats and exhibit complex correlations. As a consequence, a range of specific problems become too difficult or impractical for traditional machine learning tools (e.g., shallow neural networks). This is because their performance does not improve with more data and they cannot handle highly dimensional state/action spaces in control problems [36]. In contrast, big data fuels the performance of deep learning, as it eliminates domain expertise and instead employs hierarchical feature extraction. In essence, this means information can be distilled efficiently and increasingly abstract correlations can be obtained from the data, while reducing the pre-processing effort. One of the main advantages of DNNs is their ability to extract many kinds of relationships between a variety of time-varying inputs and outputs which makes them well suited to tackle problems such as content predictions, pattern recognition, classification, regression, clustering, and fault detection [37]. DNN can be used for prediction, inference, and big data analytics purposes enabling the network to learn from the datasets generated by its users, the environment and the network devices. For instance, ANNs can be used to analyze and predict the wireless users' mobility patterns and content requests allowing the BSs to optimize the frequency and content cached across the network. This means that the behavioral patterns pertaining to the users such as their mobility patterns and content requests will significantly affect which content to be cached, at which node in the network, and at which time, improving not only the overall resource utilization efficiently but also provide energy efficiency and improved QoS to users.

Privacy-preserving Distributed Machine Learning. Access to a large amount of data on servers led the modern machine learning algorithms to begin the revolution. On-device machine learning can take a step forward by getting the local devices involved in the learning process with the data samples that they generated and owned [38]. The advantage of reducing latency, cost, and bandwidth are followed by performing training process directly on the edge device [39].

As the edge devices store sensitive data, the need for preserving data privacy during the on-device process has been in demand. Konecný et al. (2016)[40] suggested one of the solutions for the privacy issue; choosing the exchange of model parameters instead of direct upload of the data samples, as exemplified by federated learning. However, federated learning has a drawback in the communication overhead that each device must entail: the number of parameters it uploads is proportional to its model size; therefore,

the local learning process in devices with large-sized models are impeded in federated learning. After this work, Konecný et al. (2016)[41] presented algorithms for reducing communication costs. Wang et al. (2018)[42] also stressed that the additional computation per device enables to train a global model with a smaller number of communications. Specifically, the devices have choices either trying more mini-batches in the local learning process or exchanging gradients more frequently for further decreasing the communication or computation costs.

Furthermore, there is no guarantee that a user-generated training dataset is even. A training dataset of each device is likely to be biased to some of the labels because the samples corresponding to the other labels are scarce. Cao (2014)[43] stated that non-IIDness learning has emerged as a crucial issue in multiple studies varying in fields. Zhao et al. (2018)[44] obtained experimental results under federated learning system, pointing out that a device with a non-IID dataset has lower performance in prediction, showing the accuracy drop by up to 11% for MNIST and 51% for CIFAR-10. Sattler et al. (2019)[45] found that federated averaging across all federated learning settings is vulnerable when the devices hold non-IID dataset while proving higher performance in accuracy and convergence time of the sparsified communication protocols.

For realizing the ubiquitous learning system, the edge devices need the capacity of exchanging their knowledge, i.e., the generalized information that a model obtains after it converges. Since the transmission power or the battery of those devices are weak compared to those of centralized entity, researchers have been focusing on minimizing the burden that arises from communication among devices.

Hinton et al. [46] presented one way to transfer the knowledge based on bigger ensembled networks to smaller single ones is introduced as knowledge distillation. For the convergence of local training model and saving resources than forming an ensemble model, the participants distill knowledge that would boost the generalization performance from an ensemble model including redundant parameters. Zhang et al. [47] studied a two-way distillation, in which the two networks optimize their models regardless of separating roles as a teacher or a student. Since the distillation-based cooperative learning algorithms require synchronized training datasets for all the participants, the output trading scale enlarges as the training dataset size increases.

Multi-agent learning and Networking. Other interesting applications of reinforcement learning to networking involves multiple entities, which calls for the need to model the problem as a multi-agent system. Multi-agent systems have deep ties with communication protocols due to the need of the agents to coordinate their naturally distributed policies. With that, multiple studies about learning cooperation and communication simultaneously have been conducted. Foerster et al. [48], Sukhbaatar et al. [49], Peng et al. [50], Guestrin et al. [51], and Zhang & Lesser [52] experimented various training procedures and neural network architectures to learn communication protocols. In addition, they have provided empirical results that communicative agents achieve better rewards at various tasks. Mordatch & Abbeel [53] and Havrylov & Titov [54] investigate the possibility of the artificial emergence of language. These approaches have demonstrated that agents can learn an interpretable communication protocol without providing a pre-defined context on the communication symbols. Coordinated RL [51] is an earlier work demonstrating the feasibility of structured communication and the agents' selection of jointly optimal action.

DIAL [48] and [52] attempted to address bandwidth related concerns when communicating among agents. In theory, agents can communicate large amounts of data to convey as much information as possible in order to coordinate actions. However, in most many cases, this is not practical due to network costs and other limitations. In DIAL, the training phase takes into account the limited bandwidth, such that the agents are encouraged to establish more resource-efficient protocols. The environment in [52] also has a limited-bandwidth channel in effect, due to a large amount of exchanged information in running a distributed

constraint optimization algorithm.

Recently, mutli-agent reinforcement learning (MARL) with communication has been taking advantage of the spatial locality of agents. ATOC [55] proposed to use an attentional communication model, in which agents to share information among neighboring agents instead of the entire set of agents. Another approach that uses spatial locality is DGN [56], which employed the same approach as graph convolutional networks. DGN treats each agent as a node, where each agent has connections to its neighbors. We do note that these studies do not focus on addressing problems about limited communication range and/or medium contention. Instead, they created general MARL methods that can work well when local communication is favored.

Intelligent UAVs. In general, the traditional approach to autonomous navigation is based on simultaneous localization and mapping (SLAM), and planning algorithm, in which global positioning system (GPS) and range and/or visual sensors equipped drone simultaneously builds a map of the environment and self-localizes in it. However, the SLAM based methods are computationally expensive due to explicit 3D reconstruction of the environment. Recently, with the progress of deep neural networks and the popularity of UAVs, there been an increased research effort in learning control policies for UAVs directly from raw sensory data using supervised learning and reinforcement learning based methods [57] [58]. Supervised learning with its implementation simplicity and its relative low sample complexity has now become the predominant tool used to learn visual-motor policies in UAVs. The main challenge of using supervised learning is how to acquire a relevant learning dataset, especially in urban environments which can be both tedious and dangerous. Additionally, there is always the risk that the domain-shift between expert and agent might hinder generalization capabilities of supervised learning methods. Deep reinforcement learning has also recently gained popularity thanks to its capabilities in solving learning problem without relying on a model of the environment and has now emerged as a powerful technique for automatically acquiring control policies that can process raw sensory inputs and perform complex behaviors. However, extending the trial-and-error learning process to UAVs real-world urban scenarios is often challenging and impractical. In the literature reinforcement learning based algorithms have shown to be successful in learning generalizing control policies [59], however, they usually require a large amount of UAV experience which can be costly and impractical to acquire. Another promising approach is the use of simulations to get training data for reinforcement learning tasks, while testing the learned policy in the real world [60]. Clearly, this approach also suffers from the domain shift between simulation and reality and might require some real-world data to be able to generalize [61]. In [62], F. Sadeghi and S. Levine explore the idea of whether in reinforcement learning approach, one can train vision-based navigation policies entirely in simulation and then transfer them into the real world to achieve real-world indoor flight without a single real training image. They propose a learning method entirely based on 3D CAD models using RGB images from a monocular camera. A deep convolutional neural network was trained using stochastic gradient descent (SGD) with a cross-entropy loss function to predict the Q function, that directly processes raw monocular images and outputs velocity commands used to navigate the UAV through an indoor environment. Inspired by the work on autonomous navigation in [63], A. iusti et al. [64] proposed, TrailNet, a DNN to compute the UAVs view orientation and lateral offset used to control the UAV flying over forest trails. In this case, the orientation parameters were trained using the IDSIA Swiss Alps trail dataset [64], while for learning the lateral offset parameters, they had to create dataset by mounting three GoPro action cameras on a Segway. They demonstrate the ability to generate environmental awareness with additional perceptual modules, including DNN-based object detection and obstacle detection via monocular visual odometry. In a similar work, A. Luquercio et al. [65] propose a supervised learning based convolutional neural network, called DroNet, for autonomous navigation and collision avoidance problem in unstructured, highly dynamic, urban environment. Images from a forward-looking on-board camera were processed and converted into steering commands enabling the UAV to safely navigate while avoiding obstacles. To learn steering angles from images, the authors

used Udacity's project [66] dataset taken while driving in a car, while learning collision detection, was achieved from images created by the authors by mounting a GoPro camera on a bicycle and ridding around by a bicycle. Testing the idea of caching in the sky, the M. Chen et al. [67] propose, a proactive, conceptor-based ESNs to compute the deployment location of cache enabled UAVs, to maximize the quality of experience (QoE) for the wireless devices in a cloud radio access network.

2.1.3 Challenges

2.1.3.1 Machine Learning in Dynamic Spectrum Access System

Dynamic spectrum access (DSA) system may require the information of wireless networks, such as the location of incumbent users and IoT devices, the channel coefficient between the users: receivers of incumbent users and transmitters of IoT devices, receivers and transmitters of IoT devices, receivers of IoT devices and transmitters of incumbent users. If the system can achieve all those information, DSA of IoT devices would be perfect; they can transmit with no harm to incumbent users and another IoT devices. But in reality, the system cannot collect all the location of incumbent users; due to security issues when the incumbent user belongs to national army, or privacy issues when the incumbent user is customer of mobile network operator, as well as channel coefficients. Furthermore, the location of IoT devices cannot be known because the installation of devices is random. To support the DSA of IoT devices, the system needs to capture the spectrum access opportunity with imperfect information.

2.1.3.2 On-device Distributed Machine Learning

Decentralized, distributed learning by mobile devices is welcomed nowadays for realizing customized service, private data protection, or robustness to unexpected events in communication protocol (e.g., losing network connectivity). Unfortunately, our mobile devices have a problem in locally deploying an onerous model within a short time limit though powerful high-dimensional models are already developed to perform complicated tasks. Due to the limited computational resource and transmission power, an edge device is not capable of running an extensive network. This incapability obstructs extracting complex features from each local user. Furthermore, a small network that a device can barely manage is more likely to fail in generalizing the given data if its training dataset lacks part of the subsections or class labels. One possible solution is asking for the supplement of deficient data samples, but clearly, revealing the bias of each device harms the dataset distribution security.

Not only the performance but communication efficiency impedes reaching an efficient on-device machine learning system with numerous participating devices. For instance, federated learning requires many rounds of communication between the central server and devices. In case of instant usage such as achieving low-latency in a certain system, it is more important to learn a reasonable model as fast as possible than finding an optimal model. However, it is inevitable to sacrifice the peak performance of the global model by choosing on-device learning.

2.1.3.3 Multi-agent Learning for Networking

Multi-agent systems often require a certain level of coordination to complete a task efficiently. Although it is possible to instill cooperation among multiple agents without communication, a natural solution is to let agents communicate with each other. Assuming the availability of communication is a fair in most cases, and intuitively, agents are more predictable when they communicate, thus arguably a more stable approach.

Even though communication seems to be straight-forward, a series of different challenges may still occur in addition to the original coordination problem. First, the agents must decide what sort of information should be communicated. In theory, agents can naively send their raw observation to each other so that each agent is informed of the global status. However, this is impractical due to bandwidth limitations or

communication costs. Some data are also not useful to other agents, thus adding unnecessary noise to the transmitted information. The challenge now is how to determine which information is significant to the task, and which are crucial to forming cooperative strategies. In addition, agents must be able to summarize the data into a message that satisfies the bandwidth constraints, as well as be able to decode the message so that other agents can act on it. Second, it is generally assumed that agents communicate over a shared medium. Thus, agents need to efficiently schedule their communication. This is particularly important for tasks that are time-sensitive, in which the delay in transmission may seriously affect the overall performance of the agents. Therefore, agents may need to slightly deviate from the traditional medium access contention approaches and intelligently schedule their communication. The challenge may be translated into a prioritization problem, where agents must be able to consider the urgency of their messages. Third, agents must be able to synchronize their learning so that the changes in one agent's policy does not affect others negatively. This is known as the non-stationarity problem. Although agents can communicate, the communicated information may not matter at all if the agents cannot coordinate the adjustments to their policies.

2.1.3.4 UAV

UAVs are expected to be one of the building blocks of 5G networks to support the increased number of networks connected IoT devices. In the context of wireless communication systems, UAVs can assume three key functions: as UEs to support real time video streaming and IoT applications, as relay stations to provide single hop LOS or multi-hop backhaul links and as aerial RBSs to provide on demand short term coverage in areas that may be experiencing coverage disruption, and to provide capacity boost in hotspots or other areas experiencing an occasional surge of capacity demand. UAVs can also be used to support different types of applications, e.g., cache-enabled UAVs, working as edge compute nodes, can directly send the contents that are stored at the cache to the users that request the cached contents to offload data traffic from the core network. To support these functions, UAVs must be able to autonomously and safely navigate, self-organize, optimize their flying path, optimize their compute and energy efficiency while minimizing the potential interference with terrestrial RBS and UEs.

2.2 PriMO-5G Innovation

This section presents the various studies from PriMO-5G partners regarding AI-assisted networking. In this section, we present published and ongoing works that aim to address problems in the wireless networking, privacy-preserving distributed machine learning, multi-agent learning with communication, UAV networking, and caching.

2.2.1 DSA with Reinforcement Learning for IoT Networks

2.2.1.1 Introduction

The advance of wireless communication technologies promotes a machine-to-machine (M2M) communication service, which demands on-line connection for every single machine. Internet-of-Things (IoT) technology is a candidate technology for enabling real-time Internet-based wireless communications of M2M applications. To support the massive connectivity of IoT devices, a tremendous quantity of spectrum bands is needed. Dynamic spectrum access (DSA) is one of the promising technologies to tackle the spectrum band scarcity of implementing IoT networks. The DSA-aided IoT network utilizes the spectrum bandwidth more efficiently by acquiring under-utilized spectrum bands, which are not used by incumbent users. To capture spectrum access opportunity, spectrum sensing capability is fundamental. In this context, a dedicated spectrum sensor is needed to support the IoT devices that do not have spectrum sensing capability due to small form factor. After the spectrum sensors investigate the spectrum bands, a central unit determines which IoT devices transmit or not. However, there is no prior information about the location of IoT devices as well as incumbent users. The central unit has to utilize effective method that can handle

stochastic situation. In this context, reinforcement learning (RL) is a key enabler for designating the transmission of IoT devices. The central unit interacts with the environment, i.e. the wireless communication channel through controlling the spectrum access of IoT devices. Since the numerical reward has stochastic characteristics, the central unit has to repeat trials based on the RL framework for finding proper access probability of IoT devices.

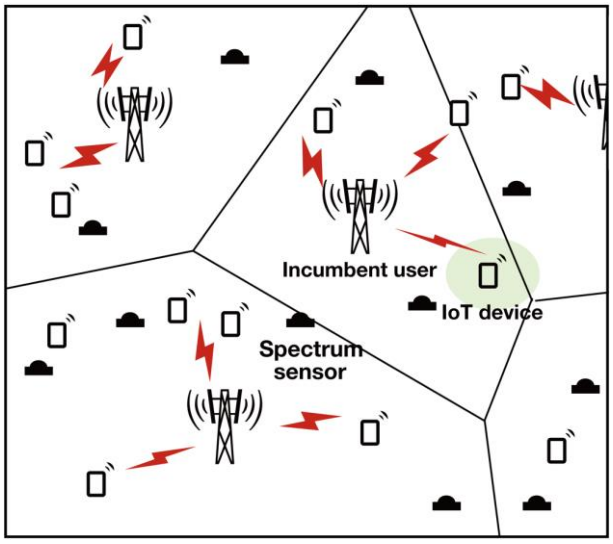


Figure 2-2: A system model of the IoT network utilizing spectrum sensor-aided dynamic spectrum access framework.

2.2.1.2 Sensor-Aided IoT Network

2.2.1.2.1 Network Model

Consider a sensor-aided IoT network that contains IoT devices with dedicated spectrum sensors. We refer this network as secondary network. Also, we refer the incumbent network as primary network. The spectrum sensors give information to the central unit, which aggregates interference from primary transmitters (PTXs). With this information, the central unit learns the access probability of secondary transmitters (STXs). After finishing the learning procedure, the central unit informs the access probability to STXs for ALOHA transmission.

2.2.1.3 Reinforcement Learning Procedure

2.2.1.3.1 The Central Unit

The central unit interacts with the environment as presented in Figure 2-3. The central unit poses an action by controlling the spectrum access of STXs and gathering the transmission results of secondary pairs as a reward. With these results, the central unit calculates the benefit of transmission of each STX. After that, the central unit adjusts the access probabilities of STXs that produce higher area spectral efficiency (ASE) value. The central unit utilizes RL algorithm, which has an advantage for handling stochastic environments.

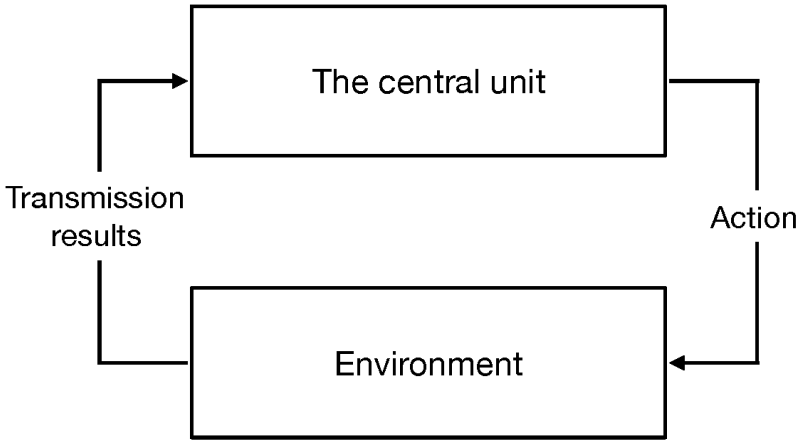


Figure 2-3: A reinforcement learning system.

2.2.1.3.2 Proposed Reinforcement Learning Procedure

The proposed reinforcement learning procedure is comprised as follows:

(a) Spectrum sensing: In this period, the central unit collects the interference value for calculating initial access probability of STXs. Sensors nearest to each STX inform the aggregate interference value to the central unit. We denote the measured aggregate interference by sensor as S_k.

(b) Initialization: The central unit initializes the access probability and the internal state vector. The central unit determines initial access probability $\mathbf{p}(0)$ of STXs with the interference level S_k measured by k^{th} sensors, where $k = 1,2,\cdots,N$. Then the central unit calculates the initial internal state values $\mathbf{w}(0)$ with $\mathbf{p}(0)$.

(c) Access probability learning: In this period, the central unit interacts with the environment, i.e., wireless network utilizing REINFORCE learning algorithm. The central unit controls the transmission of STXs along the time step $t = 1,2,\cdots,T_l$. After the transmission of STXs, the central unit collects the transmission results of STXs to assess the benefit of transmission of each STX. The access probability learning procedure of the central unit is described in Algorithm 1. Note that $\mathrm{Bernoulli}(\mathbf{p})$ is one with probability $\mathbf{p}$, otherwise zero.

(d) Determining the final access probability: After REINFORCE learning algorithm is done, the central unit determines the final access probability of STXs $\mathbf{p}_l$ as follows:

$$\mathbf{p}_l = \mathbf{p}(T_l). \quad (1)$$

Note that the access probabilities of STXs converge to one or zero for a large enough T_l. The proof about convergence behavior of access probabilities is represented in [2]. We utilize this behavior for determining access probability of STXs in the learning procedure. With the final access probability, k^{th} STXs transmit its packet according to slotted ALOHA protocol with access probability p_{lk}. In the next section, we evaluate the performance of the proposed spectrum access procedure.

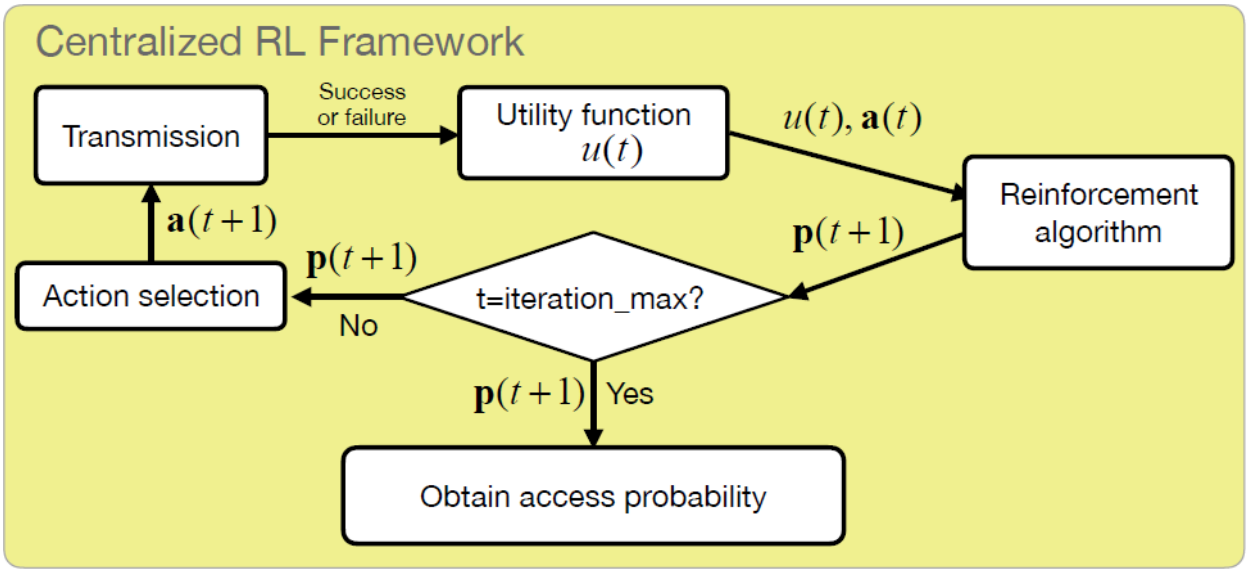


Figure 2-4: A reinforcement learning procedure conducted by central unit.

2.2.1.4 Simulation Results

2.2.1.4.1 General Settings

Consider a network within a square of length L_{net} that is set as 50 m. Unless it is mentioned otherwise, the density of sensors λ_{ssr} is identical in all simulation cases as 0.24. The STXs are located in the network randomly with the number of STXs being 20, 30, 40, or 50. The pair distance of secondary users d is 3 m. The secondary users use QAM constellation, i.e. the target SINR threshold of SRX is 3 dB. We obtain average ASE of secondary users by executing slotted ALOHA transmission with the final access probability repeating 100,000 times. We compare the performance of proposed RL based DSA system with a reference system with respect to average ASE. We implement the reference system with carrier sense multiple access with collision avoidance (CSMA/CA) protocol. We set the carrier sensing range of CSMA/CA protocol as 6 m. The initial access probability of STXs is determined by means of Sense-and-Predict (SaP) method. The interference level at STXs is different from that of sensors due to location difference. When it comes to using sensor value to obtain spectrum access probability of STXs, difference of interference level hinders accurate decision of this probability. The SaP method aims to overcome this difference by predicting the interference level at STXs using spatial correlation of interference. With the predicted interference level, the SaP method provides successful transmission probability of STXs by means of stochastic geometry approach. We refer successful transmission as opportunistic probability (OP).

2.2.1.4.2 Average Area Spectral Efficiency Validation

In Figure 2-5, we evaluate the performance of RL based DSA system in various STX densities. As the density of STXs is increasing, the average ASE is improved when the learning procedure is done. This is because more STXs are located in sparse areas far away from PTXs. These STXs just consider the other STXs around themselves, so more STXs freely communicate with their paired SRXs. After T_l has a value of around 5,000, the performance enhancement slows down in every simulation case. So we need to choose an appropriate T_l to balance the trade-off between learning cost and performance gain.

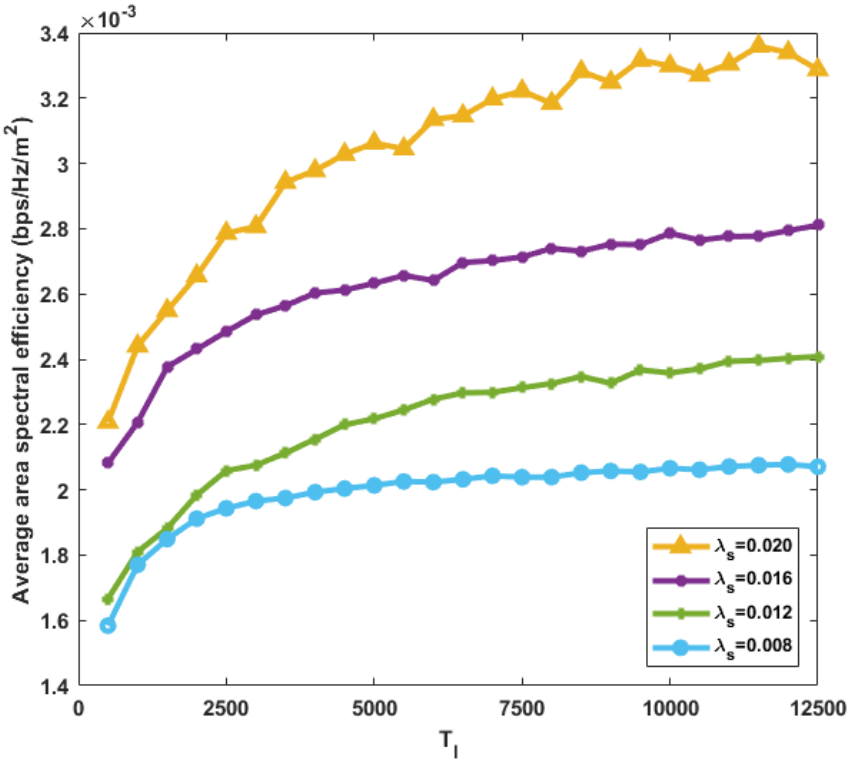


Figure 2-5 The average area spectral efficiencies of various STX densities

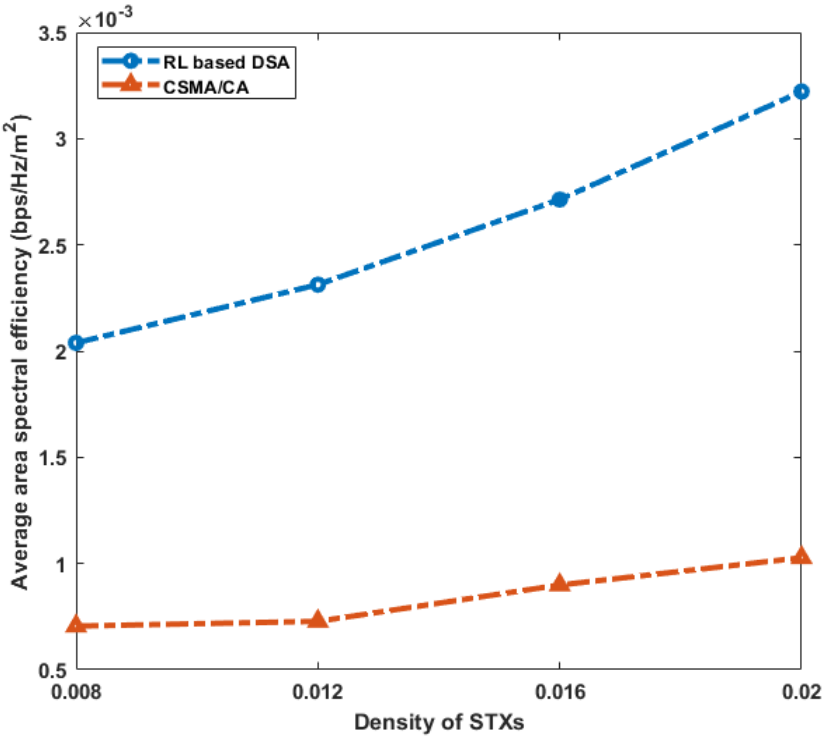


Figure 2-6: Performance comparison between proposed RL based DSA system and CSMA/CA.

In Figure 2-6, the performance of RL based DSA system surpasses that of CSMA/CA as we mentioned before. If STXs utilize CSMA/CA protocol, STXs located in sparse area lose transmission opportunity while they wait for CCA procedure. In the RL based DSA system, STXs located in sparse area have final access probability close to 1, which captures spatial transmission opportunity. The STXs that have near-to-one final access probability transmit their packet whenever transmission is needed. Therefore, the performance of RL based DSA system surpasses that of CSMA/CA protocol.

2.2.1.5 Conclusion

In this section, we proposed an RL based DSA system that aims at efficient spectrum usage for IoT networks. The limitation of IoT device is presented and the architecture of sensor-aided DSA IoT network is described. The main objective of RL based DSA system is enhancing the spatial spectrum reusability of IoT network, which is evaluated by ASE. We investigate the performance of the proposed RL based DSA system in various densities of IoT devices. As the learning period gets longer, the performance of IoT network has more performance enhancement, but gain is marginal after a certain period. We show that the impact of an IoT network to the incumbent network is marginal with respect to average spectral efficiency of the incumbent network. However, as the learning period gets longer, the performance of the incumbent network slightly degrades. As a future research topic, we have to select a proper learning period so as to reduce the impact on the incumbent network as well as the learning cost to achieve marginal performance enhancement of IoT network. Also, the system architecture when there is no dedicated control channel has to be considered to implement the RL based DSA system in real wireless communication networks.

2.2.2 Communication-Efficient On-Device Machine Learning

2.2.2.1 Introduction

What if devices such as phones, drones, or vehicles each train their local models in collaboration with others? Localizing AI into the edge devices has essential benefits regarding customized service and network bandwidth conservation. Notably, the key advantage of running machine learning on local devices is that numerous user-generated data samples can be used.

Unfortunately, edge devices cannot afford high communication cost that is proportional to the model sizes. As those devices are confined to the use of small-sized models, it is crucial to minimize the inter-device communication overhead. With this end, federated distillation (FD) is suggested – a distributed online knowledge distillation method whose communication payload size depends not on the model size but the output dimension.

Meanwhile, each device is likely to have a biased training dataset, because some of the class labels are deficient in data samples. To cope with this with much smaller communication cost, we suggest federated augmentation (FAug), a data augmentation algorithm using a generative model.

2.2.2.2 Federated distillation

Conventional distributed training algorithms exchange local model parameters every epoch. To reduce the communication cost among wirelessly interconnected devices, the devices can exchange model parameters at intervals. Federated learning, one of the most representative distributed learning methods, lets each device train with its local data samples and provide its model parameters periodically to build a global model. However, the drawback of federated learning lies in the requirement of identical model structure. If the models of participants are composed of heterogeneous structure, it is impossible to share parameters.

Meanwhile, co-distillation accepts devices with different model structure to cooperate. Each device exploits a shared training dataset for local training, considering itself as a student, and the average of the others as a teacher. A student derives its own predicted output (logit vector) and compares it with that of teacher's. To decide the referring rate, the student regularizes a parameter called temperature. As it requires synchronized training dataset for all devices, users should exchange their outputs more often as the training dataset size increases.

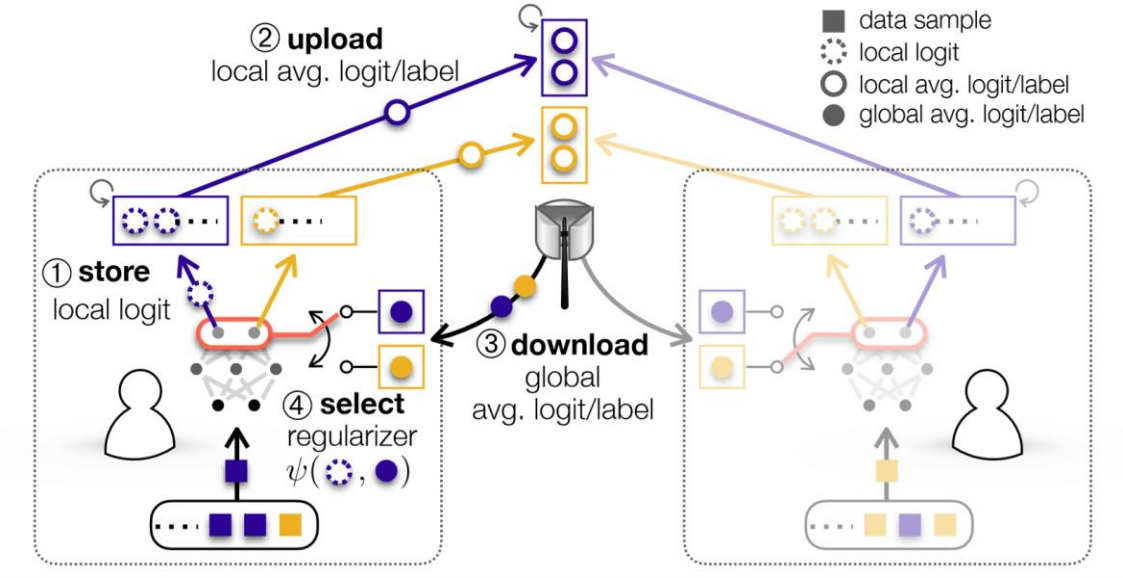


Figure 2-7: Federated distillation with 2 devices and 2 labels

Each device in federated distillation updates local weights in a similar way as co-distillation but derives mean logit vectors according to each label. For the compact exchange required by federated learning, they periodically upload these local-average logit vectors to a server. Then the server sums what it received from all devices and finds the global-average logit vector so that each device can refer to it for the next iteration.

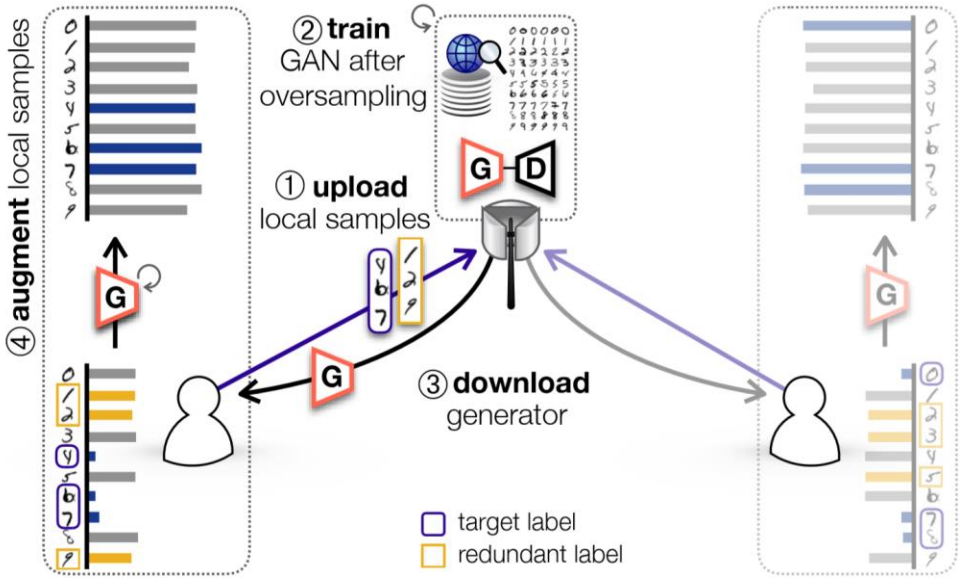


Figure 2-8: Federated augmentation with 3 target and 3 redundant MNIST labels.

2.2.2.3 Federated augmentation

If we implant the missing local data samples according to each sample's needs, the communication overhead sharply increases, particularly in case of large number of devices. Instead of directly exchanging the local data samples, a local device can generate its missing data samples using a generative model, which is named federated augmentation.

Conditional generative adversarial network (cGAN) Conditional GAN consists of two network models: a generator and a discriminator. A GAN-based system trains its generator and discriminator alternately. The objective of the generator is to produce plausible samples, while a discriminator is on the adversarial side trying to classify whether the data is original or made by the generator.

Assume that the local devices willing to participate in federated augmentation have less samples with specific labels, i.e. each device has its own target labels. To let the server be informed of the request for samples with target labels, they upload a part of their data samples with target labels to the server via wireless networks. Then the server oversamples them based on its powerful connection to the internet and trains a conditional GAN. When the GAN from the server is completely trained, the users download the generative model customized to their demand of replenishing the target labels. With this generative model, each user can generate synthetic data samples until its dataset becomes evenly distributed.

Meanwhile, privacy leakage occurs both during the uploading and downloading process due to the trading of local samples. To dilute the samples-per-label distribution, the device sends additional samples from the labels that are already abundant in samples, referred to as redundant labels. The device-server privacy leakage is measured as the ratio of the target labels to the total submitted labels.

Indeed, a device can infer the others' target labels by identifying the generable labels of its downloaded generator. This privacy leakage is quantified inter-device privacy leakage – the ratio of target labels to the generable labels. As the number of devices is sufficiently large, the inter-device privacy leakage reaches to the minimum value no matter how many target or redundant labels the devices submit.

2.2.2.4 Evaluation

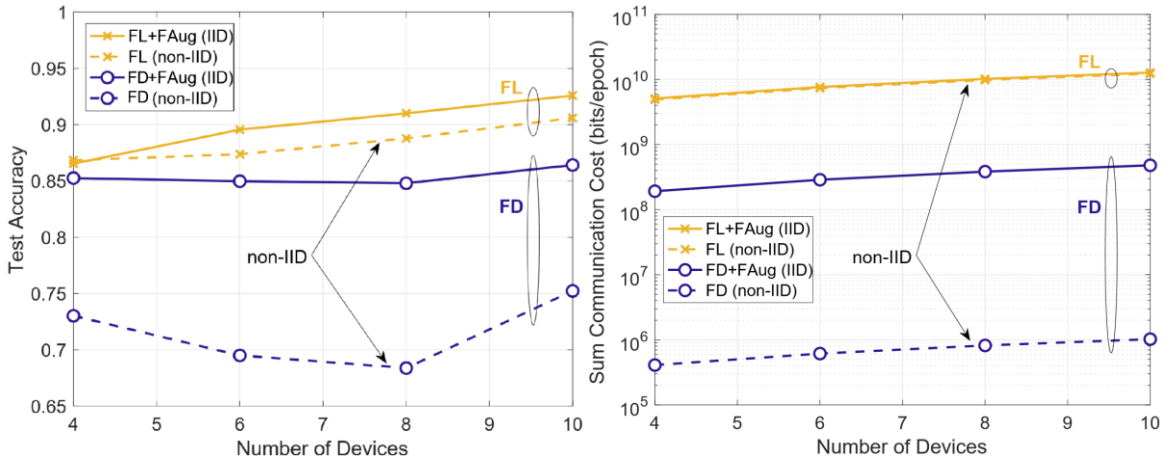


Figure 2-9. Test accuracy[1] and communication cost under a non-IID MNIST dataset: (left) test accuracy under federated learning and federated distillation in both IID and non-IID dataset cases; (right) communication cost in log scale under federated learning and federated distillation in both IID and non-IID dataset cases.

Both in federated learning and federated distillation, federated augmentation helps the participants to recover low per-label test accuracy and therefore achieve even datasets. With federated distillation, we can achieve about 25 times smaller amount of communication cost, while slightly compromising the test accuracy. The reduced accuracy of federated distillation can be complemented by federated augmentation without incurring severe communication overhead. In return for the communication overhead, federated learning is more robust against the non-IID dataset.

For the target label, the standalone training of the reference device under the original non-IID dataset yields 3.585% test accuracy, which is increased via federated augmentation with federated distillation or federated learning by 73.44% or 92.19%, validating the effectiveness of federated augmentation in combination with both federated distillation and federated learning.

2.2.2.5 Concluding remarks

Towards enabling on-device machine learning, we introduced federated distillation and federated augmentation that are communication-efficient training and data augmentation algorithms, respectively. Empirical studies showed their effectiveness that achieves comparably high accuracy with much smaller communication overhead compared to federated learning.

[1] Test accuracy is the ratio of the number of data samples that the model correctly classified to whole samples in the device's test dataset. Communication cost is the number of bits transmitted during uplink or downlink, which corresponds to the payload size of the exchanged information.

2.2.3 Learning to Schedule Communication in Multi-Agent Reinforcement Learning

2.2.3.1 Introduction

When having multiple agents (e.g. robots, UAV) in the field, a significant amount of coordination among these agents is expected in order to achieve an efficient completion of tasks. Furthermore, these agents usually have limited observations, caused by their sensor range and environmental obstacles. Communication plays a significant role in addressing these problems by allowing agents to share information, which will allow them to plan well-coordinated strategies.

In practical scenarios where agents are typically separated but are able to communicate over a shared medium, e.g., a frequency channel in wireless communications, two important constraints are imposed: bandwidth and contention for medium access [68]. The bandwidth constraint entails a limited amount of bits per unit time, and the contention constraint involves avoiding collision among multiple transmissions due to the natural aspect of signal broadcasting in wireless communication. Thus, only a restricted number of agents are allowed to transmit their messages each time step for a reliable message transfer. In this research, we use a simple model to incorporate that the aggregate information size per time step is limited by L_{band} bits and that only K_{sched} out of n agents may broadcast their messages.

We formulate the scenario as a multi-agent sequential decision-making problem, where the agents' goal is to maximize a common discounted reward. With that, we use the multi-agent reinforcement learning (MARL) framework to train the agents. We propose a new deep MARL architecture, called *SchedNet*, with the rationale of centralized training and distributed execution (CTDE) in order to efficiently achieve a common goal via decentralized cooperation. During distributed execution, agents are allowed to communicate over wireless channels where messages are broadcast to all agents in each agent's communication range. This broadcasting feature of wireless communication needs a Medium Access Control (MAC) protocol to arbitrate contending communicators in a shared medium. CSMA (Collision Sense Multiple Access) in Wi-Fi is one such MAC protocol. While prior work on MARL to date considers only the limited bandwidth constraint (e.g. [48]), we additionally address the shared medium contention issue in what we believe is the first work of its kind: which nodes are granted access to the shared medium. Intuitively, nodes with more important observations should be chosen, for which we adopt a simple yet powerful mechanism called weight-based scheduler, designed to reconcile simplicity in training with integrity of reflecting real-world MAC protocols in use (e.g., 802.11 Wi-Fi).

We comment that SchedNet is not intended for competing with other algorithms for cooperative multi-agent tasks without considering scheduling, but a complementary one. We believe that adding our idea of agent scheduling makes those algorithms much more practical and valuable.

2.2.3.2 Methods

2.2.3.2.1 Weight-based Scheduling

We consider a simple mechanism to schedule K_{sched} agents in distributed manner. We adopt a weight-based design, called *WSA* (Weight-based Scheduling Algorithm), in which each agent decides its own weight. A pre-defined rule that is conditioned on the agents' weights is used to schedule the agents. Due to their simplicity and good approximation of practical wireless scheduling protocols, we utilized the following scheduling rules:

- *Top(k)*. Selecting top k agents in terms of their weight values.

- *Softmax(k)*. Computing softmax values $\sigma_i(\boldsymbol{w}) = \frac{e^{w_i}}{\sum_{j=1}^{n} e^{w_i}}$ for each agent i, and then randomly selecting k agents according to the probability distribution $[\sigma_i(\boldsymbol{w})]_{i=1}^{n}$.

Since distributed execution is one of our major operational constraints in SchedNet or other CTDE-based MARL algorithms, Top(k) and Softmax(k) should be realizable via a weight-based mechanism in a distributed manner. In fact, this has been an active research topic to date in wireless networking, where many algorithms exist [69],[70],[71]. Using so-called CSMA [72], which is a fully distributed MAC scheduler and forms a basis of Wi-Fi, given agents' weight values, it is possible to implement Top(k) and Softmax(k). Our goal is to train agents so that every time each agent takes an action, only K_{sched} agents can broadcast their messages with limited size L_{band} with the goal of receiving the highest cumulative reward via cooperation. Each agent should determine a policy described by its scheduling weights, encoded communication messages, and actions.

2.2.3.2.2 Architecture

To this end, we propose a new deep MARL framework with scheduled communications, called SchedNet, whose overall architecture is depicted in Figure 2-10. SchedNet consists of the following three components: (i) actor network, (ii) scheduler, and (iii) critic network.

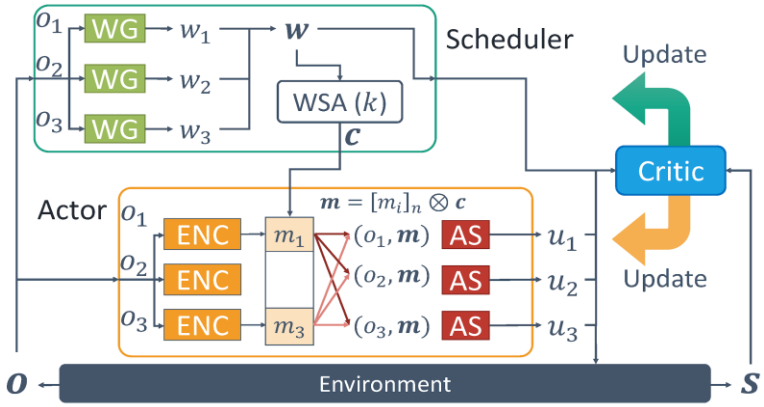


Figure 2-10: Architecture of SchedNet with three agents. Agents 1 and 3 have been scheduled for this time step.

Neural Networks. The actor network is the collection of n per-agent individual actor networks, where each agent i's individual actor network consists of a triple of the following networks: a message encoder, an action selector, and a weight generator, as specified by:

- message encoder $f_{enc}^{i}: i_i \mapsto m_i$,
- action selector $f_{as}^{i}: (o_i, \boldsymbol{m} \otimes \boldsymbol{c}) \mapsto u_i$,
- weight generator $f_{wg}^{i}: o_i \mapsto w_i$.

Here, $\boldsymbol{m} = [m_i]_n$ is the vector of each i's encoded message m_i. An agent schedule vector $\boldsymbol{c} = [c_i]_n, c_i \in \{0,1\}$ represents whether each agent is scheduled. Note that agent i's encoded message m_i is generated by a neural network $f_{enc}^{i}: i_i \mapsto m_i$. The operator "$\otimes$" concatenates all the scheduled agents' messages. For example, for $\boldsymbol{m} = [010, 111, 101]$ and $\boldsymbol{c} = [110]$, $\boldsymbol{m} \otimes \boldsymbol{c} = 010111$. This concatenation with the schedule profile $\boldsymbol{c}$ means that only those agents scheduled in $\boldsymbol{c}$ may broadcast their messages to all other agents.

We denote by θ_{as}^{i} , θ_{wg}^{i}, and θ_{enc}^{i} the parameters of the action selector, the weight generator, and the encoder of agent i, respectively, where we let $\boldsymbol{\theta}_{as} = [\theta_{as}^{i}]\_n$, and similarly define $\boldsymbol{\theta}_{wg}$ and $\boldsymbol{\theta}_{enc}$.

Coupling: Actor and Scheduler. Encoder, weight generator and the scheduler are the modules for handling the constraints of limited bandwidth and shared medium access. Their common goal is to learn the state-dependent "importance" of individual agent's observation, encoders for generating compressed messages and the scheduler for being used as a basis of an external scheduling mechanism based on the weights generated by per-agent weight generators. These three modules work together to smartly respond to time-varying states. The action selector is trained to decode the incoming message, and consequently, to take a good action for maximizing the reward. At every time step, the schedule profile $\boldsymbol{c}$ varies depending on the observation of each agent, so the incoming message $\boldsymbol{m}$ comes from a different combination of agents. Since the agents can be heterogeneous and they have their own encoder, the action selector must be able to make sense of incoming messages from different senders. However, the weight generator's policy changes, the distribution of incoming messages also changes, which is in turn affected by the pre-defined WSA. Thus, the action selector should adjust to this changed scheduling. This also affects the encoder in turn. The updates of the encoder and the action selector trigger the update of the scheduler again. Hence, weight generators, message encoders, and action selectors are strongly coupled with dependence on a specific WSA, and we train those three networks at the same time with a common critic.

Scheduling logic. The schedule profile $\boldsymbol{c}$ is determined by the WSA module, which is mathematically a mapping from all agents' weights $\boldsymbol{w}$ (generated by f_{wg}^{i}) to $\boldsymbol{c}$. Typical examples of these mappings are Top(k) and Softmax(k), as mentioned above. The scheduler of each agent is trained appropriately depending on the employed WSA algorithm.

2.2.3.2.3 Training and Execution

In the centralized training with distributed execution, for a given WSA, we include all components and modules in Figure 2-11 to search for $\boldsymbol{\theta}_{as}$, $\boldsymbol{\theta}_{wg}$, and $\boldsymbol{\theta}_{enc}$, whereas in execution, each agent i runs a certain shared medium access mechanism, well-modeled by a weightbased scheduler, and just needs three agent-specific parameters θ_{as}^{i} , θ_{wg}^{i}, and θ_{enc}^{i}.

Centralized critic. The actor is trained by dividing it into two parts: (i) message encoders and action selectors, and (ii) weight generators. This partitioning is motivated by the fact that it is hard to update both parts with one backpropagation since WSA is not differentiable. To update the actor, we use a centralized critic parametrized by θ_c to estimate the state value function $V_{\theta_c}(\boldsymbol{s})$ for the action selectors and message encoders, and the action-value function $Q_{\theta_c}^{\pi}(\boldsymbol{s}, \boldsymbol{w})$ for the weight generators. The critic is used only when training, and it can use the global state s, which includes the observation of all agents. All networks in the actor are trained with gradient-based on temporal difference backups. To share common features between $V_{\theta_c}(\boldsymbol{s})$ and $Q_{\theta_c}^{\pi}(\boldsymbol{s}, \boldsymbol{w})$ and perform efficient training, we use shared parameters in the lower layers of the neural network between the two functions, as shown in Figure 2-11.

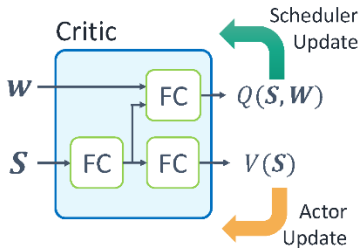


Figure 2-11: Architecture of the critic. FC stands for fully connected neural network.

Weight generators. We consider the collection of all agents' WGs as a single neural network $\mu_{\boldsymbol{\theta}_{wg}}(\boldsymbol{o})$ mapping from $\boldsymbol{o}$ to $\boldsymbol{w}$, parametrized by θ_{wg}. Noting that w_i is a continuous value, we apply the DDPG algorithm [58], where the entire policy gradient of the collection of WGs is given by:

Equation 2-1: $\nabla_{\boldsymbol{\theta}_{wg}} J(\boldsymbol{\theta}_{wg}, \cdot) = E_{\boldsymbol{w}\sim\mu_{\boldsymbol{\theta}_{wg}}}\left[\nabla_{\boldsymbol{\theta}_{wg}} \mu_{\theta_{wg}}(\boldsymbol{o}) \nabla_{\boldsymbol{w}} Q_{\theta_c}(\boldsymbol{s}, \boldsymbol{w}) \mid w = \mu_{\theta_{wg}}(\boldsymbol{o})\right]$

We sample the policy gradient for sufficient amount of experience in the set of all scheduling profiles, i.e., $C = \{c \mid \sum_{c_i} \leq k\}$. The values of $Q_{\theta_c}(\boldsymbol{s}, \boldsymbol{w})$ are estimated by the centralized critic, where $\boldsymbol{s}$ is the global state corresponding to $\boldsymbol{o}$ in a sample.

Message encoders and action selectors. The observation of each agent travels through the encoder and the action selector. We thus serialize f^i_{enc} and f^i_{as} together and merge the encoders and actions selectors of all agents into one aggregate network $\pi_{(\theta_u)}(\boldsymbol{u}|\boldsymbol{o}, \boldsymbol{c})$, which is parametrized by $\theta_u = \{\theta_{enc}, \theta_{as}\}$. This aggregate network $\pi\_(\theta_u)$ learns via backpropagation of actor-critic policy gradients, described below. The gradient of this objective function is given by

Equation 2-2: $\nabla_{\boldsymbol{\theta}_u} J(\cdot, \boldsymbol{\theta}_u) = E_{\boldsymbol{s}\sim\rho^{\pi}, \boldsymbol{u}\sim\pi_{\theta_u}}\left[\nabla_{\boldsymbol{\theta}_u} \log \pi_{\theta_u}(\boldsymbol{u}|\boldsymbol{o}, \boldsymbol{c})[r + \gamma V_{\theta_c}(\boldsymbol{s}') - V_{\theta_c}(\boldsymbol{s})]\right]$,

where $\boldsymbol{s}$ and $\boldsymbol{s}'$ are the global states corresponding to the observations at current and next time step. We can get the value of state $V_{\theta_c}(\boldsymbol{s})$ from the centralized critic and then adjust the parameters θ_u via gradient ascent accordingly.

Execution. In execution, each agent i should be able to determine the scheduling weight w_i, encoded message m_i, and action selection u_i in a distributed manner. This process must be based on its own observation, and the weights generated by its own action selector, message encoder, and weight generator with the parameters θ^i_{as} , θ^i_{enc}, and θ^i_{wg}, respectively. After each agent determines its scheduling weight, Ksched agents are scheduled by WSA, which leads the encoded messages of scheduled agents to be broadcast to all agents. Finally, each agent selects an action by using received messages. This process is sequentially repeated under different observations over time. compare SchedNet with (i) RR (round robin), which is a canonical scheduling method in communication systems where all agents are sequentially scheduled, and (ii) FC (full communication), which is the ideal configuration, wherein all the agents can send their messages without any scheduling or bandwidth constraints. We also diversify the WSA in SchedNet into: (i) Sched-Softmax(1) and (ii) Sched-Top(1) whose details are in Section 3.1. We train our models until convergence, and then evaluate them by averaging metrics for 1,000 iterations. The shaded area in each plot denotes 95% confidence intervals based on 6-10 runs with different seeds.

2.2.3.3 Results and Discussion

We compare SchedNet with a variant of DIAL [48] which allows communication with limited bandwidth. During the execution of DIAL, the limited number (k) of agents are scheduled following a simple round robin scheduling algorithm, and the agent reuses the outdated messages of non-scheduled agents to make a decision on the action to take, which is called DIAL(k). The other baselines are independent DQN (IDQN) [73] and COMA [74] in which no agent is allowed to communicate.

We experimented on the predator/prey environment, where there are multiple agents (predators) whose goal is to capture a randomly moving prey. Agents' observations include position of themselves and the relative positions of prey, if observed. We employ four agents, and they have different observation horizons, where only agent 1 has a 5 × 5 view while agents 2, 3, and 4 have a smaller, 3 × 3 view. The predators are rewarded when they capture the prey, and thus the performance metric is the number of time steps taken to capture the prey.

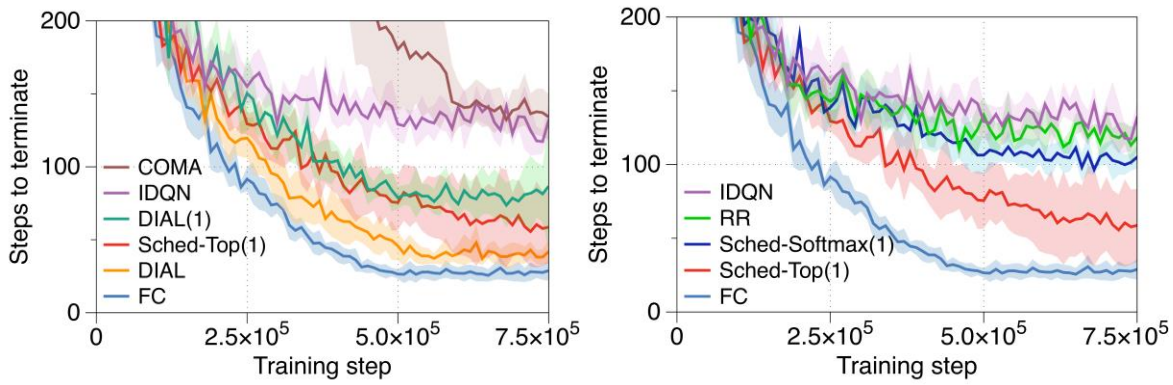


Figure 2-12 Learning curves during the learning of the PP task. The plot shows the average time taken to complete the task, where shorter time is better for agents. (left) Comparison with other baselines; (right) Comparison with scheduling schemes. $k = 1, l = 2$

Figure 2-12 (left) illustrates the learning curve of 750,000 steps in PP. In FC, since the agents can use full state information even during execution, they achieve the best performance. SchedNet outperforms IDQN and COMA in which communication is not allowed. It is observed that agents first find the prey, and then follow it until all other agents also eventually observe the prey. An agent successfully learns to follow the prey after it observes the prey but it takes a long time to meet the prey for the first time. If the agent broadcasts a message that includes the location information of the prey, then other agents can find the prey more quickly. Thus, it is natural that SchedNet and DIAL perform better than IDQN or COMA, because they are trained to work with communication. However, DIAL is not trained for working under medium contention constraints. Although DIAL works well when there is no contention constraints, under the condition where only one agent is scheduled to broadcast the message by a simple scheduling algorithm (i.e., RR), the average number of steps to capture the prey in DIAL(1) is larger than that of SchedNet-Top(1), because the outdated messages of non-scheduled agents is noisy for the agents to decide on actions. Thus, we should consider the scheduling from when we train the agents to make them work in a demanding environment.

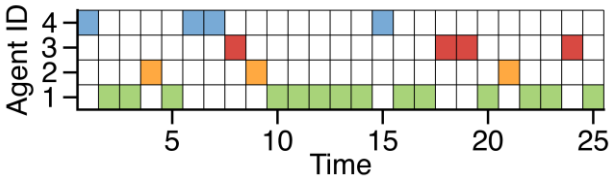


Figure 2-13: Instances of scheduling results over 25 time steps in PP

Impact of intelligent scheduling in Figure 2-12 (right), we observe that IDQN, RR, and SchedNetSoftmax(1) lie more or less on a comparable performance tier, with SchedNet-Softmax(1) as the best in the tier. SchedNet-Top(1) demonstrates a non-negligible gap better than the said tier, implying that a deterministic selection improves the agents' collective rewards the best. In particular, SchedNet-Top(1) improves the performance by 43% compared to RR. Figure 2-12 (right) lets us infer that, while all the agents are trained under the same conditions except for the scheduler, the difference in the scheduler is the sole determining factor for the variation in the performance levels. Thus, ablating away the benefit from smart encoding, the intelligent scheduling element in SchedNet can be accredited with the better performance.

Weight-based Scheduling. We attempt to explain the internal behavior of SchedNet by investigating instances of temporal scheduling profiles obtained during the execution. We observe that SchedNet has learned to schedule those agents with a farther observation horizon, realizing the rationale of importance-based assignment of scheduling priority also for the PP scenario. Recall that Agent 1 has a wider view and thus tends to obtain valuable observation more frequently. In Figure 2-13, we see that scheduling chances are distributed over (14, 3, 4, 4) where corresponding average weights are (0.74, 0.27, 0.26, 0.26), implying that those with greater observation power tend to be scheduled more often.

Message encoding. We now attempt to understand what the predator agents communicate when performing the task. Figure 2-14 shows the projections of the messages onto a 2D plane, which is generated by the scheduled agent under SchedNetTop(1) with $l = 2$. When the agent does not observe the prey (blue dot in the figure), most of the messages reside in the bottom or the left partition of the plot. On the other hand, the messages have large variance when it observes the prey (red 'x'). This is because the agent should transfer more informative messages that implicitly include the location of the prey when it observes the prey.

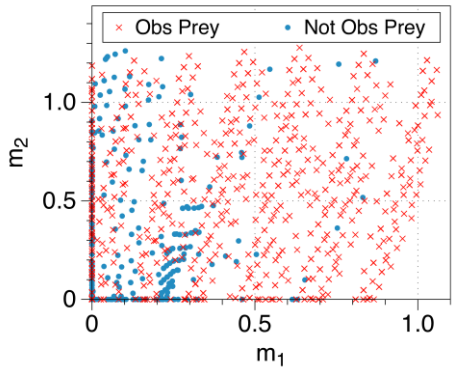


Figure 2-14: Encoded messages projected onto 2D plane in PP

2.2.3.4 Conclusion

We have proposed SchedNet for learning to schedule inter-agent communications in fully cooperative multi-agent tasks. In SchedNet, we have the centralized critic giving feedback to the actor, which consists of message encoders, action selectors, and weight generators of each individual agent. The message encoders and action selectors are criticized towards compressing observations more efficiently and selecting actions that are more rewarding in view of the cooperative task at hand. Meanwhile, the weight generators are criticized such that k agents with apparently more valuable observations are allowed to access the shared medium and broadcast their messages to all other agents. Empirical results and an accompanying ablation study indicate that the learnt encoding and scheduling behavior each significantly improve the agents' performance. We have observed that an intelligent, distributed communication scheduling can aid in a more efficient, coordinated, and rewarding behavior of learning agents in the MARL setting.

2.2.4 AirNet: A Multi-Drone Simulator for Drones, Networking and Reinforcement Learning

2.2.4.1 Introduction

Drones, or Unmanned Aerial Vehicles (UAVs) are gaining a great deal of attention not just from military sector, but also from many different domains including the research community and industry. Drones are already being utilized in many fields such as military, delivery service, aerial photography, agriculture, etc. However, there are applications or missions that cannot be carried out by a single drone due to its limited capability (e.g., battery capacity, coverage area, etc). Situations like these call for a group of drones that can collaborate to complete missions. Multi-drone systems are still in their infancy stage, yet they have a huge potential to expand the horizon of drone applications. To coordinate a fleet of drones seamlessly, sophisticated networking and control algorithms are required

Depending on applications, a multi-drone system's networking scheme could take various forms. Since it is difficult and expensive to try many different network protocols and hardware, researchers often use a network simulator to develop and test behaviors of networks. In order to simulate networks in a multi-drone situation, it is necessary to simulate a drone's dynamics with high fidelity. However, currently available network simulators do not have the capability to simulate a drone's mobility accurately. We aim to cope with this situation by integrating a network simulator and a drone simulator. We developed *AirNet*, a joint simulator that is capable of simulating both networks and a drone's dynamics. We used AirSim [75] as our drone simulator, and OMNeT++[76] as our network simulator. In addition, we introduce AirNet-gym, a framework to develop multi-agent reinforcement learning environments with AirSim.

2.2.4.2 AirNet Architecture

What AirNet does is simulate both complex dynamics of drones and network. To achieve that, we built over two pre-existing simulators, AirSim and OMNeT++, and attempt to integrate them into one system. AirSim is a simulator developed by Microsoft for drones and cars implemented as a plugin for Unreal Engine 4, a popular game engine that offers realistic 3D rendering and accurate physical simulation. OMNeT++ is a C++-based network simulator framework, which gained popularity due to its modularity and extensibility.

In OMNeT++, a network consists of a group of nodes. You can assign a certain mobility pattern to nodes, but its capability is limited that it cannot fully represent a drone's dynamics. AirNet makes 1-to-1 connection between drones in AirSim and nodes in OMNeT++. Positions of drones and their corresponding nodes in OMNeT++ are synchronized. The main challenge here is how to synchronize their positions from each side. Figure 2-15: AirNet Architecture describes the overall architecture of AirNet. The upper plane represents the AirSim side, and the lower one represents the OMNeT++ side. Each drone in AirSim has its corresponding node in OMNeT++, and a node's movement is synchronized with its corresponding drone in AirSim. As per one of our design principles that each side is modular, we designed AirNet in such a way that users of AirNet do not need to care about how to synchronize both sides; each side can be developed separately without considering the other. AirNet has several modules that serve certain functionality. Following sections describe what each module does.

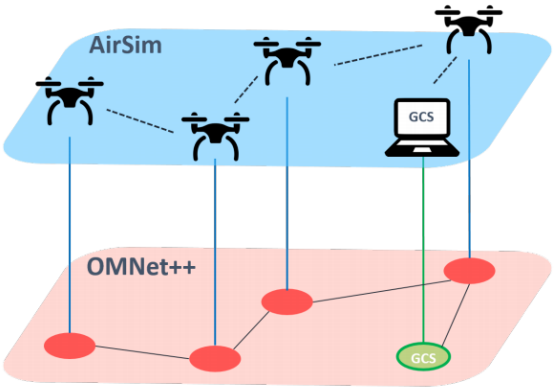


Figure 2-15: AirNet Architecture

Modules on AirSim Side

AirSim exposes APIs for controlling drones and retrieving various kinds of data. AirSim developers can write simulation code using those APIs in any form they want. They could assign a thread that handles each drone, or have a single thread control all the drones. AirNet implements its modules on AirSim side as threads that are assigned to each drone; each drone maintains its own set of modules. AirNet has two modules on AirSim side as follows.

- **Status Publisher**: This module is responsible to report the status of a drone in AirSim to its corresponding node in OMNeT++. Currently a status that is being reported includes a drone's postion, velocity (both in 3D space), and attitude (roll, pitch, and yaw). A status Publisher periodically publishes a drone's status via its TCP socket that is connected to a status receiver on OMNeT++ side. This is a stand-alone module, therefore users can develop and write their code independently of this module.
- **Network Event Sender**: This module is responsible to convey network events that occurs on a drone to its corresponding node in OMNeT++, so that the conveyed network events can be simulated in OMNeT++. When AirNet starts, this module establishes a TCP socket connection with its corresponding node in OMNeT++ and exposes the socket. Developers can use this socket when they want to generate network events.

Modules on OMNeT++ Side

AirNet implements modules on OMNeT++ side using INET framework [77]. INET framework is a library for OMNeT++, which provides various features for communication networks. INET framework has protocols for the Internet stack (UDP, TCP, IPv4, IPv6, etc), as well as wired and wireless link layer (IEEE 802.11, Ethernet, etc). INET also provides mobility models such as circle mobility, linear mobility, random waypoint mobility, etc. Users can extends these models to make a user-defined mobility.

- **Status Receiver:** A status receiver's job is to receive status reports that are sent by a status publisher in AirSim and to synchronize a node's status (position, velocity, etc) with its corresponding drone. This module maintains received status until the next status arrives.
- **AirNet Mobility**: This module is used to move a node's position according to its status that is received by the status receiver module. AirNet mobility module is implemented by extending MovingMobilityBase module that is provided by INET framework.

- **Network Event Actor**: This module receives network events that are sent by the network event sender module on OMNeT++ side. How to handle these network events depends on developers.

Synchronization between AirSim and OMNeT++

Since each simulator runs on a separate process, a form of Inter-Process Communication (IPC) is required to exchange messages between both sides. We chose to use TCP sockets as our IPC method. The main reasons we decided to use TCP socket method is (i) its universality that we can use TCP sockets in almost any environment, and (ii) it readily allows that each simulator runs on different machines. The latter reason is important because it makes AirNet more scalable. As the number of drones that are being simulated increase, more computing resources are required. Having each simulator run on separate machines can allow more computing resources which makes the system more scalable.

To transmit messages via TCP sockets, we need to serialize the messages. AirNet uses MessagePack [78], a binary serialization specification. Its main design goal is to be as simple and compact as possible. MessagePack is faster and more compact than JSON and XML. We chose to use MessagePack because it is important to keep messages compact to achieve better real-time performance. All messages that are exchanged between AirSim and OMNeT++ get serialized and deserialized by MessagePack. Figure 2-16 illustrates the system.

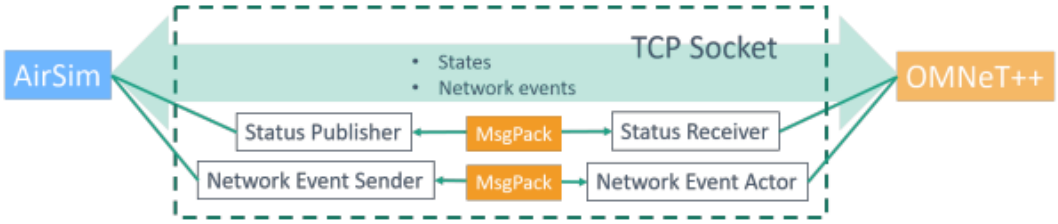


Figure 2-16: Synchronization between AirSim and OMNet++

Ground Control Station

A ground control station (GCS) is a control center that is land-based to control and monitor drones. It provides a visual interface that can track drones' status (position, velocity, etc.) and give commands to drones. Although there are several available opensource GCSs, we decided to develop our own GCS for AirNet. The main reasons are that we need a GCS that is highly customizable, and supports multiple drones. QGroundControl [79] now supports multiple vehicles, but the functionality is limited as its currently available commands are only Pause and Start Mission. We developed a Python-based GCS (Figure 2-17) using PyQt5 [80] and Google Maps API. We used PyQt5 for quick and easy customization of the graphical user interface. For displaying maps, we used Google Maps JavaScript API from Google. Using Google Maps JavaScript API, we load maps and indicate a drone' position in our GCS. Figure 2-17 shows how our GCS looks. On the left side, there are buttons for giving commands. Users can give commands to a certain drone, or all drones. Since the GCS is designed to be highly customizable, the user can easily add custom commands, or remove ones. On the right side, there are drones' status windows display drones' status. Users can monitor a drone's Global Positioning System (GPS) location or battery state. There is a system log windows on the bottom left side. Users can check system messages such as a new drone's joining. Our GCS is provided as a component of AirNet. Users can choose to use our GCS as needed. When our GCS is used, a corresponding GCS node is created in OMNeT++ just like there are corresponding nodes to drones.

Figure 2-17: Ground Control Station.

2.2.4.3 AirNet-gym

We present an extension of AirNet called AirNet-gym, which can be used to implement multi-agent reinforcement learning environments for multi-drone systems. To develop AirNet-gym, we integrated AirSim and OpenAI Gym [81]. Users of AirNet-gym can easily develop a new reinforcement learning environment by setting few parameters and implementing functions that are provided by AirNet-gym.

OpenAI Gym is a toolkit to compare and develop various reinforcement learning algorithms. Developers can make a reinforcement learning environment using the gym library. Environments that are developed using the gym library have a shared interface, so developers can easily write reinforcement learning algorithms that are compatible to the gym library and compare them. The gym library provides a diverse set of environments ranging from simple to complicated ones.

Overall Architecture AirNet-gym is designed to be a framework where users can easily develop their own multi-drone reinforcement learning environments. Figure 2-18 depicts what role AirNet-gym does in developing reinforcement learning environments. AirNet-gym takes AirSim (real or raw environment), and gives a refined environment to a learning agent that is targeted to solve a reinforcement learning problem.

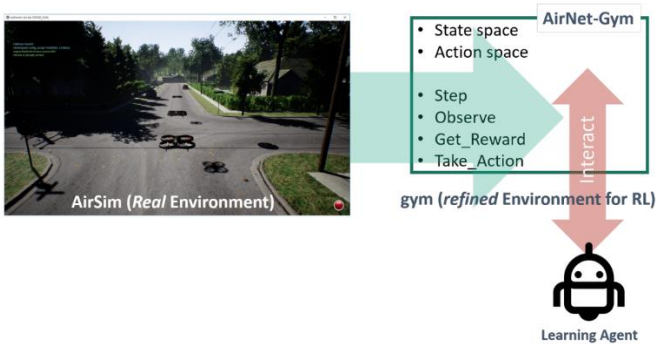


Figure 2-18 AirNet-gym

2.2.4.4 Case Studies

AirNet

To demonstrate AirNet in action, we provide a simple scenario where four drones are connected to the GCS. Each drone is directly connected to the GCS, so they form a star topology with the GCS being in the center. Drones periodically send their status information to the GCS, and GCS sends commands (Take off, Go forward, Land, etc) to drones.

To simulate a scenario, users need to set the initial configuration. This includes the number of drones, their initial positions and GCS's initial position. These initial configurations are done on the AirSim side. In this scenario, there are four drones, one GCS, and their initial positions are set to be around the N1 building in KAIST, where our laboratory is located. When the simulation starts, the status publisher module starts to work. The status publisher module sends each drone's initial status to the status receiver module in OMNeT++, and AirNet mobility module sets each node's initial position according to its received initial status. Then each drone tries to connect to the GCS, and the GCS approves the connection. Notice that users can configure the communication and network scheme without changing other settings because OMNeT++ is designed to be modular.

Figure 2-19 shows how each component of AirNet looks when they are in action. In OMNeT++, which is on the topmost of Figure 2-19, you can see four drones are connected to the GCS. The blue squares represent drones. There is the GCS in the middle of Figure 2-19. In this scenario, giving commands to drones is done by the GCS. Users need to click command buttons to give commands to drones. Figure 2-19 is taken when we gave Random command to all drones. On the bottommost, you can see the AirSim simulator running. You can build the environment as you want, using Unreal Editor. In this scenario we made a simple environment by modifying the default environment given by AirSim.

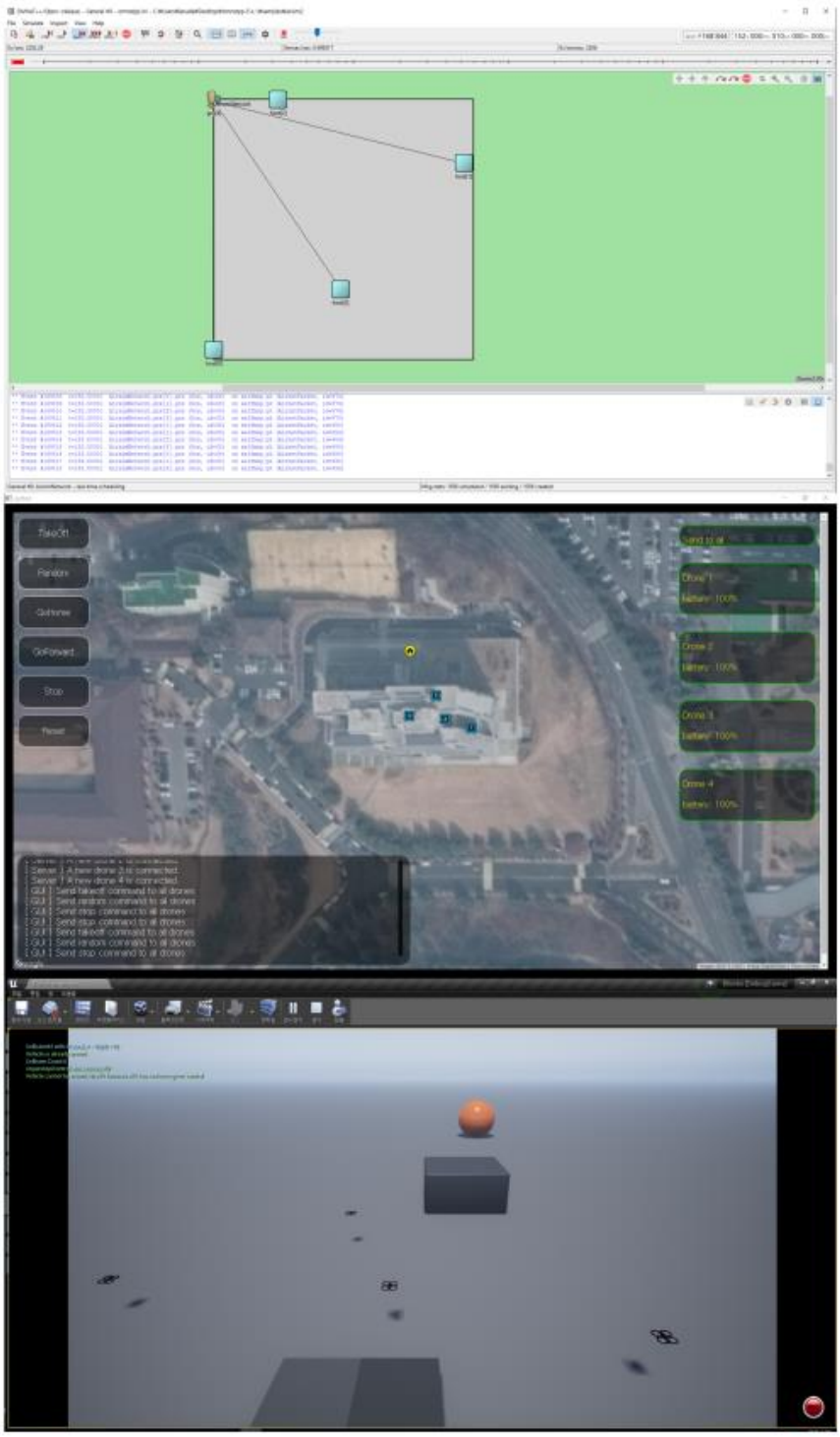


Figure 2-19: AirNet Demonstration

AirNet-gym

To demonstrate AirNet-gym in use, we developed a simple environment using AirNet-gym, called simple occupation. There are two drones and two targets in this scenario, and each drone's job is to occupy the target that is located closer to the drone's initial position. Figure 2-20 (left) is a simplified picture of the scenario. Drones are represented by black drone icons.

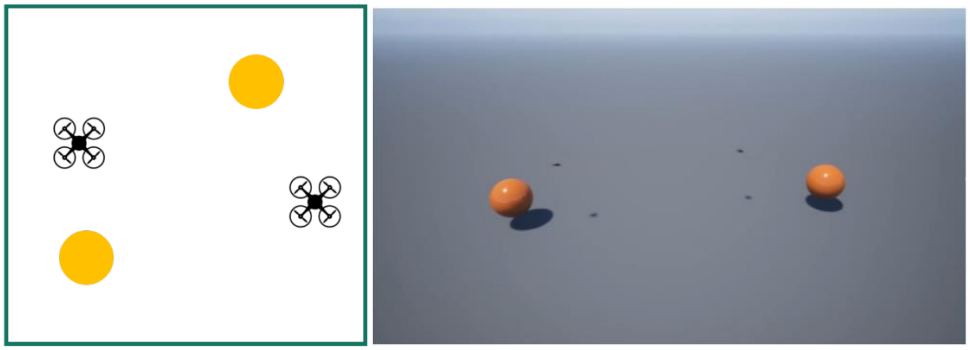


Figure 2-20: (left) Simple occupation task; (right) AirNet-gym view

We compared two different reinforcement learning algorithms upon this environment. First one is independent Deep Q-Network [82], which is inspired by independent Q-learning[83]. Each agent learns independently and agents treat other agents as part of the environment. This approach does not guarantee convergence. The second algorithm we used is Value Decomposition Networks (VDNs)[84]. In general, VDNs show more stable training than independent DQN.

To compare both algorithms, we check the performance of each algorithm at step 10000, 20000, 30000 and 50000. Figure 2-20 (right) shows AirSim when running our scenario. The orange-colored balls are the targets. At step 10000, both algorithms showed poor performance. No drone succeeded to occupy its target. At step 20000, both algorithms showed half-succeeded result. Only one of the drones succeeded occupying its target. At step 30000, Independent DQN achieved the goal of the task; all two drones successfully occupied their targets. But VDNs still showed half-succeeded result, as it did at step 20000. At step 50000, VDNs finally achieved the goal of the task. In general, VDNs show better performance than independent DQN as independent DQN is one of the most naive version of multi-agent reinforcement learning algorithms. But in our scenario, independent DQN achieved the goal faster than VDNs. We believe this is because our scenario is too simple to fully enjoy the superiority of VDNs over independent DQN.

2.2.5 On the Asymptotic Content Routing Stretch in Network of Caches: Impact of Popularity Learning

2.2.5.1 Introduction

We aim at analytically understanding a sort of fundamental limit on how long it takes for a content requester to fetch the content when caches are connected as a network. Analyzing a network of caches is known to be a daunting task in general, since there exists a complex inter-play among the underlying (i) time-varying content popularity, (ii) content request routing, and (iii) dynamic content replacement policy. In particular, a dynamic content replacement policy is the key mechanism that plays a role of learning the popularity of contents and adaptively reconfiguring the contents in caches to the changes of active contents, where popular examples include LFU, LRU, and their variants (e.g., k-LRU and LRFU). However, analytically studying such policies even just for a *single* cache is known to be challenging[85],[86],[87], and thus a network of caches with the replacement policies is significantly challenging to analyze. To achieve our goal,

despite a large degree of theoretical challenges of a network of caches, we model a network of caches under the random dynamics of content arrivals and asymptotically study the routing stretch that refers to the number of hops until a requested content is served to its requester when the size of networked caches and the number of contents scales. The metric of routing stretch in the network of caches can be used as a good approximation of the average delay in accessing the contents, provided the network load is stable so that queueing delay at a cache is regarded as an averaged constant.

2.2.5.2 Model and Problem Statement

Network. We consider a sequence of graph $\mathcal{G}(n) = \big(\mathcal{V}(n), \mathcal{E}(n)\big)$, where $\mathcal{V}(n)$ the set of nodes or caches with $|\mathcal{V}(n)| = n$, where n is the system scale size in our asymptotic study, and $\mathcal{E}(n) \subset \mathcal{V}(n) \times \mathcal{V}(n)$ describes direct connectivity between caches. Let $d_{max}(n)$ be the maximum distance between two nodes. We let $\mathcal{C}(n)$ be the set of entire contents, and for each content $c \in \mathcal{C}(n)$, there exists its associated repository (or simply called server) that originally and permanently contains c and thus are finally accessed if a content is not fetched from an intermediate cache. Denote s_c be the content c's repository. We assume that contents are of equal size and each content $c \in \mathcal{C}$ is stored in a single server[2], say s_c, and each server $s_c \in \mathcal{S}$ is attached to a node $v_c \coloneqq v_{s_c} \in \mathcal{V}$. Let $\mathcal{S}(n)$ be the set of such original content servers. Each content server is located uniformly at random in $\mathcal{V}$. Each node $v \in V$ can cache a set of contents (thus node and cache can be interchangeably used throughout this report), having the cache size $b_v(n) \geq 0$ with the network-wide cache budget $\mathcal{B}(n)$. We assume that each cache size is equivalent across nodes, i.e., $b(n) = b_v(n) = \frac{\mathcal{B}(n)}{n}$ for all $v \in \mathcal{V}$.

Performance metric: Average routing stretch. Our primary performance metric is the response delay till a content request is fetched and served. As a useful approximation of the delay, we use the *(content) routing stretch*, defined as the number of (expected) hops until it finds the desired content, i.e., HIT occurs. Formally, let random variable X_i be the routing stretch of the i-th content request (in the entire system) and $T(n)$ be the content requests in the entire network of size n. Then, the average routing stretch Δ of the cache network is defined as follows:

$$\text{Equation 2-3:} \quad \Delta \triangleq \mathbb{E}[D], \qquad D \coloneqq \frac{1}{T(n)} \sum_{i=1}^{T(n)} X_i$$

which depends on the given system setups $\mathcal{G}, \alpha,$ and a routing policy, as well as our controlled caching policy. Our interest is on the asymptotic characterization of Δ for large n.

2.2.5.3 Centralized Popularity Learning and Content Replacement Policies

2.2.5.3.1 Oracle Policy

This policy is the one that is assumed to obtain the true popularity statistics $[p_c : c \in \mathcal{C}]$ for free, and its algorithm is described in what follows:

[2] This assumption does not restrict our results, because even when per-content multiple repositories exist, our asymptotic results hold as long as the number of repositories is $\Theta(1)$.

Oracle policy

S1. Whenever any new request for a content c arrives at a node, say v, a routing path from v to c's original server v_c is ready from a given routing algorithm. Let such a routing path be a sequence of cache nodes $P_{v,c} = (v_1, v_2, \dots, v_c)$.

S2. Then, the contents are magically placed in the sequence of nodes from v_1 to v_c, with the decreasing order of content popularity (which is given for free), where each node w in the routing path can cache the $b = B/n$ number of contents for the corresponding request.

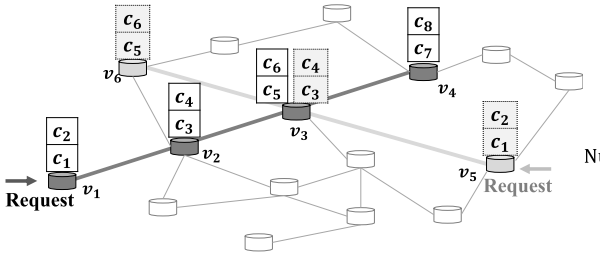


Figure 2-21: Example of content placement in Oracle: c_i is the i-th popular content in its ranking.

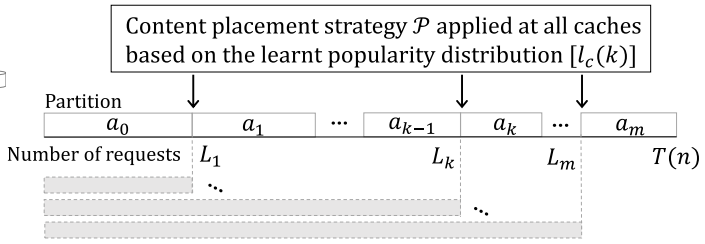


Figure 2-22: Framework of online caching policy RLP$(\boldsymbol{a}, m, \mathcal{P})$.

To illustrate, we consider the example in Figure 2-21, where suppose that $b = 2$, i.e., each node can cache 2 contents. Assume that we have a new content request generated at v_1, the content's original server is at v_4, and its priori given routing path is (v_1, v_2, v_3, v_4). Then, we cache the contents from v_1 to v_4 in the decreasing order of popularity, two contents at each node in the routing path, thus total 8 contents in the path. Note that Oracle is *unrealistic* due to the following reasons: In addition to magically-given knowledge on the true popularity statistics, there may be the case when a cache should store the contents beyond its given cache size. For example, as illustrated in Figure 2-21, for a request generated at v_5 whose routing path to the original sever is (v_5, v_3, v_6), v_3 should store the contents c_3, c_4, whereas v_3 is supposed to store the contents c_5, c_6 for the request by v_1. We allow this violation of cache size limit in Oracle, because we plan to use Oracle as a policy providing a lower bound of the routing stretch.

2.2.5.3.2 RLP Class and RLP-TC (Tilting and Cutting)

We now consider a class of polices, called RLP (Repeated Learning and Placement), where a policy in the RLP class has repeated steps for popularity learning and content placement. We claim that the RLP class is highly general so as to include any possible policies that mix popularity learning and configuring cache contents in a centralized manner.

We now elaborate on the class of RLP policies. An RLP policy, RLP$(\boldsymbol{a}, m, \mathcal{P})$, is parameterized by (i) the number of repetition steps m, (ii) m-dimensional vector $\boldsymbol{a}(m) = [a_i : i = 0,1, \dots, m]$, where $[a_i : i = 0,1, \dots, m]$ defines the $m+1$ sequential temporal partitions from the first to $T(n)$ requests, and (iii) content placement strategy $\mathcal{P}$. See Figure 2-22 for a pictorial description of an RLP$(\boldsymbol{a}, m, \mathcal{P})$. Note that $\sum_{i=0}^{m} a_i = T(n)$. Let $L_k = \sum_{i=0}^{k-1} a_i$, i.e., the aggregate number of requests until the partition a_{k-1}. Then, the partition a_k turns out to be the number of requests from when the system receives $(L_k + 1)$-th request to L_{k+1}-th request. The basic idea of the RLP$(\boldsymbol{a}, m, \mathcal{P})$ is that at each partition a_k we first learn and estimate the content popularity distribution using the L_k requests, and use the learnt popularity in the content placement strategy $\mathcal{P}$.

Examples of content placement strategy $\mathcal{P}$ include a random strategy of simply placing contents uniformly at random and a popularity-proportional strategy where the probability that a content is placed in a cache is proportional to the (learnt) popularity.

RLP$(\boldsymbol{a}, m, \mathcal{P})$ is formally described by the following recursive procedures, that specify what have to be done at the start of each partition $a_k, k = 0,1, \ldots$:

RLP$(\boldsymbol{a}, m, \mathcal{P})$

At partition a_0: Contents are placed uniformly at random at each cache of size $b = B/n$.

At partition a_k:

- *Popularity learning phase*: The system learns the popularity distribution $[l_c(k) : c \in \mathcal{C}]$ by computing the following empirical distribution:

$$\text{Equation 2-4:} \quad l_c(k) = \frac{\sum_{j=1}^{L_k} Y_c^j}{L_k},$$

  where $Y_c^j = 1$ if j^{th} request is for content c, and 0 otherwise.

- *Content placement phase*: Then, the content placement strategy $\mathcal{P}$ is applied at all caches based on the learnt popularity distribution $[l_c(k) : c \in \mathcal{C}]$, and new requests over partition a_k are served.

We now propose a policy in the RLP class, called RLP-TC (RLP with *tilted popularity distribution with cutting*. Since RLP-TC is a policy in the RLP class, its unique feature is characterized by (i) content placement strategy $\mathcal{P}$ and (ii) a construction of $\boldsymbol{a}(m)$ (see Section 2.2.5.3.2). As a content placement strategy $\mathcal{P}$, we re-manufacture the learnt empirical distribution into a "tilted distribution" and use it for content placement, which we call TC (Tilted learnt popularity with Cutting).

Regimes: Speed of popularity change. Prior to describing RLP-TC, we first describe three different regimes with respect to how fast content popularity changes: *Fast*, *Normal*, and *Slow*, which, in practice, may differ depending on the content categories[88]. This classification is used to describe the caching policies, present our analytical results and their interpretations. We define three regimes of $T(n)$ as follows:

$$\text{Equation 2-5:} \quad \begin{cases} \text{FAST} & if\ (\log d_{\text{A}}) \leq T(n) \leq \Theta(d_{\text{A}} \log d_{\text{A}}), \\ \text{NORMAL} & if\ \Theta(d_{\text{A}} \log d_A) < T(n) < \Theta(d_{\text{A}}^2 \log d_{\text{A}} \cdot M^2), \\ \text{SLOW} & if\ \Theta(d_{\text{A}}^2 \log d_{\text{A}} \cdot M^2) \leq T(n), \end{cases}$$

where recall d_A is the average to-server-distance, and $M = M(\alpha)$ is such that:

$$\text{Equation 2-6:} \quad \frac{1}{M} = \sum_{i=1}^{d_A} \frac{1}{i^{\alpha/2}} = \begin{cases} \Theta(1) & if\ \alpha > 2, \\ \Theta(\log d_{\text{A}}) & if\ \alpha = 2, \\ \Theta(d_{\text{A}}^{1-\alpha/2}) & if\ 2 > \alpha > 1. \end{cases}$$

Note that our classification differs in NORMAL and SLOW regimes depending on M, which also relies on

the popularity bias parameter α.

Policy description. We first describe and explain RLP-TC policy, followed by its rationale.

RLP-TC=RLP($\boldsymbol{a}, m$,TC) policy

INPUT: $T(n)$ and α.

- **Construction of $\boldsymbol{m}$ and $\boldsymbol{a}$:** We first choose a_0 with its dependence on $T(n)$ as follows:

  Equation 2-7: $$a_0 = \begin{cases} T(n) & \text{if FAST,} \\ \Theta(\log d_{\mathrm{A}}) & \text{if NORMAL and SLOW,} \end{cases}$$

  Then, for $\Theta(d_{\mathrm{A}}\log d_{\mathrm{A}}) < T(n)$, choose $a_1 = \max\{o(a_0), \Theta(1)\}$, and select m:

  Equation 2-8: $$m = \log_{\mathrm{r}}\left(1 - \frac{(1-r)(T(n)-a_0)}{a_1} \right),$$

  where

  Equation 2-9: $$r = 1 - \frac{a_1}{T(n)} \quad (< 1).$$

  Then, the remaining sequence $(a_2, a_3, \dots, a_m)$ is constructed by the geometric series, starting from a_1, with the common ratio r, i.e., $a_k = a_1 r^{k-1}, k = 1, \dots, m$.

- **TC strategy:** In the general RLP policy description, at each step a_k, we apply the following procedures to the *content placement phase*:

  S1. *Construction of tilted popularity distribution*. Using the empirical distribution $[l_c(k) : c \in \mathcal{C}]$, we compute the following tilted distribution $[\hat{l}_c(k) : c \in \mathcal{C}]$:

  Equation 2-10 $$\hat{l}_c = \begin{cases} \frac{\sqrt{l_c}}{\sum_{c=1}^{\hat{\imath}} \sqrt{l_c}} & \text{if } rank(c) \le \hat{\imath}, \\ 0 & \text{otherwise,} \end{cases}$$

  where $\hat{\imath} = b \cdot d_{\mathrm{A}}$ and recall that $rank(c)$ is the popularity ranking of content c.

  S2. At each cache, $b(= B/n)$ contents are randomly selected according to the distribution $[\hat{l}_c(k) : c \in \mathcal{C}]$ without duplication of contents.

Policy explanation. We present the rationale of RLP-TC. First, the constructed $\boldsymbol{a}$ and m depend on a given $T(n)$ and the popularity bias α, where a_0's dependence is of significant importance. As observed in (Equation 2-7), the length of a_0 decreases in NORMAL and SLOW rather than FAST, because for larger $T(n)$, more chances to learn the popularity are allowed, whereas for smaller $T(n)$, the initial learning becomes more crucial. In NORMAL and SLOW, the remaining steps $(a_1, a_2, \dots, a_m)$ as well as the total number of iterations m are chosen as a geometric series such that their sum equals to $T(n)$, as seen in (Equation 2-8) and (Equation 2-9). Second, as a content placement strategy, the proposed TC strategy first

re-manufactures the learnt popularity distribution into a *tilted form with cutting,* as in (Equation 2-10) at each step k. It means that we nullify the distribution of unpopular contents, where we maintain the popularity only up to the ranking index $\hat{\imath} = b \cdot d_{\mathrm{A}}$, and renormalize the distribution (more specifically taking the square root, i.e., $\sqrt{l_c}$, as seen in (Equation 2-10)). Then, we randomly place the contents according to the computed tilted distribution **(S2)**. To intuitively understand, take an example of three contents, where the originally learnt distribution in step k is $(l_c(k)) = (0.7,0.2,0.1)$ with $\hat{\imath} = 2$. Then, in our tilted distribution with cutting, we have $\left(\hat{l}_c(k)\right) = (0.65,0.35,0)$. We will explain how this helps in achieving short routing stretch next.

Rationale of tilting with cutting. A natural way is to directly use the empirically learnt popularity distribution with which contents are randomly placed at each cache. However, what we do is to use a tilted distribution $[\hat{l}_c(k)]$, such that $[\hat{l}_c(k) \propto \sqrt{l_c(k)} : c \in \mathcal{C}$. The key effects of tilting with cutting are summarized in what follows: (i) *tilting:* making the popularity distribution less biased and (ii) *cutting:* making unpopular contents uncached. If we consider only a single, stand-alone cache, just the empirical distribution-based placement might be enough. However, in the network of caches, the performance of our interest, which is a routing stretch, is a non-trivial complex function of coupled behaviors among the caches in a given routing path. This non-trivial relationship requires us to re-consider the obtained empirical popularity distribution and also effectively use the available rooms for caching whose size is strictly smaller than the number of contents. This motivation leads us to tilt the empirical distribution with unpopular contents excluded from caches.

Why square root in tilting? The remaining question in tilting, is why the choice of $\sqrt{l_c(k)}$ is made for the obtained empirical distribution $\sqrt{l_c(k)}$? Just for simplicity of exposition, assume $b = 1$, i.e., each node can cache only one content, and also the to- server-distance d_{A} is large. We now consider a cache placement strategy under which content c_i (whose popularity distribution is p_i) is cached in each cache with probability q_i. For large $T(n)$, the expected routing stretch Δ for d_{A} roughly becomes:

$$\text{Equation 2-11:}\quad \Delta = \sum_{i=1}^{|\mathcal{C}|} p_i \cdot \frac{1}{q_i} = \left(\sum_{i=1}^{|\mathcal{C}|} p_i \frac{1}{q_i}\right)\left(\sum_{i=1}^{|\mathcal{C}|} q_i\right) \geq \left(\sum_{i=1}^{|\mathcal{C}|} p_i^{\frac{1}{2}}\right)^2,$$

where $1/q_i$ is the average stretch from a requester to the cached node of content c_i, and the last inequality comes from the Cauchy-Schwarz inequality. In Cauchy-Schwarz inequality, it is widely known that the equality holds if and only if there is some constant k such that $p_i \frac{1}{q_i} = k \cdot q_i$ for all i. Therefore, Δ is minimized when $q_i \propto i^{-\frac{\alpha}{2}}$, and the minimum value is $\left(\sum_{i=1}^{|\mathcal{C}|} p_i^{\frac{1}{2}}\right)^2$. This is why $\sqrt{l_i}$ is selected for TC where l_i goes to p_i for sufficiently large $T(n)$. Note that a special case when $q_i = p_i$ corresponds to the case utilizing content popularity distribution of the cache placement, and $\Delta = |\mathcal{C}|$.

2.2.5.4 Analysis: Routing Stretch

We now present our main results on the average routing stretch Δ, as addressed in (1), for Oracle and RLP-TC. To make our analysis of Δ tractable, we first rewrite Δ as the average over a random to-server-distance d_{tsd} from a content requester to the corresponding content server (where the randomness comes from the location of content requesters and the corresponding content servers) as follows:

$$\text{Equation 2-12:}\qquad \Delta = \mathbb{E}[\Delta(d_{\mathrm{tsd}})] = \sum_d f_{\mathrm{tsd}}(d)\Delta(d)$$

where, to abuse the notation, $\Delta(d)$ is the "expected" routing stretch when to-server-distance is d, where the

expectation is taken with respect to the randomness in the contents and content caching policy, and $f_{\mathrm{tsd}}(d)$ is the distribution of d. Note that having a closed form $f_{\mathrm{tsd}}(d)$ is challenging and thus makes the routing-stretch analysis hard, because $f_{\mathrm{tsd}}(d)$ depends on the given topology $\mathcal{G}$ and the underlying routing algorithm. For example, even for a Erdös Rényi (ER) random graph, which is one of the simplest random graphs, when the shortest path routing algorithm is used, $f_{\mathrm{tsd}}(d)$ is still unknown. Thus, to purely focus on our interest, we use Jensen's inequality and obtain:

Equation 2-13: $\Delta = \mathbb{E}[\Delta(d_{\mathrm{tsd}})] \leq \Delta(\mathbb{E}[d_{\mathrm{tsd}}]) = \Delta(d_{\mathrm{A}})$,

where recall that d_{A} is the average to-server-distance, and we now consider $\Delta(d_{\mathrm{A}})$ as our major metric to analyze. To differentiate from $\Delta(d)$ for any given d, we often call $\Delta(d_{\mathrm{A}})$ *average stretch upper-bound*, or simply *stretch upper-bound*.

2.2.5.4.1 Oracle

This is formally presented in Theorem 1 which states that it is optimal in the sense that Oracle has shorter routing stretch than any other policy in the RLP class, and in Theorem 2 which presents the asymptotic average routing stretch of Oracle.

Theorem 1. *Let $D^{\mathcal{O}}$ and $D^{\mathcal{A}}$ be the random routing stretches (as defined in* (1)*) of* Oracle *and an arbitrary policy $\mathcal{A}$ in the RLP class, respectively. Then, the following stochastic dominance of* Oracle *holds: $D^{\mathcal{O}} \leq_{st} D^{\mathcal{A}}$, which means $\mathbb{P}[D^{\mathcal{O}} > x] \leq \mathbb{P}[D^{\mathcal{A}} > x]$* for any $x \geq 0$.

Theorem 2. *For a given to-server-distance d between a pair of a content requester and its server, the average routing stretch $\Delta(d)$ and the stretch upper-bound $\Delta(d_{\mathrm{A}})$ of* Oracle *scale are as those in Table 2-2.*

Thanks to Theorem 1, the result of asymptotic routing stretches in *Table 2-2* provides lower bounds of $\Delta(d)$ of any policy in the RLP class. As expected, as α decreases, we lose the power of caching, thus leading to the increase of routing stretch. When $\alpha > 2$, the stretch is order-wise optimal (i.e., a constant order), and only up to $\alpha = 2$, the stretch is sub-polynomial. Note that when $0 < \alpha \leq 1$, the caching gain vanishes, so requiring to reach the corresponding original server. We now seek to find a policy in the RLP class whose performance is close to that of Oracle, if any.

2.2.5.4.2 RLP-TC

We now present the result on the stretch bound $\Delta(d_{\mathrm{A}})$ of RLP-TC in Theorem 3. We will focus only on the case when $\alpha > 1$, because even the lower-bound provided by Oracle proves that there is no caching gain for $\alpha \leq 1$, see Theorem 1.

Theorem 3. *For $\alpha > 1$, the upper-bound of routing stretch $\Delta(d_{\mathrm{A}})$ for* RLP-TC *scales as follows: With high probability,*

Equation 2-14: $$\Delta(d_A) = \begin{cases} \Theta(d_A) & if \quad \text{FAST}, \\ \Theta\left(d_A^2 \frac{\log d_A}{T(n)}\right) & if \quad \text{NORMAL}, \\ \Theta\left(\frac{1}{M^2}\right) & if \quad \text{SLOW}. \end{cases}$$

Note that the performance of RLP-TC depends on $T(n)$, as expected. In FAST, $\Delta(d_{\mathrm{A}}) = \Theta(d_{\mathrm{A}})$, i.e., there is no gain due to the lack of time to learn and apply such a learning result to the content placement. In

NORMAL, $\Delta(d_A)$ decreases as $T(n)$ increases, because the repeated learning process helps where we have enough time to learn the popularity and use such knowledge in placing contents, until $\Delta(d_A)$ reaches $\Theta(1/M^2)$ in (10) at the threshold $T(n) = \Theta(d_A^2 \log d_A \cdot M^2)$ in (2). After this threshold, i.e., SLOW, the repeated learning and placement do not help in reducing routing stretch, so as to keep the stretch $\Theta(1/M^2)$. Since M is inversely proportional to α as in (3), we can conclude that a system efficiently utilizes the chance of learning with relatively smaller $T(n)$ for the cases of having highly biased content popularity. In SLOW, RLP-TC is ***near-optimal***, since it achieves $\Delta(d_A)$ as Oracle in *Table 2-2* for $\alpha > 1$ except the case $\alpha = 2$, we have an order-wise difference $\Theta(\log d_A) = O(\log n) = o(n)$ between $\Theta(\log^2 d_A)$ in RLP-TC and $\Theta(\log d_A)$ in Oracle.

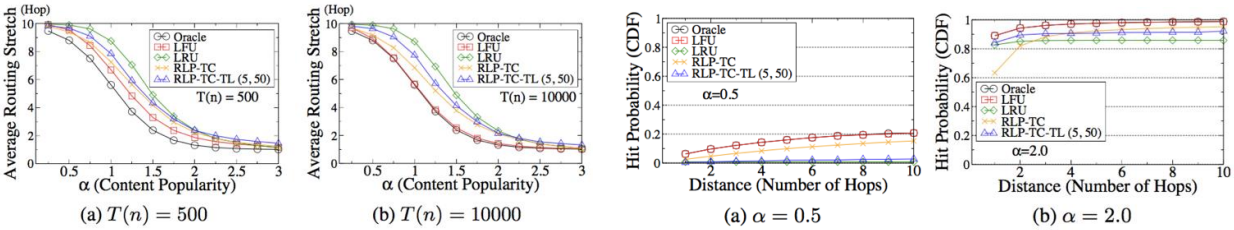
Table 2-2: Routing stretch: Oracle

Popularity	$\Delta(d)$	$\Delta(d_A)$
$2 < \alpha$	$\Theta(1)$	$\Theta(1)$
$\alpha = 2$	$\Theta(\log d)$	$\Theta(\log d_A)$
$1 < \alpha < 2$	$\Theta(d^{2-\alpha})$	$\Theta(d_A^{2-\alpha})$
$0 < \alpha \leq 1$	$\Theta(d)$	$\Theta(d_A)$

Table 2-1: Stretch UBs of RLP-TC and RLP-TC-OL (RLP-TC with One-time Learning)

Regime		RLP-TC-OL	RLP-TC
FAST		$\Theta(d_A)$	$\Theta(d_A)$
NORMAL		$\Theta\left(\sqrt{d_A^3 \frac{\log d_A}{T(n)}}\right)$	$\Theta\left(d_A^2 \frac{\log d_A}{T(n)}\right)$
SLOW	$T(n) < \Theta(d_A^3 \log d_A)$	$\Theta\left(\sqrt{d_A^3 \frac{\log d_A}{T(n)}}\right)$	$\Theta(\frac{1}{M^2})$
	$T(n) \geq \Theta(d_A^3 \log d_A)$	$\Theta(\frac{1}{M^2})$	$\Theta(\frac{1}{M^2})$

2.2.5.5 Simulation Results



(a) $T(n) = 500$ (b) $T(n) = 10000$ (a) $\alpha = 0.5$ (b) $\alpha = 2.0$

Figure 2-23: Line topology. Average routing of five policies (Oracle, LFU, LRU, RLP-TC, and RLP-TC-TL (5,50)) for various α at $T(n) = 500$ and 10000.

Figure 2-24: Line topology. Hit probability (CDF) of five policies (Oracle, LFU, LRU, RLP-TC, and RLP-TC-TL (5,50)) for high ranked 50 contents over $\alpha = 0.5$, and 2.0.

Line topology. We compare the routing stretches of RLP-TC, and RLP-TC-TL (5, 50) with LRU and LFU as shown in Figure 2-23 for $T(n) = 500$, and 10000. Oracle has the lowest stretch. RLP-TC, and RLP-TC-TL (5, 50) performs between LFU and LRU, where we see that (i) a small number of popularity learning at the starting period of the system is highly beneficial (at least under our setting), and (ii) our analysis of centralized RLP-TC algorithms can be a good approximation of the one between LFU and LRU. Note that it is natural that LFU outperforms LRU, since LRU has more strength when the content configurations are dynamically changing. Figure 2-24 and Figure 2-25 show on average when the requested contents experience HIT. Oracle and LFU show the similar cumulative distribution function (CDF) of hit probability for 50 top-ranked contents, since LFU operates so as to autonomously place the contents as an ascending order of content popularity based on the history of requested contents at each node. Although the hit

probability of RLP-TC for the first node is lower than that of LRU because RLP-TC probabilistically selects the contents based on the estimated popularity, at 10-th node, the sum hit probability (CDF) of RLP-TC exceeds that of LRU, where RLP-TC smartly utilizes the tilted distribution with cutting. Thus, in LRU, there is no increment of the sum hit probabilities (CDF) between 10 and 50 top-ranked contents due to caching unpopular contents (over the rank 50). Despite less active learning in RLP-TC-TL (5, 50), it is able to detect the relatively highly popular contents, achieving the high hit probability at the first node, but due to the lack of information, RLP-TC-TL (5, 50) does not cache contents within ranked 10-th to 50-th contents in the backside caches in a line, so as to have the higher routing stretch than RLP-TC.

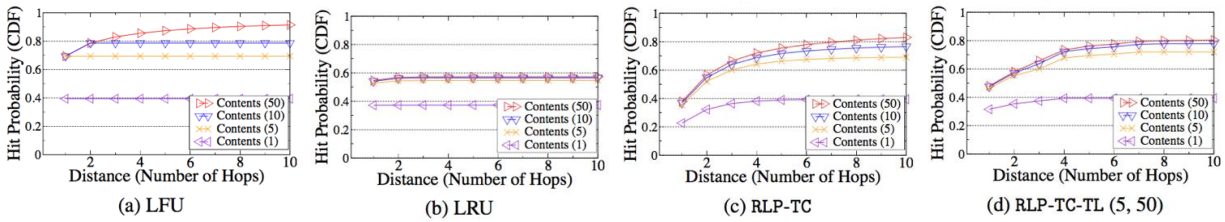


Figure 2-25: Line topology. Hit probability (CDF) of four policies (LFU, LRU, RLP-TC, and RLP-TC-TL (5,50)) for high ranked $1, 5, 10,$ and 50 contents for $\alpha = 1.5$.

Tree and AS topologies. First in tree topology, we compare the performance of RLP-TC ($a_i = 1$ for $0 \leq i \leq T(n)$) with LFU and LRU when content popularity follows Zipf- like distribution with $|\mathcal{C}| = 1000$ and $b = 5$. As shown in Figure 2-26, average stretches of LFU and LRU are similar to that of RLP-TC when $T(n) = 10000$, and RLP-TC outperforms LFU and LRU for $T(n) = 500$ because RLP-TC utilizes the gathered information and replaces cached contents based on information for each request. In AS topologies, we perform the simulation during 100000 slots where we focus on SLOW regime assuming that popularity distribution is a priori given to RLP-TC. For each test, we first place content servers uniformly at random, and a content request arrives at a cache with probability 0.5 at the beginning of the time slot, and unresolved requests are forwarded to the next cache under the shortest path routing. Figure 2-27 shows the results of various caching policies over three topologies. We compare the performance of RLP-TC with dynamic caching strategies LFU and LRU. Figure 2-27 shows the absolute (average) routing stretch performances of two dynamic cache replacement policies, LRU and LFU, compared to the RLP-TC, for three graph topologies. As done in the analysis earlier, the average routing stretch in y-axis corresponds to the number of hops. Table 2-3 shows the normalized average routing stretches of LFU and LRU by that of RLP-TC. We observe that RLP-TC's routing stretch has good match in those of LFU and LRU (on average, about 5.2% and 9.6% differences for LFU and LRU, respectively). Note that LFU is known to perform better than LRU with higher implementation complexity. From our simulation results, our analysis considering static cache policies can be used to predict the large-scale cache networks' average routing stretch performance.

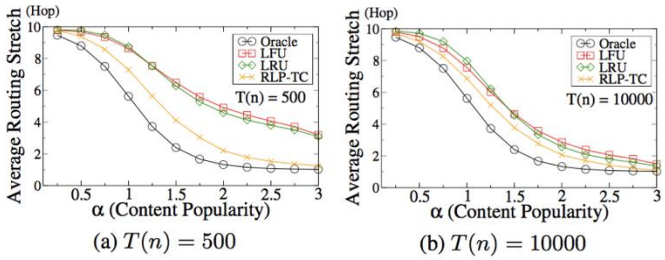


Figure 2-26: Tree topology. Average routing stretches of four policies (Oracle, LFU, LRU, and RLP-TC) for various α at $T(n) = 500$ and 10000.

Topology α	0.5	1	1.5	2
Cogent (LFU)	1.017	0.967	0.874	0.836
Cogent (LRU)	1.048	1.170	1.167	1.063
Colt (LFU)	1.011	0.979	0.891	0.839
Colt (LRU)	1.041	1.158	1.152	1.026
TW (LFU)	1.017	1.016	0.945	0.937
TW (LRU)	1.030	1.135	1.097	1.066

Table 2-3: AS topology. Average routing stretches of LFU and LRU normalized by that of RLP-TC.

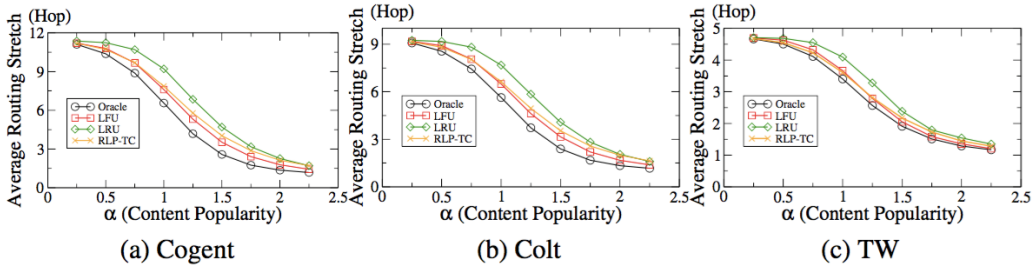


Figure 2-27: AS topology. Average routing stretch performance.

2.2.5.6 Conclusion

We presented asymptotic analysis of the routing stretch of large-scale cache networks. We focused on quantitatively understanding the relation between content popularity and average to-server-distance, as well as the impact of learning and cache sizing heterogeneity. We studied the asymptotic routing stretch of cache networks under on-line repeated learning and content placement policy. We also derived that this scaling routing stretch is made under different dependence of the speed of popularity change, average to-server-distance in the network of caches, and the shape of the popularity distribution.

3 AI-assisted Edge Computing

3.1 Overview

The recent years has seen the rise of Internet of Things (IoT). From our mobile devices to home sensors, billions of things are now connected to the cloud to which they transmit large amount of data of different volume, variety, and velocity. This increase in data introduces new challenges to the classical cloud-based computing paradigms which motivates edge computing research. Several state-of-the-art AI-assisted edge computing works, particularly in Unmanned Aerial Vehicles (UAVs), are presented in this section.

3.1.1 Literature Review

In the Primo5G project, Mobile Edge Computing (MEC) is one of the key enablers for improving Quality-of-Service. In a glance, MEC refers to an IT service and computing environment at the edge of the mobile network, within the radio access network (RAN) and in close proximity to mobile subscribers [89] via a MEC-enabled infrastructure, in fact, users can directly connect to the nearest cloud service-enabled edge network and benefit of low-latency services, offloading for fast computational applications, caching, and so on. Also, MEC technology can be utilised for agile and optimised service deployment by leveraging location information about the subscribers into decision making algorithms.

3.1.1.1 Location-aware MEC

The 5GC, as standardized by 3GPP in Release-15, contains elements on the user plane and on the control plane that can allow a lean MEC-enabled infrastructure. For instance, by exploiting the clean control plane-user plane separation, it is possible to deploy User Plane Functions (UPFs) in a distributed fashion, co-located with the gNBs on moving nodes, so as to minimize the user plane latency. Co-located with the UPFs, we could also envision local clouds where computational resources exist and application servers can be hosted. Thus, by leveraging these flexibilities into the design of MEC algorithm/network deployment, it is expected that challenges concerning offloading techniques, low-latency, storage, energy efficiency and mobility management can be solved.

Of a particular interest is the adoption of location-awareness along with artificial intelligence [90] [91] [92] [93] in MEC. For instance, in [94] a location-aware load prediction approach is developed to forecast the load at each Edge Data Center (EDC). The key is to utilizes both historical load information of the current Edge data center as well as of those in the near proximity. In [95], location awareness is exploited to devise location-customized caching schemes to maximize the total content hit rate. In [96] [97] [98], other examples related to caching, privacy and services are considered.

With a focus on moving networks and, more specifically, on the Primo5G use case of forest fire fighting, a location-aware MEC is a promising solution to improve the robustness of the system to frequent disconnection of link between devices and edge network. When the device is in moving state, in fact, service quality of the application is degraded due to varying properties of network parameters such as delay, bandwidth, jitter etc. without any interruption. In [99] and, more in particular in [100] [101], a similar problem is addressed and resolved with the aid of machine learning and position information. More specifically, a deep q-learning method is utilized for maintaining a reliable connection between a pilot robot and a set of followers.

In addition to preserving connectivity, a location-aware MEC can also provide support for service migration (relocation of the application instance). The work in [102], for instance, tackles the problem of resource migration when a mobile node moves from one to another location area. The proposed method attempts to maximize the QoS in terms of latency by considering the current conditions in the network. In other words

the, the migration algorithm attempts to locate the mobile edge platform that is related to the new cell and check if there are resources available to support the migration process.

3.1.1.2 Super Resolution

Recently, deep-learning-based methods have achieved great success in super-resolution tasks. Despite the common opinion that deeper neural networks perform better, low-level image processing domains such as super resolution have been known to suffer stability issues when training them. Dong et al. [103] tackled this issue finding that a sequence of convolutional neural networks show great potential for the Super Resolution (SR) task. Lee et al. [104] replaced the 9-1-5 architecture of Super Resolution Convolutional Neural Network (SRCNN) with twenty layers of 3x3 convolution filter and applied global skip connection, based on inspiration from Resnet [105]. As a result, the method outperformed existing approaches by a considerable margin, confirmed the possibility of applying deep neural networks to SR tasks, and greatly inspired subsequent studies [106] [107] [108] [109]. Although differences exist in their approaches, the aforementioned studies have all contributed significantly to increasing either the depth of neural networks or the stability of training in deeper networks. As the capability of a neural network is proportional to the depth of the network, a trade-off typically exists between a network's depth and its performance and computational cost [110]. Deeper networks perform better, but require more execution time and computational resources. Therefore, selecting a suitable model with proper consideration of these trade-offs has been an important factor in achieving system stability and efficiency. These trade-offs are equally applied to SR tasks, as summarized in [111].

3.1.1.3 Auction-based Algorithm

There have been several research results to solve limited battery and limited resource scheduling problems through auctions [112] [113] [114]. Park et al. [112] proposed a method to optimize battery assignment and drone scheduling, assuming that the battery can be quickly replaced. The joint assignment and scheduling problem is formulated as a two-stage problem, where the assignment problem is solved by a heuristic and the scheduling problem is formulated as an integer-linear programming (ILP) problem. This paper proposes the scheduling algorithm based on auction. The proposed method uses information provided by drones capable of communicating with mobile charging stations to overcome the inability of a central service provider to acquire perfect state information in a distributed drone network. However, a solution based on battery assignment necessarily maps to a stationary service station. This imposes some limitations [113], which are mitigated when using mobile charging stations.

Couture-Beil et al. [113] considered a system of mobile robots executing a transportation task supported by a charging station. The location of the charging station is a major factor in determining the operations and performance of the robots, and the paper assumes that the mobile charging station is itself an autonomous robot that attempts to incrementally improve its location. Although this work considers a mobile charging station, the problem of charging scheduling is not considered. In a more general scenario, the resources of the charging station are limited and the number of robots to be charged may be larger than the actual charging capacity of the station. Therefore, the charging system will need to implement forms of prioritization to optimize the charging process. The method proposed herein incorporates a notion of priority using an auction formulation based on the valuation of the drones.

Wellman et al. [114] proposed an auction mechanism to solve a resource allocation problem in a distributed computing system. The inherently distributed nature of the system makes the resolution of the problem much harder. The paper proposes an auction-based solution to address such challenge. The proposed mechanism is configured as a two auction mechanism, used to compute optimal solutions at the single unit within the distributed scheduling problem in a computationally efficient manner. However, Wellman et al. [114] assume prior knowledge of the environment where the auction mechanism is executed, which may limit its application in real-world distributed scenarios. The method we propose herein uses an auction-

based solution to solve the resource allocation problem, and employs deep learning to extract the required features automatically from the environment, so that prior knowledge is not necessary.

3.1.1.4 AI-based Transmission

Fog computing shares the same principle of moving computing resources to the edge with mobile edge computing [115]. Different architectures of vehicular fog computing, which is used to address the challenges in immersive video service, have been proposed in the literature. For example, Satyanarayanan et al. [116] proposed to turn each vehicle into one fog node and select a coordinator for each zone. Xiao et al. [117] preferred to turn commercial fleets into fog nodes so as to serve neighbouring vehicles and passengers, while Hou et al. [118] recommended utilizing additional computing power on slow moving or parked vehicles. In addition, Ni et al. [119] studied the architecture of fog-based vehicular crowd sensing considering security, fairness, and privacy.

Relevant to multi-client task allocation, previous research works have investigated task allocation in fog/edge computing [120] [121] [122]. Li et al. [123] minimized service response time and energy consumption by jointly optimizing the offloading strategy and the Quality-of-Result (QoR) for all edge nodes. Sardellitti et al. [124] jointly optimized radio and computational resources of multiple cells in edge computing. Liu et al. [125] studied the multitask allocation problem for the edge environment with consideration of resource-intensive and latency-sensitive mobile applications. Dinh et al. [126] presented an offloading framework to jointly minimize the execution latency of tasks and power consumption of devices considering CPU frequency. Deng et al. [127] studied the trade-off between energy consumption and transmission delay in a cloud computing system. However, these existing results cannot be directly applied to multi-drones firefighting scenario, as they did not consider the mobility of firefighting trucks. Feng et al. [128] proposed a job caching framework in VFC based on ant colony optimization algorithm. However, it focused on caching and did not consider other scenarios e.g., local processing. Chameleon is the joint optimization to provide service latency and quality in VFC, with consideration of the mobility of fog nodes.

3.1.2 Challenges

3.1.2.1 Control of Trade-off between Computing and Communication loads

The scenarios that are considered in this project include the two-tier hierarchical edge computing system consisting of the MEC server, drone commanders, and drone members. After collecting the real-time images, the drone members deliver the images to their drone commander and/or the MEC server directly. At the MEC server, processing tasks, e.g., super-resolution imaging, are required to achieve certain level of final image qualities to analyze the situation around the disaster scene and to give instructions to drone members and firefighters. When all members in the hierarchy are assumed to have their own power engines, there exists a tradeoff between computation and communication loads in this scenario.

To provide real-time visual information, drone members should transmit the extreme quantity of sequential images and communication loads are expected very huge. Moreover, the channel condition near the disaster scene is expected to be very harsh and resources should be distributed to a lot of UAVs, therefore drone members would prefer to transmit the compressed version of collected images. However, the more compressed the images received by the drone commander are, the more the computational work is required to improve the image quality. Depending on channel capacities, computation capabilities of all drones and the MEC server and the hierarchy structure, decisions on how much compress the collected images, which tier process the compressed images, and how to organize the hierarchy and to group drone members under monitoring of drone commanders should be appropriately made.

3.1.2.2 Battery Charge Problem of UAVs

State-of-the-art drone technologies have severe flight time limitations due to weight constraints, which inevitably lead to a relatively small amount of available energy. Therefore, frequent battery replacement or recharging is necessary in applications such as delivery, exploration, or support to the wireless infrastructure. Mobile charging stations (i.e., mobile stations with charging equipment) for outdoor ad-hoc battery charging is one of the feasible solutions to address this issue. However, the ability of these platforms to charge the drones is limited in terms of the number and charging time.

There are several ways to powering the drones which have been proposed in the literature. We can divide them into two main classes: (i) harvesting energy directly from the surrounding environment, and (ii) taking energy from an electrical source such as a charging station [113]. Within the latter class of approaches, the charging stations can be either stationary or mobile. However, solutions based on stationary charging stations may constrain the geographical area of operations around specific locations. In order to deal with this issue, mobile charging stations can be used although they face other challenges [113]. As they are mobile, the size of these charging stations needs to be comparably smaller to that of fixed stations. As a consequence, the capacity of the system has limitations, leading to relatively low charging speeds and a relatively smaller number of drones that can be charged simultaneously [113].

3.1.2.3 UAV Positioning

Since the channel conditions are very harsh around the disaster scene because there are so many blockages, fire, debris, etc., UAVs need to find their line-of-sights to collect the necessary information as well as to communicate with firefighters at the disaster scene. Furthermore, the command center does not know the exact situation around the scene, therefore UAVs should find their positions by themselves using learning-based methods. Along with the challenge in 3.1.2.1, UAV positioning is very critical to control the tradeoff between computation and communication loads, to charge UAVs, and to allocate resources to two-tier hierarchical edge computing system.

3.1.2.4 AI-based Transmission

To provide edge computing based immersive video service with high performance and reliability, there are still several technical challenges to be overcome. We discuss these challenges and potential solutions below.

3.1.2.4.1 Interference in drones-to-fog Communication Networks

Due to the interference in D2D communication networks, the packet loss rate of the Dedicated Short Range Communications (DSRC) connections increases dramatically with the number of connected drones. The fog nodes running the Wireless Access in Vehicular Environment (WAVE) stack are supposed to be equipped with a single radio. However, in theory, they can communicate on all seven channels in the DSRC band. To eliminate channel competition, one solution is that each vehicular fog node (firefighting truck) dynamically assigns orthogonal channels to drones. Since the number of available channels is limited, these channels will be spatially reused among drones. If some fog nodes are closely located, the cellular fog node (zone head) will help assign channels in a coordinated manner (through dynamic point selection, coordinated beamforming, or scheduling) to prevent severe inter-fog interference.

3.1.2.4.2 Service Interruption

Firefighting trucks and drones move at varying speed. As a result, the distance between fog nodes and drones changes all the time. When drones leave the communication range of a connected vehicular fog node, the ongoing fog computing services such as video streaming will be interrupted. As the driving routes of vehicular fog nodes are controllable, it is possible to predict the service availability and to plan the

migration of the service to other fog nodes. For example, videos can be forwarded to zone heads when the drones are flying away from a vehicular fog node.

3.1.2.4.3 Resource Management

In the practice of multi-drone firefighting deployment, it is important to coordinate and manage all the vehicular fog nodes in an efficient manner. The resource management strategies must handle two issues. First, one drone may be located within the communication ranges of several vehicular fog nodes. In this case, it is a question of how to select the most suitable vehicular fog node for the drone. Second, vehicular fog nodes are deployed in a distributed manner. They may not be able to communicate directly with each other due to long distance. It would be more feasible to deploy resource management modules on cellular fog nodes, and utilize the wired connections between cellular fog nodes and the cellular connectivity of vehicular fog nodes to schedule the service deployment.

3.2 PriMO-5G Innovation

This section will present innovative contributions from PriMO-5G partners. These are current state-of-the-art research in AI-assisted edge computing.

3.2.1 Resource Optimization for Network Slicing and MEC using AI

Generally, IP based networks operate on a best effort, which means that all traffic has equal priority and equal probabilities of being delivered. Similarly, when a network becomes congested, all traffic has an equal probability of being delayed or dropped in the worst case. Quality of Service (QoS) selects network traffic, prioritizes the traffic according to its relative importance, and uses congestion avoidance to provide priority indexed treatment. QoS can also limit the bandwidth used by a network as well as make network performance more predictable and bandwidth utilization more effective. The QoS can be enforced in the packets at link layer 2 or network layer 3. The port/node level QoS classification at link layer 2 is based on VLAN priority (PRI) field. This means that all tagged traffic is classified based on a VLAN PRI and untagged traffic classified as best effort (BE). The QoS classification at layer 3 can be done using Type of Service (ToS) if IP transport is used or packet tagging if MPLS is used.

Following the introduction of 5G technology along with the expected increase in connected devices, network operators are facing the biggest challenge to maintain the quality of service and performance to meet customer demands using traditional ways.

SDN is proposed as a solution to solve the challenges faced by traditional networks by separating the control and data plane of the network device to ensure QoS of the network traffic in an efficient way. SDN provides network programmability, flexibility, and agility of networks. SDN is expected to introduce and enforce dynamic policies to ensure QoS as it provide network programmability and flexibility.

SDN plays an important role in traffic engineering and it can be used for dynamic load balancing. Dynamic load balancing (DLB) reroutes traffic using statistics collected from each device in the network. DLB is a mechanism that takes statistical parameters from each network device and evaluates the network traffic to modify the flow accordingly. Optimizing the usage of network resources using DLB is far better than static round robin routing algorithm. Round Robin Algorithm (RRA) is the default load balancing strategy which treats all available paths equally. The drawbacks in large networks are the performance cost and overhead for collecting statistics from each device in the network and computing the best route. Moreover, DLB needs high computing power to calculate the best path. Therefore, there is a need to find some other mechanism to measure and optimize the usage of resources given the complexity of the network and traffic growth. Thus, studying the trends of traffic to identify which time of the day traffic load is quite high in some areas and low in other areas is important to design optimal TE logic. For example, in areas where there is a high number of office buildings, the traffic load is high during day time and week days, but it is quite low on

weekends. These kinds of trends are similar every working day around those buildings or shopping malls. Machine Learning (ML) is used to develop a control logic on the network behavior and train the system so that traffic can be rerouted with no need to calculate and compute new optimal paths.

We use supervised learning where there is new training data that can predict the typical outcome in a similar situation. Rather than writing algorithms to do the load balancing of the network, the supervised learning takes several samples of labeled data and trains the devices. Therefore, if similar data is encountered, the device can recognize it and take actions based on the training. In supervised learning the device is trained for some known output. The idea behind supervised learning is that for some inputs, we want to have a certain value as an output. The machine learning algorithms run based on the inputs until output values close enough to the target value are obtained.

We consider the usage of SDN based network slicing as part of a mobile backhaul orchestrator for managing the mobile backhaul network using machine learning algorithms. The end to end solution requires network slicing in the radio access network as well as in the mobile backhaul. However, due to the lack of radio network slicing, we will focus in mobile backhaul slicing and evaluate the benefits of machine learning. The proposed mobile backhaul orchestrator (MBO) that will create and manage network slices based on SDN and machine learning logic will guarantee QoS on each slice.

The MBO will include a network monitoring to provide the learning data for the ML. This module will receive traffic and will collect statistical network information from each device (switch) in the network every 30 minutes. We divide the collected information in two parts, the port statistics and flow statistics. Flow statistics indicates the type of packets, packet size, duration, and number of packets that match the specific flow. Those flow statistics are gathered as a reply to the flow stat request message sent to the switches. On the other hand, port statistics of a specific switch can be collected by sending port stat request message to the device under investigation. The reply message includes the number of transmitted and received packets, total size of received and transmitted bytes, time stamp, number of error packets received, port number of the specific switch and its id number, and packet drops of both received and transmitted packets.

The network monitoring module is interpolating a specific information such as byte count and time stamps, and transfers them to the route optimizer for further processing to identify and optimize the usage of resources. Machine Learning Engine, which is part of the MBO, collects information from the network monitoring and calculates optimal routes for each slice. The ML engine is the key component of the MBO which is designed in this work and includes machine learning technology to provide optimal usage of resources.

After integrating all the modules into the MBO, we evaluate the system by emulating a mobile backhaul as shown in

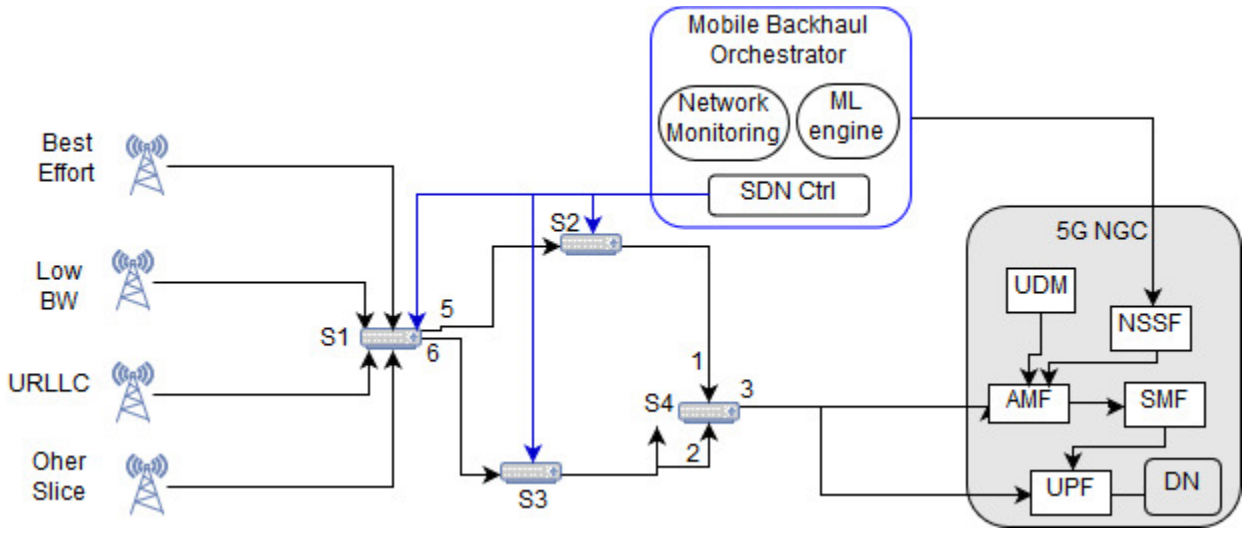


Figure 3-1. We use Minimet [129] to emulate the backhaul topology. The Mininet image version 2.2.2 which has 1GB RAM and 10 GB HDD was loaded in Oracle virtual box. The Mininet image has already installed OVS (Open Virtual Switch) [130], Ryu controller for managing the SDN switches.

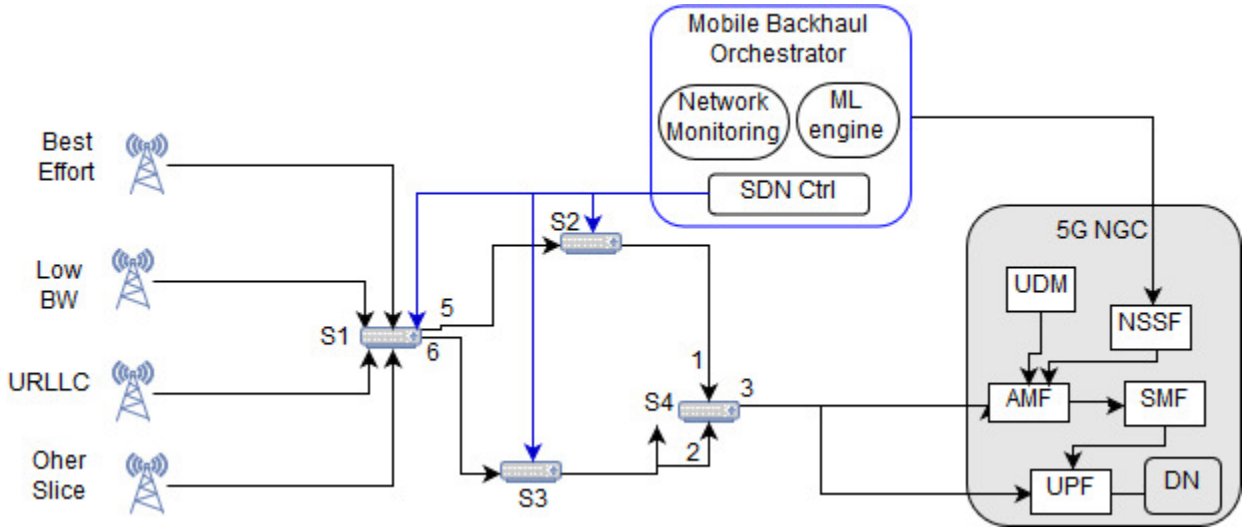


Figure 3-1: Network slice management using MBO with SDN switches.

The proposed network consists of four switches and four slices: URLLC (Ultra Reliable Low Latency communication), Low BW(IoT), best effort traffic, and other slices used for Mobile broadband traffic. Each traffic minimum rate assigned to each slice is 1Mbs and the traffic from different eNB is assigned to different slice which is then aggregated in the first switch S1. We use current ToS packet marking to prioritize the traffic from each eNB assigned to different network slice. Figure 3-2 shows that if the rate increases beyond the minimum rate of 1Mbps assigned to each slice, for example when performing tests with 600k, 1Mbps, 3Mbps and 5Mbps rates, the delay variation increases as the traffic increases. In this scenario, we take the

URLL traffic as a sample to show the delay variation when increasing the data bitrate while keeping other traffics on their minimum rate (1Mbps).

In Figure 3-2, we observe that increasing the rate to 3Mbps (green line) shows no packet drop, but the jitter increases reasonably. However, increasing the rate to 5Mbps (as shown in the light blue line) the jitter increases tremendously, and packet drop was observed. As time goes on, if traffic rate is increased, then the packet drop will be increased.

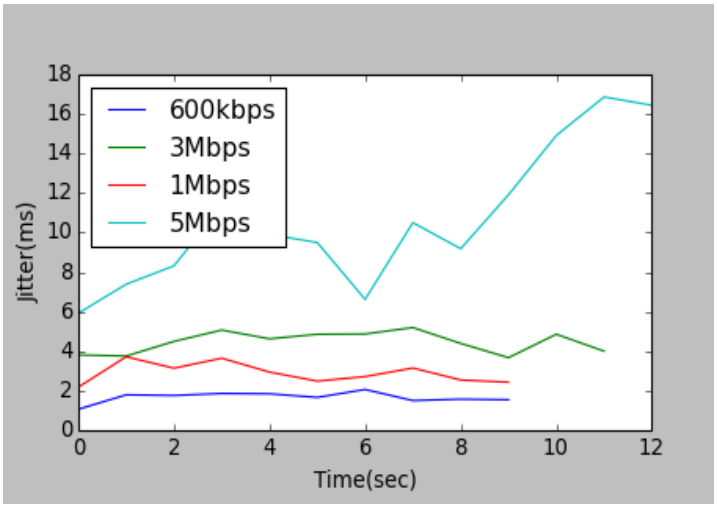


Figure 3-2: Delay variation with traffic increase

Therefore, if the network is managed using existing QoS techniques based on traffic prioritization using packet marking, the switch cannot ensure the required QoS. Thus, we require a mechanism enhanced with ML that will react and set new rules on the switches based on historical data.

From the different types of machine learning techniques available, we consider the supervised learning is suitable for managing the network resources. The idea behind supervised learning is that, for some inputs, we want to have certain value as an output. Thus, the supervised ML algorithms run based on the inputs received from the monitoring system until they get output values close enough to the target value which provides optimal usage of the network resources. Therefore, using this technique we evaluate if a given link is congested or not based on the network information such as jitter, bandwidth utilization, packet loss as an input.

In order to confirm whether ML has an impact in the QoS, we perform a similar test using an SDN controller that set new rules in the switches by supervised learning. We identify which ports are more exposed to congestion and apply traffic engineering on the switches connected to those ports. The previous topology figure shows that there are two routes from s1 to the server, through port 5 and port 6, and two routes for the reverse traffic from the server (core network), through port 1 and 2 at switch 5. First, the MBO network monitoring collects information from the traffic flow at both switches and the Machine Learning engine predicts how the network traffic behaves in different situations and perform different tasks when congestion happens to those ports. As packets start to arrive at the switches, the SDN monitor starts to collect traffic information periodically by sending port and flow stat requests to the SDN controller every 5 seconds. The SDN monitoring engine generates traffic statistics which describes the status of the ports of each switch.

The network monitor module sends all the collected data to the ML engine module. The ML engine starts to monitor the port status of each device and evaluate it against the set-up threshold values. For example, once statistical information of each device is collected, the AI engine calculates the bandwidth available in each port, how much packet loss is there, and what the current jitter is. The ML engine is set in such a way that if a link uses more than 50 percent of the available bandwidth at port 5 of switch 1 and at port 1 of switch 4, then the ML module will send a message to the controller to install new path and reroute the new arrivals of best effort traffic to port 6 in switch 1 and port 2 in switch 4, respectively.

We generate 1Mbps of URLLC traffic which is the minimum rate for that slice and 10Mbps traffic for the best effort traffic which is the maximum capacity of the link. We observe the bandwidth utilization at port 5, but in this case, the link was not congested enough to force the ML module to reroute the traffic. Initially, the available bandwidth at port 5 is not below the minimum threshold. Therefore, port 6 is still 100% free as there is no traffic going through the link. We purposely did not generate any traffic for the other two slices at port 6 in order to see the full availability of the link. Next, we start to increase the rate gradually and the AI module is still following the traffic every 5 seconds and calculating the necessary parameters. The ML engine evaluates against the set minimum requirement. It then acts once the values are below the threshold which forces the ML module to reroute the new traffic to port 6. The results show that initially, the link was not congested with a minimum of 1Mbps rate traffic for the URLL slice and 10Mbps for the best effort traffic. However, as the bit rate increases the jitter, latency also increases while the available bandwidth starts to get lower and lower and reaches below 50% of the available bandwidth. At this point, the ML is forced to act and reroute the new arrival best effort traffic to the route through port 6 as shown in Figure 3-3. This shows that ML will ensure the switches are managed properly and traffic is routed through different links or ports to ensure certain network slices get the necessary resources.

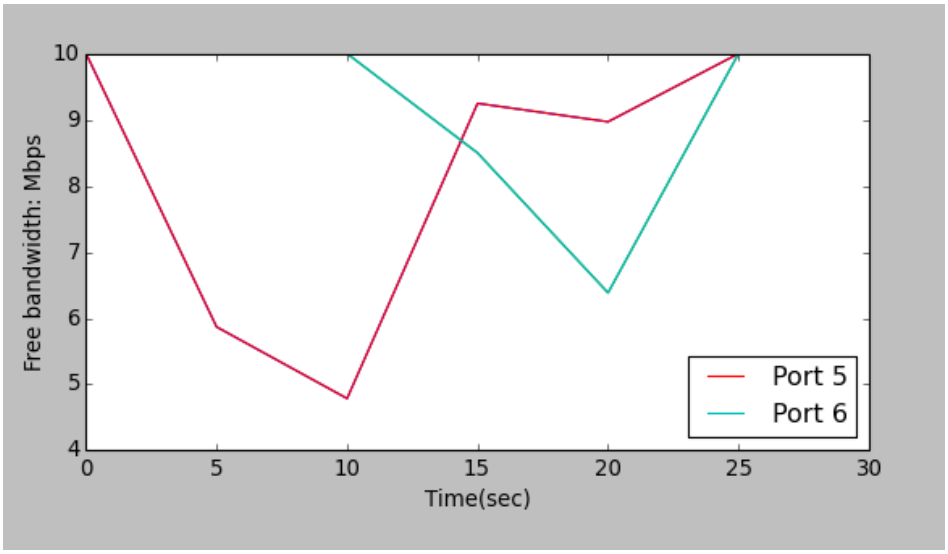


Figure 3-3: ML based on congestion takes the action to re-route the traffic.

3.2.1.1 Conclusion

In this section, we have analyzed the result of integrating supervised ML with SDN network management to deliver optimal usage of resources for network slicing. The SDN and ML is part of an orchestrator named

Mobile Backhaul Orchestrator that will provide the interaction between logical components defined in 3GPP with physical resources in the transport network.

3.2.2 Depth-Controllable SR Deep Neural Network

3.2.2.1 Summary

We propose a depth-controllable very deep super-resolution network (DCVDSR), which can dynamically select the depth to be used, from four layers to twenty layers in a single network, with only minimal additional parameters. Figure 3-4 shows the depth switching approach of the proposed DCVDSR. In addition, we propose and analyse a new training strategy using inverse auxiliary loss to prevent the performance from favouring a specific depth option while training DCVDSR in an end-to-end manner. Finally, we evaluate the performance at each depth option of DCVDSR by comparing DCVDSR with baseline models which have corresponding hidden layers. As a result, we achieve performances comparable to the baselines in all depth options of the DCVDSR through the proposed learning method, and even outperform the baselines in depth options deeper than eleven layers.

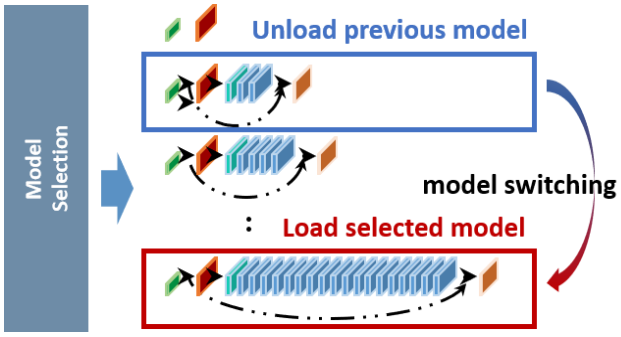


Figure 3-4: Conventional model switching

3.2.2.1.1 Deep super resolution network

Super-resolution is one of the fundamental and practical problems in the field of computer vision research. Single-image super-resolution (SISR) aims to produce a high-resolution (HR) image from a single low-resolution observation. SISR tries to reconstruct HR images from LR observations that are as close to the real images and as visually pleasant as possible. However, SISR is fundamentally a challenging problem due to the fact that a LR image can correspond to a set of HR images, while most of them are not expected.

Although deep-learning-based methods have recently achieved great success in super-resolution tasks, the existing studies have all contributed significantly to increasing either the depth of neural networks or the stability of training in deeper networks.

As the capability of a neural networks is proportional to the depth of the network, a trade-off typically exists between a network's depth and its performance and computational cost. Deeper networks perform better, but require more execution time and computational resources. Therefore, selecting a suitable model with proper consideration of these trade-offs has been an important factor in achieving system stability and efficiency. These trade-offs are equally applied to SR tasks. However, determining an optimal model statically can be difficult in some cases, such as when the state of the system or I/O rate changes during

operation, or when the system cannot be known in advance (e.g., applications for heterogeneous mobile devices). This issue can be handled this uncertainty by preparing batches of models at various scales and then switching them dynamically to fit the state of the system.

Figure 3-4 shows an example of this model-switching approach with batches of models. However, this approach may incur increased use of memory resources in proportion to the number of models and model-switching overhead, which should be resolved for practical applications. To overcome these limitations, this study aims to achieve the following research objectives: 1) to configure models with different depths as a shared single network in order to remove redundancies on their weights, and 2) to make the performance of the available depth options in a shared single neural network comparable to each model with a different depth.

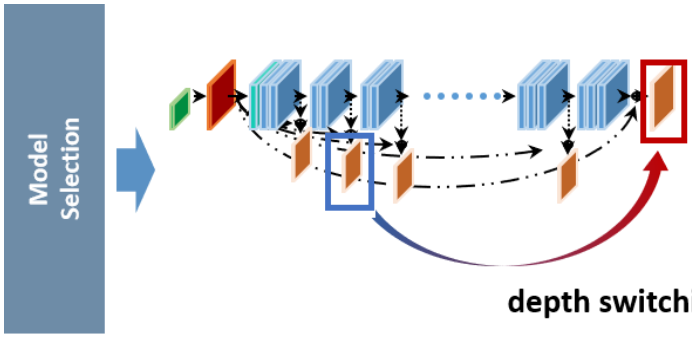


Figure 3-5: Proposed depth switching. Adding output branches in the middle of the hidden layers of VDSR.

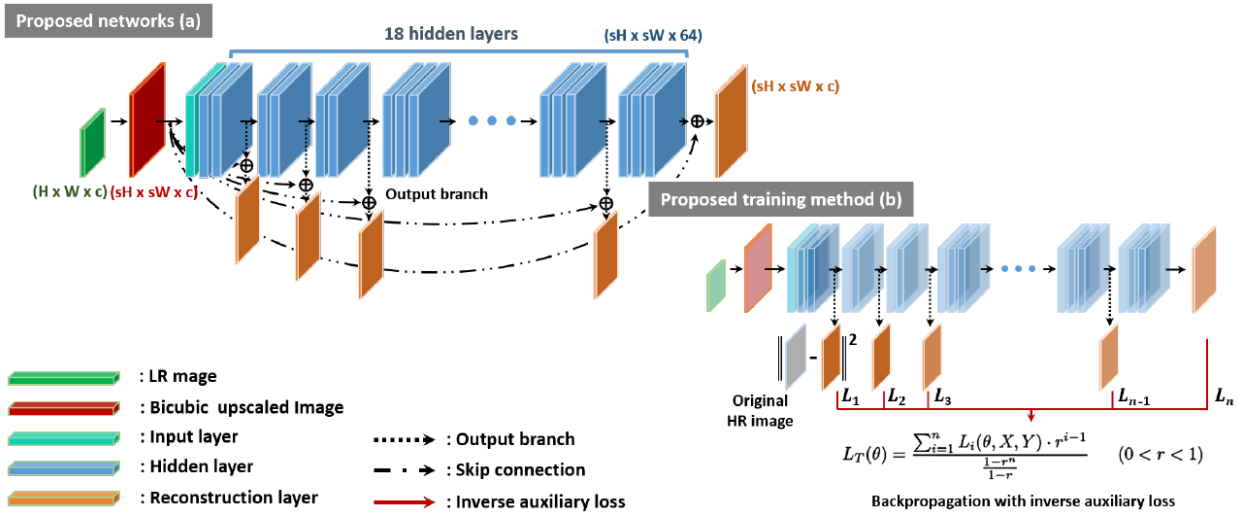


Figure 3-6: Architecture of DCVDSR. (a) DCVDSR contains several output branches which consist of convolutional layers that are the same as the output layer, and each output branch is connected with the interpolated low-resolution image, (b) our training strategy using inverse auxiliary loss.

3.2.2.2 Methodology

3.2.2.2.1 Baseline Architecture

We use the VDSR [105] architecture as a baseline because we focus on verifying the applicability and generality of the proposed approaches rather than outperforming existing state-of-the-art performances. In our framework, VDSR can be adopted easily due to its monotonous structure. Moreover, in spite of its simple architecture, VDSR still demonstrates good performance. In contrast to other approaches (especially post-upscaling approaches [108] [109], VDSR is flexible for different scale factors (from x2 to x4) which enables us to consider additional trade-off factors. Note that VDSR can serve as a connection point to further studies because it shares theoretical and structural properties with other works [106] [107].

3.2.2.2.2 Output branch

We propose adding output branches in the middle of the hidden layers of VDSR. Figure 3-6 (a) shows the overall architecture. These output branches consist of the 3 x 3 x 64 x c convolution filters, just like the output layer of VDSR, where c denotes the dimension of the target image. The output branches have two-fold responsibilities. First, the output branch transforms an intermediate feature map into a shape of the target image. Second, each output branch carries a different part of the weights, which cannot be shared across the entire network. Like conventional trade-off problems, the ratio of unbound weights increases as the depth of the output branch increases. Thus, the performance of each branch can be improved. However, this also causes the number of parameters to be increased. To solve this problem, we allow each output branch to have only one layer.

As previously mentioned, because the shape of the outputs obtained from each branch is the same as the target image, we calculate the losses from all branches and exploit them for training. This is conceptually similar to the auxiliary classifier of the Inception network [131] and the multi-scale loss[132] [133] proposed. The auxiliary classifier is used to alleviate the gradient vanishing problem in the lower layers as the network deepens by multiplying the loss of each auxiliary branch by the decay rate 0.3^*n*, where *n* denotes the order of the branch from the output layer. Unlike the Inception network and the multi-scale loss, each output branch in the proposed DCVDSR has the explicit role of outputting the image in each branch, so it seems more reasonable that the losses should be averaged rather than decayed. However, the averaged loss still contains some problems. To handle these problems, we propose a novel training strategy that uses the inverse auxiliary loss.

3.2.2.2.3 Contiguous and Progressive Skip Connection

VDSR uses a global skip connection to add the upscaled input image to the result of the output layer using bicubic interpolation, and this improves the stability of training. However, because other output branches exist in DCVDSR, distortion may occur among the branches during training, if skip connection is applied only to the last output layer. In the case of training with the loss at the last output layer, for example, the weights are trained to reflect only the difference from the input image due to the skip connection. However, in other output branches, training is performed to produce output images from scratch, so the extent to which weights can be updated may be relatively larger than that of the previous layer. Therefore, in order to maintain consistency and to share the weights properly, the skip connection should be applied to all of the branches. Figure 3-7 shows two ways of doing the skip connection method.

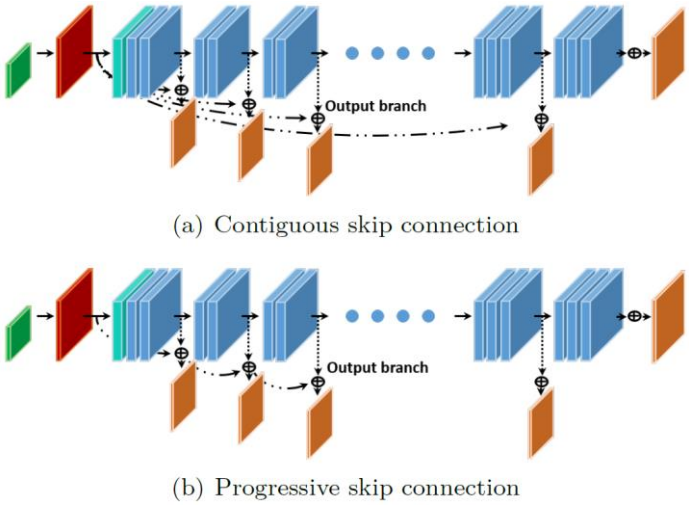


Figure 3-7: (a) contiguous skip connection, in which the global skip connection is applied to all of the output branches, as in the output layer; (b) progressive skip connection, in which the result of the previous branch is progressively added to the result of the next branch.

Figure 3-7(a) depicts the case in which global skip connection is applied to all of the output branches, i.e., the output layer. Figure 3-7(b) depicts the case in which the result of the previous branch is progressively added to the result of the next branch. We call these two approaches as *contiguous* and *progressive skip connections*, respectively. Equation 3-1 and Equation 3-2 denote the equations of the contiguous and progressive skip connections at the second output branch, respectively.

Equation 3-1: $$G_{cont}(X) = O_2\left(H_2\left(H_1(X)\right)\right) + X,$$

Equation 3-2: $$G_{prog}(X) = O_2\left(H_2\left(H_1(X)\right)\right) + O_1\left(H_1(X)\right) + X,$$

where X, G_{cont}, G_{prog}, O_2, and H_2 denote the upscaled input image, result of contiguous skip connection, result of progressive skip connection, output of n-th output branch, and output of the n-th hidden layer, respectively.

If the neural network is well trained, the result of $O_1\left(H_1(X)\right) + X$ in Equation 3-2 will be closer to the ground-truth image than X. Therefore, $O_2\left(H_2\left(H_1(X)\right)\right)$ in Equation 3-2 has an advantage by training a smaller residual than Equation 3-1. Nonetheless, the progressive skip connection performs worse than the contiguous skip connection, as shown in Table 3-1. This is presumably because the progressive skip connection is dependent on the previous branches, so the training at the shallower branches can be interrupted by the deeper branches. We reserve further analysis on this issue for future work and use the contiguous skip connection in this study.

3.2.2.2.4 Inverse Auxiliary Loss

The proposed model trained using the averaged loss demonstrates performance comparable to that of the baselines in the middle branches (depth 8 to 11) and even outperforms the baselines in the deeper branches (depth 14 to 20). However, in the shallow branches (depth 4 to 6), considerable performance degradation occurs compared to the baselines (over 0.2dB). As in the Inception network, the loss of the

shallow branches can help to improve the stability of training at the deeper branches, while the convergence of the shallower branches can be disturbed by the deeper branches. Therefore, in order to achieve performance comparable to the baseline over the entire interval, we need an advanced approach of training the shallower branches explicitly.

For this purpose, it is worth considering stacked transfer learning, which trains a shallow layer and then sequentially trains other hidden layers and branches while freezing the pretrained shallower one. This approach is advantageous because it makes the training procedure at shallower depths more explicit, and the training at the deeper branches proceeds after the shallower branches have completely converged. However, the sharing of weights with the deeper layers cannot be taken into account in this approach. Thus, it is difficult to expect performance improvement as the network becomes deeper (see stack transfer (hard) in Table 3-2, in which there is no performance improvement from depth 8).

It is effective to stop the pre-training at shallower branches earlier, before the network completely converges. Stopping early causes the deeper branches to share the weights while preventing the weights from fitting too hard to the shallower branches. The *stacked transfer (soft)* of Table 3-2 shows promising results, with the performances slightly degrading at the shallower branches and increasing at the deeper branches. Thus, we propose an *inverse averaged loss* to take advantage of both the averaged loss and the stacked transfer learning. The proposed method is shown in Figure 3-6(b).

The equation of inverse auxiliary loss is as follows:

$$L_T(\theta) = \frac{\sum_{i=1}^{n} L_i(\theta, X, Y) \cdot r^{i-1}}{\frac{1-r^n}{1-r}}, \qquad (0 < r < 1),$$

where θ, X, Y, r and n denote the parameters of neural network, the upscaled input image, target labeled image, decay coefficient, and the number of total branches, respectively.

Model	Depth						
	0 (Bicubic) PSNR/SSIM	4 (SRCNN) PSNR/SSIM	6 PSNR/SSIM	8 PSNR/SSIM	11 PSNR/SSIM	14 PSNR/SSIM	20 (VDSR) PSNR/SSIM
Baseline (original)	30.4 / 0.8682	32.7 / 0.909	-	-	-	-	33.6 / 0.921
Baseline (implement)	30.4 / 0.8682	32.56/0.91	33.01/0.916	33.229/0.918	33.379/0.92	33.435/0.92	33.523/0.921
Averaged loss (Cont)	30.4 / 0.8682	32.19/0.9058	32.79/0.9128	33.11/0.9164	33.3/0.9187	33.48/0.9208	33.59/0.9222
Averaged loss (Prog)	30.4 / 0.8682	32.182/0.905	32.716/0.912	33.06/0.916	33.231/0.917	33.356/0.919	33.446/0.92

Table 3-1: Results of contiguous and progressive skip connection. We compare our methods against two baselines, the original VDSR and the re-implemented version of VDSR using TensorFlow. To compare the shallowest networks, we use SRCNN, which is based on VDSR.

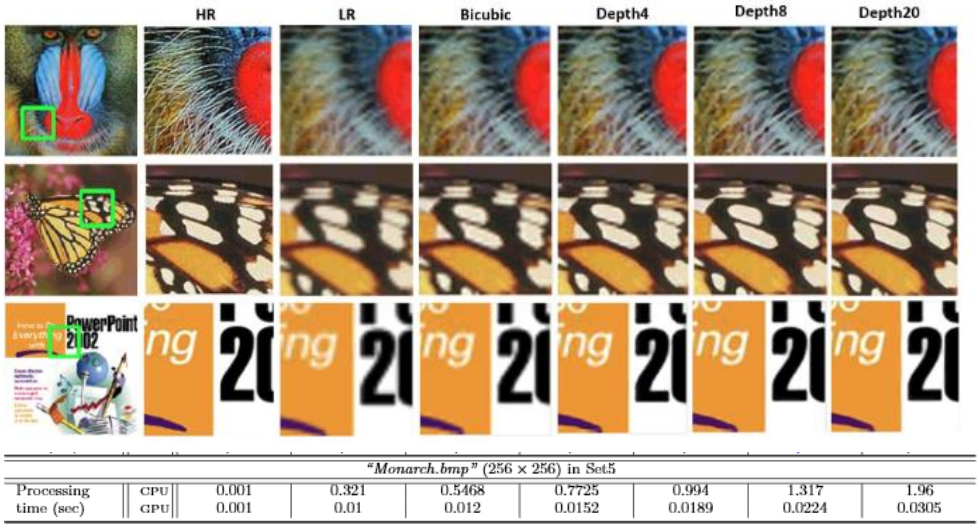
Model	Depth						
	0 (Bicubic) PSNR/SSIM	4 (SRCNN) PSNR/SSIM	6 PSNR/SSIM	8 PSNR/SSIM	11 PSNR/SSIM	14 PSNR/SSIM	20 (VDSR) PSNR/SSIM
Baseline (original)	30.4 / 0.8682	32.7 / 0.909	-	-	-	-	33.6 / 0.921
Baseline (implement)	30.4 / 0.8682	32.56/0.91	33.01/0.916	33.229/0.918	33.379/0.92	33.435/0.92	33.523/0.921
Averaged loss	30.4 / 0.8682	32.19/0.9058	32.79/0.9128	33.11/0.9164	33.3/0.9187	33.48/0.9208	33.59/0.9222
Stacked transfer (soft)	30.4 / 0.8682	32.47/0.91	32.86/0.9140	32.952/0.9148	32.959/0.9149	32.96/0.9149	32.96/0.914
Stacked transfer (hard)	30.4 / 0.8682	32.56/0.91	32.68/0.9121	32.72/0.9125	32.724/0.9125	32.73/0.9126	32.73/0.9126
Inverse aux loss (0.5)	30.4 / 0.8682	32.43/0.9093	32.97/0.915	33.18/0.9172	33.38/0.9195	33.49/0.9207	33.57/0.9214

Table 3-2: Quantitative results of stacked transfer learning approaches. Stacked transfer (soft) denotes the

results obtained by transferring the early stopped pre-trained model (20 epochs) to lower layers of a deeper network. Stacked transfer (hard) denotes the results obtained by transferring the hardly converged pre-trained model (50 epochs). When the PSNR score is equal to or better than the baseline (our implementation), the results are presented in blue. When the PSNR score is less than 0.3dB compared to the baseline (original paper), the results are presented in red.

3.2.2.3 Evaluation

We conduct a qualitative evaluation of the proposed DCVDSR with the following considerations: 1) whether the proposed model has a visually plausible ability to upscale low-resolution images, and 2) whether the proposed model possesses trade-off factors, i.e., its visual quality is improved progressively as the depth of layers becomes deeper. We perform up-scaling using a different branch of each depth (0, 4, 8, 20) for patches of three images, namely, baboon, monarch butterfly, and PPT3, which are widely used for evaluating super-resolution tasks. Note that it is equivalent to the bicubic interpolation method, especially at a depth of 0. We firstly obtain low-resolution images down-scaled by bicubic interpolation methods with a scaling factor 3 from the original images, and then we upscale them using DCVDSR with the corresponding scaling factor. As shown in Figure 3-6, the output branch at the shallowest depth performs significantly better than the bicubic interpolation and the quality gradually improves as the layer becomes deeper. The improvement is distinct from that in the shallow layer; however, as the layer becomes deeper, the difference can be compromised according to the purpose. Therefore, for practical purposes, it may be reasonable to use only up to 11 layers when computational resources are insufficient.



"Monarch.bmp" (256 × 256) in Set5								
Processing time (sec)	CPU	0.001	0.321	0.5468	0.7725	0.994	1.317	1.96
	GPU	0.001	0.01	0.012	0.0152	0.0189	0.0224	0.0305

Figure 3-8: Results of qualitative evaluation. Evaluation was performed at a different branch of each depth (0, 4, 8, 20) for patches of three images, namely, baboon, monarch butterfly, and PPT3. Note that it is exactly equivalent to the bicubic interpolation, especially at a depth of 0. Low-resolution images are obtained from original images by down-sampling using the bicubic interpolation method with a scaling factor 3, and then up-sampling was performed using DCVDSR with a corresponding scaling factor.

3.2.2.4 Conclusion

The proposed depth-controllable very deep super-resolution network has a convenient property that can scale the processing speed and image quality, given a low-resolution input image, by controlling the depth or the number of hidden layers of a single convolutional neural network. The output images from the

intermediate hidden layers are enhanced as the number of intermediate layers increases. The network at a specified depth (e.g., twenty layers) is successfully trained while also being able to output up-sampled images at the intermediate stages. For example, the presented intermediate stages are layers 4, 6, 8, 11, and 14. The proposed network at those depths is compared to the baseline networks, mainly to VDSR and SRCNN at a depth of 4, at three scaling factors (x2, X3, x4). We also propose a novel training strategy that uses the inverse auxiliary loss and contiguous/progressive skip connections. The proposed training principles help in training the network end-to-end without biasing the performance at a specific layer. In conclusion, the proposed network was designed so that within a single network, shallower parts of the network work independently of deeper parts. We believe this modular characteristic is especially suitable for practical applications with limited computational resources.

3.2.3 Auction-based Resource Allocation using Deep Learning

3.2.3.1 Introduction

The possibility to use commercial drones in a broad range of applications is being extensively studied by the research community, and they are expected to man operations in remote location. In general, commercial drones have inherent limitations in the amount of energy available to support their operations. This is due to the energy/weight ratio of current energy storage technologies, where increasing the capacity of the battery beyond a certain point degrades flight time due to excessive weight. As a consequence, effective battery management is one of the main enablers of practical deployments of drone-based technologies and applications. Importantly, in applications requiring extensive flight time, the energy constraint problem cannot be solved by only optimizing power consumption. Thus, charging during the completion of long-term tasks has been proposed to extend the operational range of the drones.

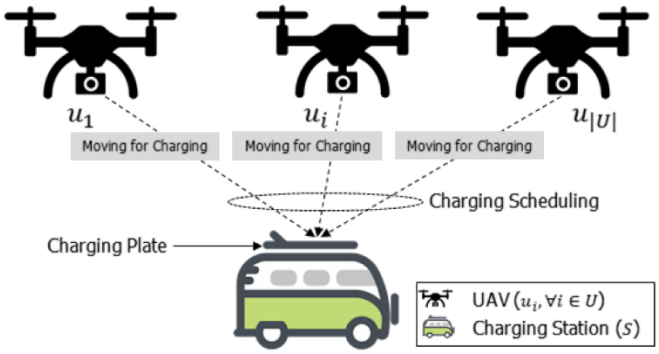


Figure 3-9: Multi-Drone Network Model for Mobile Charging stations

We consider a scenario where multiple drones compete to access the services provided by a mobile charging station. The proposed framework controls the charging process of the drones, where the charging station takes the role of leader in the distributed drone-charging system, and coordination within the system is supported by Internet-of-Vehicle (IoV) networking functions. We take an econometric approach, where the problem of controlling the scheduling is formulated as an auction, whose objective is to maximize the utility of the drones (i.e., the difference between payment and bid during auction computation) as well as the station's revenue (i.e., payment received by the drones through charging scheduling). In the drone network model considered, the drones are assumed to be non-cooperative, that is, they operate independently and in a distributive manner. Furthermore, the mobile charging station is not assumed to know the exact true values associated with each drone, which becomes available only when the actual values are submitted. The auction approach we take is especially suitable to solve the problem of assigning time slots to drones in this information-limited system. Among the various auction formulations available (e.g., ascending auction, descending auction, first price auction, second price auction), we choose a second price auction formulation, where the highest bidder wins, but the price paid is set to the second highest bid.

In the considered system model, the mobile station is the auctioneer and owner/seller of the resource (that is, the charging time slot) and each drone is considered as a buyer. The drones are in competition for scheduling battery charging with price bidding via its own private valuation for auction. As auctioneer, the mobile station (i) receives all bids from the drones, (ii) calculates the charging time allocation probabilities and payments, (iii) assigns the charging time to the drone (i.e., the winner in auction) who bids the highest value, corresponding to the largest allocation probability, (iv) announces the value which should be paid by the winner drone, and (v) receives the payment.

During the auction, drones strategically submit bids to increase their profits, i.e., utility. Similarly, the resource-owned auctioneer is not a sacrificial seller; thus, it is required to consider the revenue

auctioneer, i.e., profitable. Therefore, revenue-optimal auctions have been considered as one of major objectives in auction design. Among various auction algorithms, Myerson auction is one of the most efficient revenue-optimal single-item auctions. The auction transforms the bid value, and then the winner and payment are determined based on the transformed bid. At that point, if the transformation function is monotonic, the revenue-optimal auction is configured. Therefore, the proposed auction designs the revenue-optimal auction based on the concept of the Myerson auction.

The charging scheduling system of the drones is still in the early stages of research. Key properties of the system such as drone location distribution and residual energy distribution have not yet been fully characterized in the literature. Therefore, a system that can extract the desired data (i.e., distribution of drones) from the actual system without prior knowledge or assumptions is desirable. Therefore, this paper takes advantage of deep learning to learn important features on-the-fly from the operating environment. Recently, frameworks combining game theory and deep learning have been active subject of research. Results illustrate applications of such approach in various domains. The key is that deep learning can automatically extract and learn important features from data, and it has been widely demonstrated that neural network structures can approximate complex non-linear functions.

3.2.3.2 Deep Learning Based Auction Design

3.2.3.2.1 Deep Learning Networks

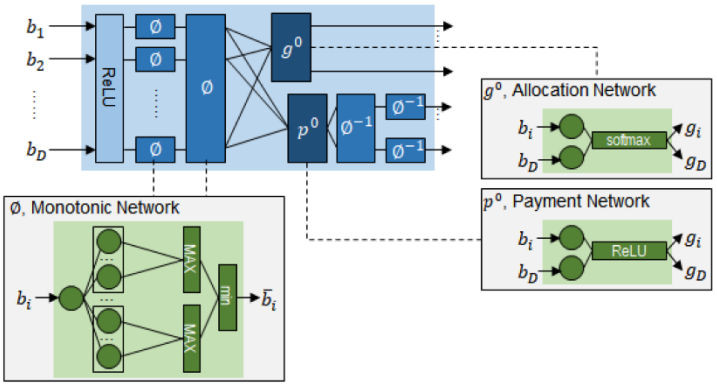


Figure 3-10: Proposed Deep Learning Framework (Revenue Network) for Revenue-Optimal Auction Computation

In this research, the virtual valuation function is replaced by the two-layer network $Ø^{mononet}$, composed of the monotonic networks.

Monotonic Network

As shown in Figure 3-10, the monotonic network is a three-layer deep neural network. The input layer is configured with multiple groups composed of sets of linear units. The maximum value of each group is calculated in the second layer. The last layer selects the minimum value of the given output of the second layer. As the name suggests, the monotonic network is monotonic, and this characteristic is preserved regardless of the number of groups, units, and the order of min/max operations.

Virtual Valuation Network

The virtual valuation function in the Myerson auction is replaced with the monotonic network. The computations of $Ø^{shared}$ and $Ø_i$ are implemented as follows.

Equation 3-3: $\overline{b_i} = Ø_i^{shared}(b_i') = min_{1\le g\le \mathcal{G}}\{max_{1\le n\le \mathcal{N}}(w_{g,n}^{shared} b_i' + \beta_{g,n}^{shared})\}$

Equation 3-4: $b_i' = \phi_i(b_i) = min_{1\le g\le \mathcal{G}}\{max_{1\le n\le \mathcal{N}}(w_{g,n}^{i} b_i + \beta_{g,n}^{i})\}$

The bid b_i of the drone u_i is transformed to $\overline{b_i}$ via the virtual valuation network $\emptyset^{mononet}$. In the $\emptyset^{mononet}$, all outcomes of $\emptyset^{shared}$ are calculated on the same weights, whereas the $\emptyset_i$ calculates the outcome using different weights for each bid. The inverse computation of $\emptyset^{mononet}$ is denoted by $\emptyset^{-1}_{mononet}$. The $\emptyset^{-1}_{mononet}$ that determines the payment of the winner drone u_i which is composed of two networks. In the computation of $\emptyset^{-1}_{mononet}$, the weights of $\emptyset^{mononet}$ are used. Thus the computations of two layers can be expressed as follows:

Equation 3-5: $$p_i' = \emptyset^{-1}_{shared}(\overline{p_i}) = max_{1\le g\le \mathcal{G}}\left\{min_{1\le n\le \mathcal{N}}\left(w^{shared}_{g,n}\right)^{-1}\left(\overline{p_i} - \beta^{shared}_{g,n}\right)\right\}$$

Equation 3-6: $$p_i = \emptyset_i^{-1}(p_i') = max_{1\le g\le \mathcal{G}}\left\{min_{1\le n\le \mathcal{N}}\left(w^{i}_{g,n}\right)^{-1}\left(p_i' - \beta^{i}_{g,n}\right)\right\}$$

The payment $\overline{p_i}$ of the drone u_i is transformed to p_i via the $\emptyset^{-1}_{mononet}$. The $\emptyset^{-1}_{mononet}$ consists of $\emptyset_i^{-1}$ and $\emptyset^{-1}_{shared}$. The same weights are used to calculate all outcomes of $\emptyset^{-1}_{shared}$. The outcome of $\emptyset_i^{-1}$ is calculated based on different weights for each bid, as shown in Equation 3-6.

The monotonic network is responsible for the transformation of the virtual bid $\overline{b_i}$ in auction. As mentioned above, the optimal revenue is equivalent to the optimal virtual surplus. Thus, the monotonic network is a major component of the auction. However, in order to configure the revenue-optimal auction, the additional allocation and payment rule networks are required. In this work, the payment and allocation rules are configured with ReLU and softmax functions, respectively, which have been mainly used in deep learning as activation functions. These make backpropagation easy during the training process.

Allocation Rule Network (g_i)

This section describes in more details the structure of the allocation rule network (g_i). In this paper, since this allocation rule network is implemented using a deep neural network, the probability is calculated using softmax which converts the input vector into a probability vector. The allocation rule network g awards the charging service to the highest bidder drone; thus, the highest probability is assigned to the highest bidder. The continuous function traditional auction is approximated using the deep network, which converts the input vector $\overline{b}$ to the probability vector. In the SPA auction with reserve price 0 (SPA-0), the allocation rule network assigns the highest winning probability to the highest bidder whose transformed bid $\overline{b_i}$ is greater than 0, $\overline{b_i} > 0$. The softmax based assignment can be calculated as follows:

Equation 3-7: $$g_i = softmax\left(\overline{b_1}, \dots, \overline{b_U}; k\right) = \frac{e^{k\overline{b_i}}}{\sum_{j=1}^{U} e^{k\overline{b_j}}}$$

The parameter k is a constant value and it determines the quality of the approximation. As k increases, the quality of the approximation increases, whereas the smoothness in the allocation network decreases. For simplicity, this means that the higher k makes a large difference between the allocation probabilities of users. When the networks are trained to minimize the loss function, the value of $g_i(b_i)$ increases. As a result, since the profit of the auctioneer is related to the second highest $g_i(b_i)$, it is also a function of the parameter k.

Payment Rule Network ($\overline{p_i}$)

This section describes the structure of the payment rule network ($\overline{p_i}$). The ReLU is widely used in deep learning computation as an activation function. In the proposed auction, the payment p_i of drone u_i is calculated from the transformed bid $\overline{b_i}$. Before the computation of $\emptyset^{-1}_{mononet}$, the deep network excludes the bid below the reserve price 0 via $\mathrm{ReLU}(\overline{b_i}) \triangleq \max(\overline{b_i}, 0)$. The input $\overline{b_i}$ is the second highest transformed bid which is the output of $max_{j\neq i}(\overline{b_i})$.The payment rule network can be then calculated as:

Equation 3-8: $$\overline{p_i} = ReLU\{max_{j\neq i}(\overline{b_i})\}$$

and the result $\overline{p_i}$ is used as an input of Equation 3-5, i.e., the actual payment of winner drone.

3.2.3.2.2 Overall Auction Mechanism

Input : t, f, Bid sets $\boldsymbol{b} \triangleq (b_1, \ldots, b_U)$
Output: allocation probability set $\boldsymbol{g_i} \triangleq (g_1, \ldots, g_U)$,
payment set $\boldsymbol{p_i} \triangleq (p_1, \ldots, p_U)$

1 **while** *Mobile charging system is* ***idle*** **do**
2 ▷ Drones: charging scheduling valuation v_i;
3 ▷ Drones: submit bid b_i;
4 ▷ $b'_i = \phi_i(b_i) = \min_{1 \le g \le \mathcal{G}} \left\{ \max_{1 \le n \le \mathcal{N}} (w^i_{g,n} b_i + \beta^i_{g,n}) \right\}$;
5 ▷ $\overline{b_i} = \phi_{shared}(b'_i)$;
6 ▷ $g_i = \texttt{softmax}\left(\overline{b_1}, \ldots, \overline{b_U}; k\right) = \frac{e^{k\overline{b_i}}}{\sum_{j=1}^{U} e^{k\overline{b_j}}}$;
7 ▷ $\overline{p_i} = \texttt{ReLU}\left\{\max_{j \neq i}(\overline{b_i})\right\}$;
8 ▷ $p'_i = \phi^{-1}_{shared}(\overline{p_i})$;
9 ▷ $p_i = \phi_i^{-1}(p'_i) = \max_{1 \le g \le \mathcal{G}} \left\{ \min_{1 \le n \le \mathcal{N}} (w^i_{g,n})^{-1} (p'_i - \beta^i_{g,n}) \right\}$;
10 ▷ Calculate winner and payment $(\boldsymbol{g_k}, \boldsymbol{p_k})$;
11 ▷ Winner Drone: Pay payment;
12 ▷ Allocate charging system to the winner;
13 **end**

Algorithm 1: Deep Learning-Based Algorithm for the Auction Controlling the Charging Scheduling

The overall deep learning-based auction mechanism is summarized in Algorithm 1. If the mobile charging system becomes idle, the auction is initiated (line [1]). The valuation v_i for the charging time is computed by each drone u_i based on its own private criteria. Then, based on the individual private valuation, each drone submits its bid b_i (line [2 - 3]). The mobile charging station runs the auction using the pre-trained networks. If $\overline{p}$ = 0, then all the drones assign a low valuation to the charging time and the mobile charging system does not allocate the charging time to users. If there exist bids which are larger than reserve price 0, the corresponding allocation and payment probabilities are calculated using the proposed deep learning networks, i.e., virtual valuation network, allocation rule network, and payment rule network (line [4 - 10]). Because the proposed deep learning auction is the variant of SPA-0, any bid below the reserve price 0 is converted to 0 (line [7]). As shown in line [11], the mobile charging station assigns the payment of p_i to the drone u_i with the highest g_i. Finally, the mobile charging station allocates the charging time to the winner drone u_i (line [12]). The drone then reaches the charging station and occupy it for the duration of the slot. After the winner drone leaves the charging station, next iteration starts if the mobile station is idle.

3.2.3.3 Future Work

The proposed deep learning based auction is revenue optimal for mobile charging scheduling in distributed multi-drone networks. The mobile charging scheduling problem is interpreted as auction problem where each drone bids its own valuation and then the charging station schedules drones based on it in terms of revenue-optimality. Through the proposed deep-learning based solution approach, the charging auction enables efficient scheduling by automatically learning the required knowledge (i.e., bids distribution), which is required in conventional auction mechanisms. Therefore, environmental information is not required anymore in auction computation. This makes effective troubleshooting possible in distributed multi-drone networks. The proposed algorithm only requires payment and allocation probabilities by the multi-drones. The loss function in deep learning computation is an important factor that allows the proposed action to be constructed based on environment independent information. As verified via software prototype based performance evaluation, following facts are observed: (i) guaranteeing optimal revenue in terms of individual rationality and dominant strategy

incentive compatibility, (ii) limiting the false bids of drones by increasing the payment to the false-bid drones, and (iii) enabling a revenue optimal auction to be constructed without complex prior knowledge, i.e., bids distribution.

As future research directions, advanced auction mechanism designs with multiple mobile charging stations are worthy to consider. In this case, the problem can be formulated with multi-item auction and then the corresponding mathematical formulation, verification, and analysis are desired. Furthermore, the proposed deep learning-based auction mechanisms can be advantageous in various applications. For example, visual attention is considerable because it can be reformulated as resource allocation.

3.2.4 AI-based Quality-Aware Transmission

3.2.4.1 Introduction

Intelligent firefighting is an area which provides immersive video service, the involved applications mainly depend on the computer vision technology and continuously generate object recognition tasks with huge volume of data created by drones (i.e., extracting and recognizing objects from video streaming captured by drone cameras). These object recognition tasks consume a lot of computing resources and are extremely sensitive to latency. However, processing these tasks locally may be slow because the computing resources equipped in drones are limited (e.g., space, weight limitation and economic consideration) [134] while uploading them to cloud is not applicable due to the remarkable transmission delay.

To conform the firefighting environment where operations are latency-critical, we apply a novel computing paradigm, vehicular fog computing (VFC) [116] [117] [118] [135] , to address the challenges in immersive video service. Its key idea is to push large amount of intelligence (i.e., computing and communicating resources) to the fog nodes (such as firefighting trucks and small cells) near where the visual-based application tasks are being generated and enable drones to offload the generated tasks to the fog nodes. With shorter physical distance and less communication hops, the VFC can greatly reduce the communication latency.

When considering providing immersive video service, a good quality (e.g., high resolution) of the video stream is required to detect small objects. Besides, the service latency, including video transmission and processing latency, is expected to be reduced. However, there is an anti-correlation relationship between the resolution of the video stream and the service latency. Specifically, the higher the video resolution, the larger the data size and the higher the transmission latency. Furthermore, higher video resolution involves more pixels and higher processing latency. In this section, we propose Chameleon, a service latency and video quality balanced task allocation scheme that allows the drones to offload object recognition tasks to nearby fog nodes.

However, the computing resources equipped on the fog nodes are finite and the workload of fog nodes need to be considered. In Chameleon, we suppose there exists only one type of immersive video service application and the computing demand from each drone is equal. Thus, the workload of a fog node is directly affected by the number of surrounding drones. In firefighting, there may be several fog nodes surrounding a drone at the same time and the drone need to select an appropriate one for offloading service. We design a dynamic task allocation solution to optimize service latency and quality under the application-specific requirements, e.g., communicating and computing demands. The optimization objectives include both minimizing the average service latency and reducing the overall quality loss. As it proves to be an NP-hard problem, an event-triggered task allocation architecture, i.e., Dynamic Task Allocation (DTA) is proposed using Binary Particle Swarm Optimization (BPSO) to solve it.

In this section, we first describe the multi-drone firefighting scenario and explore the profiles of the applications which are based on video transmission and procession. Then, we introduce the BPSO based task allocation before we conclude.

3.2.4.2 Firefighting Scenario

In this section, we describe the firefighting scenario with multi-drones. The related terms are defined and an overview of the process of Chameleon is given.

3.2.4.2.1 Related Terms

1. **Fog nodes.** In Chameleon, we consider two types of fog nodes: 1) infrastructural fog nodes – the computing nodes collocated with network infrastructures (e.g. 5G cell towers), and 2) vehicular fog nodes – the computing nodes carried by moving vehicles (e.g. firefighting trucks) with on-board opportunistic communication modules (e.g. Wi-Fi and DSRC).

2. **Tasks:** The process of an immersive video application can be broken down into a set of services. For example, search and rescue survivors in a firing area includes services such as object recognition and video streaming. Services are independently deplorable and each service has its own latency, quality constraints, and workload profiles. In Chameleon, a task refers to a service instance. For instance, suppose there is a drone demand for recognizing potential survivors from a video clip taken by its camera and requires offloading this service instance to fog nodes due to its limited computing ability. Then, the task in this service instance consists of video uploading, video preprocessing, feature extraction, pattern matching, and results downloading. The task is the basic unit for task allocation. That is to say, from the perspective of task allocation, a task cannot be divided into sub-tasks.

3. **Service Zones**: We assume that the firefighting area is completely covered by cellular networks. In similar manner [136], an urban area is divided into service zones and a stationary fog node within a zone is selected to manage and coordinate all the fog nodes in the same zone. The coordinator is called **zone head**. For simplification of the system model, a Long-Term Evolution (LTE) base station is always selected to be the zone head and assume that mobile fog nodes are deployed on commercial fleets, e.g., taxis and buses. With the existing cellular registration mechanisms, mobile fog nodes always inform the zone head as they enter or leave the zone. In addition, they regularly report their moving directions, locations, and available capacities to the zone head. Note that locations and dynamics of fog nodes, drones, as well as base stations are visible to Chameleon.

3.2.4.2.2 Process of Chameleon

The task allocation process in Chameleon is manifested in Figure 3-11. The whole process consists of 3 steps as shown below.

1. **Mobile fog nodes discovering**. In the initial phase, a drone needs to determine which mobile fog nodes are within its communication range. It broadcasts one-hop probe messages through DSRC and collects responses from fog nodes. Fog nodes that respond are included in the list of fog candidates. As shown in Figure 3-11, the fog candidates of drone A are the fog nodes within the communication range of A.

2. **Requests sending.** After the fog candidates are discovered, the drone sends a request to the zone head via LTE. The request contains information about the tasks to be offloaded to fog candidates.

   (a) *Task ID*: The unique ID of a task.

   (b) *Task Profiles*: Description of the generated workload and the task-specific constraints, such as tolerable latency, supported video resolutions, and data size.

   (c) *Task Generator*: The drone that generates data and sends the request.

   (d) *Fog Candidates*: The fog nodes within the communication

3. **Assignment of tasks to fog candidates**. When receiving a request from any drone, the zone head executes the task allocation algorithm to decide where to run the tasks. Concerning the frequent changes in network topology, the zone head would estimate the service paths and filter out the fog nodes whose computing resources are fully occupied. Furthermore, with the existing cellular

registration mechanisms, the zone head would check the positions of mobile fog nodes periodically to avoid out-of-date information.

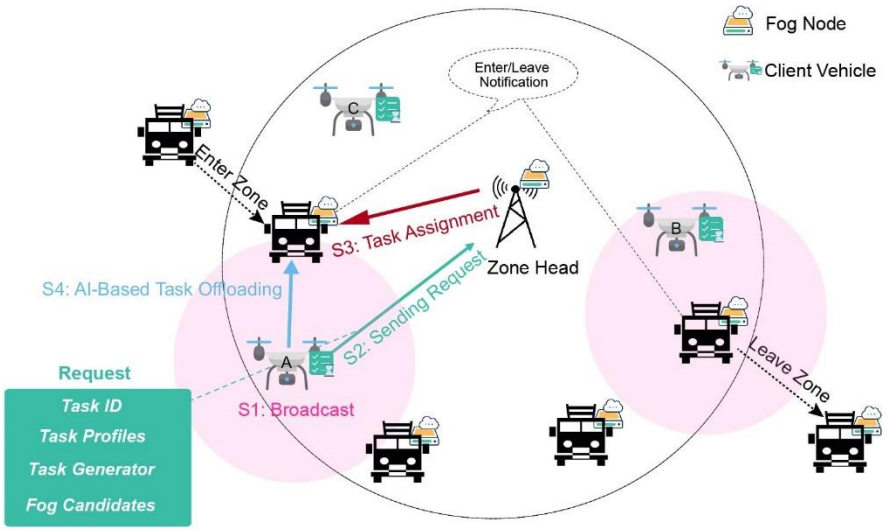


Figure 3-11: Multi-drone Firefighting Scenario

3.2.4.3 Video-Based Application Profiling

To get the immersive video service application profiles in real world environment, we first measure the transmission latency of tasks under various frame rates with different resolution. Then, we adopt a real-time object detection application to explore processing latency of tasks under different devices with various workloads.

3.2.4.3.1 Transmission Latency

To profile the transmission latency of the tasks with different resolution, we implement a video streaming application based on WebRTC. Furthermore, as shown in Figure 3-12 (left), we place a server laptop in a parking vehicle and a client laptop in a moving vehicle and connect them with an 802.11ac router. We let the client laptop continuously transmit video streaming with different resolutions, frame rates, and communication distance.

Figure 3-12 (center) illustrates the transmission latency of tasks with different resolutions at frame rate 5fps. We can see that the task transmission latency stays at a certain level (less than 25ms) when the resolution is 360p and increases rapidly (from 20ms to 65ms) when the resolution is 720p. In firefighting scenario, it is hard to predict the communication distance variation between the drones and fog nodes. Thus, with 802.11ac access technology, we set the transmission latency of tasks with 360p resolution as 25ms and that of tasks with 720p resolution as 65ms. We assume that the transmission latency of 5G access technology can achieve 100mbps within 90m coverage radius [137]. By calculation, we set the transmission latency of tasks with 360p resolution with 5G access technology as 10ms and that of tasks with 720p resolution as 20ms.

3.2.4.3.2 Processing Latency

To profile the processing time of the tasks under different fog node workload, we use a real-time object detection application, YOLO, for the use case in the simulation [138]. To test the processing latency with different computing resources, we run YOLO on a Linux-OS desktop (with Quadro K2200 CPU, 4GB memory) and a GPU server (NVIDIA Corporation GV100, 16GB memory) to extract the objects in 1000 images with 360p and 720p resolution, respectively. Due to the memory limitation, one YOLO process can be operated in the desktop and five in the GPU server. According to Figure 3-12 (right),

the processing latency of tasks processed by the GPU server outperforms that of tasks processed by the desktop to a large extent on both 360p and 720p resolutions. However, when the load of GPU server is heavy (more than five progresses), the new incoming task needs to wait for the earlier tasks to be processed.

For immersive video service specification, we suppose the service latency (transmission latency and processing latency) cannot exceed 200ms. According to Figure 3-12 (center and right), we set that one fog node can allow at most ten drones connect to it simultaneously. Because in this case, the worst situation is that one task with 720p is offloaded to the fog node when there are already five tasks in the fog node memory. Then the service latency of that task is equal to the summation of transmission latency, waiting latency and processing latency, which is 60 + 65 + 65 = 190 milliseconds (the worst case is that the one of the previous tasks is with 720p resolution).

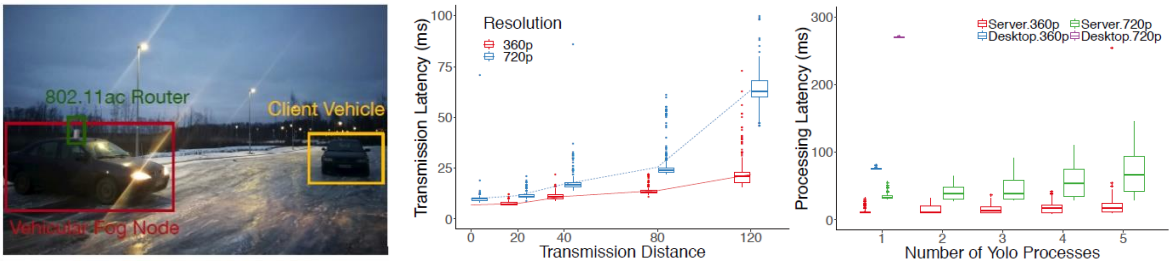


Figure 3-12: Application Profiling. (a) Experiments scenario; (b) Transmission latency; (c) Processing latency

3.2.4.4 BPSO-Based Task Allocation

In this section, the system model of Chameleon is presented. K_i is defined as the set of tasks generated by drone i, and J_i is defined as the set of fog candidates for drone i. A binary variable x_{ij} is defined to indicate whether the task *k* is generated by *i*, and another binary variable x_{ij} is defined to indicate whether fog node j is available for i (i.e., the fog node j is in the list of fog candidate). In the optimization, we try to find the value of x_{ij} (x_{ij} is a binary variable), which indicates the assignment of tasks. We propose a heuristic algorithm BPSO based on Partial Swarm Optimization (PSO) [139]. However, the original PSO algorithm is designed for solving problems with continuous solutions. For it to fit the problem, we first parameterized the particles.

Similar with LBO, BPSO uses the unassigned tasks set U as input and manages to find the assignment decision x_{jk}. In this paper, Quality Loss of Results (QLR) is proposed to quantify the near-optimal of the user acquired service quality (i.e, quality degradation). For each task k, q_k is defined as the level of QLR for each task k.

In BPSO, a swarm set P with |P| particles are generated. Each particle has a search space of 2× |K| dimension. As shown in Table 3-3, the dimensions $1 \times |K|$ demonstrate the decision of the task assignment. Each part of dimension has a discrete set of possible values limited to $\{1 \leq ass_k \leq |J|\}$. Therefore, the results of dimensions |K| can be transferred to task assignment set X, where $x_{jk} = 1$ when $ass_k = j$. The last part of dimensions from |K| + 1 to $2 \times |K|$ refers to the quality loss level selected for each task k, and each dimension has a discrete set of possible values limited to $\{1 \leq q_k \leq |K|\}$.

In PSO, the p^{th} particle is denoted by $X_p = (x_{p1}; x_{p2}; \ldots; x_{pD})$ and the best position it has experienced (with the best fitness value) is recorded as X_{pbest}. The index number of the best position experienced by all particles in the swarm is called G_{best}. The velocity of particle p is represented by $V_p = (v_{p1}; v_{p2}; \ldots; v_{pD})$. For each generation, its d dimension $(1 \leq d \leq D)$ updates according to the following equation:

Equation 3-9: $$V_{pd} = wV_{pd} + C_1 rand_1\left[X^p_{best} - X_{pd}\right] + C_2 rand_2\left[G_{best} - X_{pd}\right]$$

Equation 3-10: $X_{pd} = X_{pd} + X_{pd+1}$,

where w is the inertia weight, C_1 and C_2 are acceleration constants, and $rand_1$ and $rand_2$ are two random functions with range [0, 1]. In order to adapt to the problem, we customize the method in [140] and propose the BPSO algorithm. We first initialize the related parameters, which are global velocity (C_2), local velocity vector (C_1), inertia weight (w), according to [141] ($C_1 = C_2 = 1$; w = 0.9). Simultaneously, we randomly generate as many potential assignments for the problem as the size of the initial population |P|.

The algorithm keeps an updated version of two special variables throughout the course of its execution: global best position G_{best} and local best position X_{pbest}. It does that by conducting two ongoing comparisons. First, for each particle, compare its fitness value with the best position X_{pbest} it has experienced. If it is better, then it will be set as the current best position X_{pbest}. Second, for each particle, compare its fitness value with the best position G_{best} experienced globally. If it is better, reset G_{best}'s index number. These two positions affect the new velocity of every particle in the population according to Equation 3-9. As shown in this equation, two random parameters control the amount of effect the two positions (i.e., G_{best} and X_{pbest}) impose over the new particle velocity. The algorithm uses the new velocity to update the particle's current position with a new position according to Equation 3-10. Note that until now, the new positions of particles consist of continuous values, which are meaningless solutions for the task allocation problem. We round the values of all dimensions in each particle to the nearest integer and check whether the values fit all the binary constraints. Then, the algorithm evaluates the fitness of these particles according to their rounded positions. Once the current positions are better than the previous ones and all the constraints are satisfied, all particles will adjust their positions which constitutes the new status of the population. This process continues until the termination condition is satisfied (e.g., maximum iteration or getting the required result).

	Assign_1	...	$\text{Assign}_{\lvert\mathcal{K}\rvert}$	QLR_1	...	$\text{QLR}_{\lvert\mathcal{K}\rvert}$
Particle_1	ass_1	...	$ass_{\lvert\mathcal{K}\rvert}$	q_1	...	$q_{\lvert\mathcal{K}\rvert}$
...	...	...	...	...	...	...
Particle_p	ass_1	...	$ass_{\lvert\mathcal{K}\rvert}$	q_1	...	$q_{\lvert\mathcal{K}\rvert}$
...	...	...	...	...	...	...
$\text{Particle}_{\lvert P\rvert}$	ass_1	...	$ass_{\lvert\mathcal{K}\rvert}$	q_1	...	$q_{\lvert\mathcal{K}\rvert}$

Table 3-3: BPSO Particles

3.2.4.5 Conclusion

In this section, we highlighted Chameleon, a dynamic task allocation solution for multi-drone firefighting scenario. It is designed to minimize average service latency while reducing the overall quality loss. We considered the constraints on service latency, quality loss, and fog node capacity and formulated the task allocation process as a bi-objective optimization problem, where a trade-off is maintained between the service latency and quality loss. As it is proved to be an NP-hard problem, we proposed an event-triggered dynamic task allocation framework based on LBO and BPSO to solve the optimization problem. To be specific, using LBO, the problem first should be linearized, while BPSO solves the nonlinear discontinuity model directly.

4 Conclusions and Outlooks

This deliverable aims to show that the goal of the PriMO-5G project is attainable by providing a summary of the research innovations that are currently being developed in the project's application domain. We presented application-driven algorithms that cater to the different aspects of AI-assisted communications.

We divided the document into two major sections, namely AI-assisted networking and AI-assisted edge computing. We started with a brief overview of artificial intelligence and machine learning, in which different terminologies used in the succeeding parts of the document were introduced. For each of the two major major sections, we presented a literature survey that summarizes how machine learning is being used in networking and edge computing and then summarized some of the challenges that we aim to solve.

For the AI-assisted networking, we proposed various innovations regarding IoT networks, distributed machine learning, communication scheduling, caching, and UAV networking. We have shown fruitful results in multiple studies which we highlight as the following: (1) an efficient dynamic spectrum access for IoT networks through reinforcement learning, (2) a privacy-preserving method to share knowledge for machine learning modules in local devices, (3) a communicative multi-agent reinforcement learning algorithm that caters to multiple network limitations, (4) a simulator for UAV networking that also doubles as a reinforcement learning environment, and (5) an analysis of the routing stretch of large-scale cache networks.

For the AI-assisted edge computing, we introduced a wide array of developments in network slicing, MECs, and drone applications. In particular, we have produced the following: (1) a machine learning-based mobile backhaul orchestrator that creates and manages network slices, (2) a depth-controllable super-resolution network for scaling processing speed and image quality, (3) a deep learning based auction design applied to drone charging scheduling, and (4) a quality-aware transmission policy to address the latency sensitivity of tasks such as object recognition.

Through these innovative solutions, we have shown that AI-assisted communication is feasible and within our grasps. For example, we can integrate some of the presented studies in a firefighting scenario. Using the UAV network simulator and the MARL algorithm presented in Chapter 2, we can easily train drones to cooperate, given the possibility of limited networking capabilities. In addition, we can employ the video transmission methods, as well as the drone charging scheduling from Chapter 3 to make the drones more efficient. Although these developments were tested separately and are currently detached from one another, we foresee that renditions of these advancements will be integrated to create an efficient and reliable network that can handle the evolving needs of different devices in the upcoming 5G system.

5 References

[1] Usama Muhammad, Qadir Junaid, Raza Aunn, Arif Hunain, Yau Kok-Lim, Elkhatib Yehia, Hussain Amir, Al-Fuqaha Ala. Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, May 2015.

[3] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano and O. M. Caicedo, “A comprehensive survey on machine learning for networking: evolution, applications and research opportunities,” Journal of Internet Services and Applications (2018) 9:16.

[4] S.C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, “Qos-aware adaptive routing in multi-layer hierarchical software defined networks: a reinforcement learning approach,” Services Computing (SCC). IEEE International Conference on. IEEE; 2016. p. 25–33.

[5] Bhorkar AA, Naghshvar M, Javidi T, Rao BD. Adaptive opportunistic routing for wireless ad hoc networks. EEE/ACM Trans Netw. 2012;20(1): 243–56.

[6] Li, Wei & Zhou, Fan & Roy Chowdhury, Kaushik & Meleis, Waleed. QTCP: Adaptive Congestion Control with Reinforcement Learning. IEEE Transactions on Network Science and Engineering 2018 PP. 1-1. 10.1109/TNSE.2018.2835758.

[7] Jay, Nathan & Rotman, Noga & Brighten Godfrey, P & Schapira, Michael & Tamar, Aviv. (2018). Internet Congestion Control via Deep Reinforcement Learning. https://arxiv.org/abs/1810.03259.

[8] Cortez P, Rio M, Rocha M, Sousa P. Internet traffic forecasting using neural networks. Proceedings of IEEE International Joint Conference on Neural Networks (IJCNN). IEEE; 2006. p. 2635–42.

[9] Samira, Chabaa & Zeroual, Abdelouhab & Antari, Jilali. (2010). Identification and Prediction of Internet Traffic Using Artificial Neural Networks. JILSA. 2. 147-155. 10.4236/jilsa.2010.23018.

[10] Chen Z, Wen J, Geng Y. Predicting future traffic using hidden markov models. Proceedings of 24th IEEE International Conference on Network Protocols (ICNP). IEEE; 2016. p. 1–6.

[11] Li Y, Liu H, Yang W, Hu D, Xu W. Inter-data-center network traffic prediction with elephant flows. IEEE; 2016b, pp. 206–13.

[12] Poupart P, Chen Z, Jaini P, Fung F, Susanto H, Geng Y, Chen L, Chen K, Jin H. Online flow size prediction for improved network routing. IEEE; 2016, pp. 1–6.

[13] Jin Y, Duffield N, Erman J, Haffner P, Sen S, Zhang ZL. A modular machine learning system for flow-level traffic classification in large networks. ACM Trans Knowl Discov Data (TKDD). 2012;6(1):4.

[14] Shbair WM, Cholez T, Francois J, Chrisment. A multi-level framework to identify https services. IEEE/IFIP Network Operations and Management Symposium (NOMS) 2016. p. 240–8.

[15] Chen, Li & Lingys, Justinas & Chen, Kai & Liu, Feng. “AuTO: Scaling Deep Reinforcement Learning for Datacenter-Scale Automatic Traffic Optimization”, ACM SIGCOMM, 191-205, 2018.

[16] Lu X, Wang H, Zhou R, Ge B. Using hessian locally linear embedding for autonomic failure prediction. Nature & Biologically Inspired, Computing, 2009. NaBIC 2009. World Congress on. IEEE; 2009. p. 772–6.

[17] Kumar Y, Farooq H, Imran A. Fault prediction and reliability analysis in a real cellular network. Wireless Communications and Mobile Computing Conference (IWCMC), 2017 13th International. IEEE; 2017. p. 1090–1095.

[18] Johnsson A, Meirosu C. Towards automatic network fault localization in real time using probabilistic inference. Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on. Piscataway IEEE; 2013. p. 1393–8.

[19] Moustapha AI, Selmic RR. Wireless sensor network modeling using modified recurrent neural networks: Application to fault detection. IEEE Trans Instrum Meas. 2008;57(5):981–8.

[20] M. Simsek, M. Bennis, and I. Guvenc, Learning based frequency and time-domain inter-cell interference coordination in HetNets. IEEE Trans. Veh. Technol., vol. 64, no. 10, pp. 4589–4602, Oct. 2015.

[21] A. Galindo-Serrano and L. Giupponi. Distributed Q-learning for aggregated interference control in cognitive radio networks IEEE Trans. Veh. Technol., vol. 59, no. 4, pp. 1823–1834, May 2010.

[22] W. Lee, M. Kim, and D. Cho. Deep power control: Transmit power control scheme based on convolutional neural network. IEEE Commun. Lett., vol. 22, no. 6, pp. 1276–1279, Apr. 2018.

[23] C. Fan, B. Li, C. Zhao, W. Guo, and Y. C. Liang. Learning-based spectrum sharing and spatial reuse in mm-wave ultra-dense networks. IEEE Trans. Veh. Technol., vol. 67, no. 6, pp. 4954–4968, Jun. 2018.

[24] M. Chen, W. Saad, and C. Yin. Echo state networks for self-organizing resource allocation in LTE-U with uplink-downlink decoupling. IEEE Trans. Wireless Commun., vol. 16, no. 1, pp. 3–16, Jan. 2017.

[25] G. Cao, Z. Lu, X.Wen, T. Lei, and Z. Hu. AIF: An artificial intelligence framework for smart wireless network management. IEEE Commun. Lett., vol. 22, no. 2, pp. 400–403, Feb. 2018.

[26] Z. Wang, Y. Xu, L. Li, H. Tian, and S. Cui. Handover control in wireless systems via asynchronous multi-user deep reinforcement learning arXiv:1801.02077v2, May 2018, accessed on Jun. 30, 2018.

[27] M. Jaber, M. Imran, R. Tafazolli, and A. Tukmanov. An adaptive backhaul-aware cell range extension approach. Proceedings of ICCW, London, UK, Jun. 2015, pp. 74–79.

[28] G. Alnwaimi, S. Vahid, and K. Moessner, “Dynamic heterogeneous learning games for opportunistic access in lte-based macro/femtocell deployments,” IEEE Transactions on Wireless Communications, vol. 14, no. 4, pp. 2294–2308, Apr. 2015.

[29] F. Bernardo, R. Agust, J. Perez-Romero, and O. Sallent, “An application of reinforcement learning for efficient spectrum usage in next-generation mobile cellular networks,” IEEE Transactions on Systems, Man, and Cybernetics-PART C: Applications and Reviews, vol. 40, no. 4, pp. 477–484, Jul. 2010.

[30] O. Onireti, A. Zoha, J. Moysen, A. Imran, L. Giupponi, M. A. Imran, and A. Abu-Dayya, “A cell outage management framework for dense heterogeneous networks,” IEEE Transactions on Vehicular Technology, vol. 65, no. 4, pp. 2097–2113, Apr. 2016.

[31] S. Maghsudi and S. Stanczak, “Channel selection for network-assisted d2d communication via no-regret bandit learning with calibrated forecasting,” IEEE Transactions on Wireless Communications, vol. 14, no. 3, pp. 1309–1322, Mar. 2015.

[32] R. Bonnefoi, L. Besson, C. Moy, E. Kaufmann, and J. Palicot, “Multiarmed bandit learning in iot networks: learning helps even in nonstationary settings,” in Proc. 12th EAI International Conference on Cognitive Radio Oriented Wireless Networks (CROWNCOM) 2017, Lisbon, Portugal, pp. 173–185, Feb. 2018.

[33] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, “Deep reinforcement learning for dynamic multichannel access in wireless networks,” IEEE Transactions on Cognitive Communications and Networking, vol. 4, no. 2, pp. 257–264, Jun. 2018.

[34] V. Raj, I. Dias, T. Tholeti, and S. Kalyani, “Spectrum access in cognitive radio using a two stage reinforcement learning approach,” IEEE Journal of Selected Topics in Signal Processing, vol. 12, no. 1, pp. 20–34, Jan. 2018.

[35] Simsek Meryem, Bennis Mehdi, Guvenc Ismail. (2015). Mobility management in HetNets: a learning-based perspective. EURASIP Journal on Wireless Communications and Networking. 2015. 10.1186/s13638-015-0244-2.

[36] Zhang Chaoyun, Patras Paul, Haddadi Hamed. (2018). Deep Learning in Mobile and Wireless Networking: A Survey. IEEE Communications Surveys & Tutorials. PP. 10.1109/COMST.2019.2904897, 2018.

[37] Chen, Mingzhe, Challita, Ursula, Saad, Walid, Yin, Changchuan, Debbah, Mérouane. (2017). Machine Learning for Wireless Networks with Artificial Intelligence: A Tutorial on Neural Networks. https://arxiv.org/abs/1710.02913.

[38] S. Ravi, “On-device machine intelligence” Google AI Blog. Feb. 2017 [Online, Accessed: 2019-03-17]. URL: https://ai.googleblog.com/2017/02/on-device-machine-intelligence.html.

[39] M.R. Sprague, A. Jalalirad, M. Scavuzzo, C. Capota, M. Neun, L. Do, and M. Kopp, “Asynchronous federated learning for geospatial applications.” In Proc. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases and Knowledge Discovery in Databases Workshops (ECML PKDD)*, Dublin, Ireland, 2018, pp. 21-28.

[40] J. Konecný, H. B. McMagan, and D. Ramage, “Federated optimization: distributed machine learning for on-device intelligence” [Online]. ArXiv preprint: https://arxiv.org/abs/1610.02527.

[41] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtarik, A. T. Suresh, and D. Bacon. “Federated learning: Strategies for improving communication efficiency.” In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[42] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen, and M. Chen, “In-Edge AI: Intelligentizing Mobile Edge Computing, Caching and Communication by Federated Learning”, 2018. Available: https://arxiv.org/abs/1809.07857.

[43] L. Cao, "Non-IIDness Learning in Behavioral and Social Data", The Computer Journal, vol. 57, no. 9, pp. 1358-1370, 2014. Available: 10.1093/comjnl/bxt084.

[44] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra. “Federated learning with non-IID data” [Online]. arXiv preprint: https://arxiv.org/abs/1806.00582.

[45] F. Sattler, S. Wiedemann, K. R. Müller, and W. Samek, “Robust and communication-efficient federated learning from non-IID data,” 2019. arXiv preprint: https://arxiv.org/abs/1903.02891.

[46] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network”, 2015. Available: https://arxiv.org/abs/1503.02531.

[47] Y.Zhang, T.Xiang, T.M.Hospedales, and H.Lu. “Deep mutual learning”, 2017. Available: https://arXiv.org/abs/1706.00384.

[48] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, “Learning to communicate with deep multi-agent reinforcement learning,” In Proceedings of Advances in Neural Information Processing Systems, pp. 2137–2145, 2016.

[49] S. Sukhbaatar, R. Fergus, et al., “Learning multiagent communication with backpropagation,” In Proceedings of Advances in Neural Information Processing Systems, pp. 2244–2252, 2016.

[50] P. Peng, Q. Yuan, Y. Wen, Y. Yang, Z. Tang, H. Long, and J. Wang, “Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games,” arxiv preprint arXiv:1703.10069, 2017.

[51] C. Guestrin, M. Lagoudakis, and R. Parr, “Coordinated reinforcement learning,” In Proceedings of International Conference on Machine Learning, 2002.

[52] C. Zhang and V. Lesser, “Coordinating multi-agent reinforcement learning with limited communication,” In Proceedings of International conference on Autonomous agents and multiagent systems, 2013.

[53] I. Mordatch and P. Abbeel, “Emergence of grounded compositional language in multi-agent populations,” arxiv preprint arXiv:1703.10069, 2017.

[54] S. Havrylov and I. Titov, “Emergence of language with multi-agent games: learning to communicate with sequences of symbols,” In Proceedings of Advances in Neural Information Processing Systems, pp. 2146–2156, 2017.

[55] J. Jiang and Z. Lu, “Learning attentional communication for multi-agent cooperation,” arXiv preprint arXiv:1805.07733, 2018.

[56] J. Jiang, C. Dun, and Z. Lu, “Graph Convolutional Reinforcement Learning for Multi-agent Cooperation,” arXiv preprint arXiv:1910.09202, 2019.

[57] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, Learning monocular reactive UAV control in cluttered natural environments. IEEE Int. Conf. Robot. Autom. (ICRA), 2013, pp. 1765–1772.

[58] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[59] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, Trust region policy optimization. Proceedings of the 32nd International Conference on Machine Learning (ICML-15), 2015, pp. 1889–1897.

[60] M. Mancini, G. Costante, P. Valigi, T. A. Ciarfuglia, J. Delmerico, and D. Scaramuzza, “Towards domain independence for learning-based monocular depth estimation,” IEEE Robot. Autom. Lett., 2017.

[61] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, “Target-driven visual navigation in indoor scenes using deep reinforcement learning,” IEEE International Conference on Robotics and Automation (ICRA), may 2017.

[62] Fereshteh Sadeghi, Sergey Levine. CAD^2 RL: Real Single-Image Flight Without a Single Real Image Robotics: Science and Systems XIII, July 2017.

[63] M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, D. Scaramuzza, L. M. Gambardella. A machine learning approach to visual perception of forest trails for mobile robots. IEEE Robotics and Automation Letters, 1(2): 661–667, July 2016.

[64] Smolyanskiy, Nikolai & Kamenev, Alexey & Smith, Jeffrey & Birchfield, Stan. (2017). Toward Low-Flying Autonomous MAV Trail Navigation using Deep Neural Networks for Environmental Awareness. https://arxiv.org/abs/1705.02550.

[65] Loquercio, Antonio & Maqueda, Ana & R. Del Blanco, Carlos & Scaramuzza, Davide. (2018). DroNet: Learning to Fly by Driving. IEEE Robotics and Automation Letters. PP. 1-1. 10.1109/LRA.2018.2795643.

[66] Udacity, “An Open Source Self-Driving Car,” https://www.udacity.com/self-driving-car, 2016.

[67] Mingzhe Chen, Walid Saad, Mérouane Debbah. Caching in the Sky: Proactive Deployment of Cache-Enabled Unmanned Aerial Vehicles for Optimized Quality-of-Experience. IEEE Journal On Selected Areas in Communications, Vol. 35, No. 5, May 2017.

[68] T. Rappaport. Wireless Communications: Principles and Practice. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition, 2001. ISBN 0130422320.

[69] L. Tassiulas and A. Ephremides, “Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks,” in IEEE transactions on automatic control, 37(12):1936–1948, 1992.

[70] Y. Yi, A. Proutiere, and M. Chiang, “Complexity in wireless scheduling: Impact and tradeoffs,” in Proceedings of ACM Mobihoc, 2008.

[71] L. Jiang and J. Walrand, “A distributed CSMA algorithm for throughput and utility maximization in wireless networks,” in IEEE/ACM Transactions on Networking (ToN), 18(3):960–972, 2010.

[72] J. F. Kurose. Computer networking: A top-down approach featuring the internet. Pearson Education India, 2005.

[73] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, “Multiagent cooperation and competition with deep reinforcement learning,” PloS one, 12(4):e0172395, 2017.

[74] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, “Counterfactual multi-agent policy gradients,” in Proceedings of AAAI, 2018.

[75] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “Airsim: High-fidelity visual and physical simulation for autonomous vehicles,” in Field and service robotics, pages 621–635. Springer, 2018.

[76] Omnet++. https://omnetpp.org/.

[77] INET framework. https://inet.omnetpp.org/.

[78] Messagepack. https://msgpack.org/.

[79] QGroundControl. http://qgroundcontrol.com/.

[80] PyQt. https://riverbankcomputing.com/software/pyqt/intro.

[81] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” arXiv preprint arXiv:1606.01540, 2016.

[82] V. Mnih, K. Kavukcuoglu, D. Silver, A. A Rusu, J. Veness, M. G Bellemare, A. Graves, M. Riedmiller, A. K Fidjeland, G. Ostrovski, et al, “Human-level control through deep reinforcement learning,” in Nature, 518(7540):529, 2015.

[83] M. Tan, “Multi-agent reinforcement learning: Independent vs. cooperative agents,” in Proceedings of the tenth international conference on machine learning, pages 330–337, 1993.

[84] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, “Value-decomposition networks for cooperative multi-agent learning,” arXiv preprint arXiv:1706.05296, 2017.

[85] Fricker, C., Robert, P., Roberts, J.: A versatile and accurate approximation for LRU cache performance. In: Proc. ITC (2012).

[86] Garetto, M., Leonardi, E., Martina, V.: A unified approach to the performance analysis of caching systems. ACM TOMPECS 1(3), 12 (2016).

[87] Jelenkovic´, P.: Asymptotic approximation of the move-to-front search cost distribution and least-recently-used caching fault probabilities. The Annals of Applied Probability 9(2), 430– 464 (1999).

[88] Sikdar, S., Chaudhary, A., Kumar, S., Ganguly, N., Chakraborty, A., Kumar, G., Patil, A.,Mukherjee, A.: Identifying and characterizing sleeping beauties on youtube. In: Proc. ACM CSCW (2016).

[89] N. Abbas, Y. Zhang, A. Taherkordi and T. Skeie, "Mobile Edge Computing: A Survey," in IEEE Internet of Things Journal, vol. 5, no. 1, pp. 450-465, Feb. 2018.

[90] F. D. Vita, D. Bruneo, A. Puliafito, G. Nardini, A. Virdis and G. Stea, "A Deep Reinforcement Learning Approach For Data Migration in Multi-Access Edge Computing," 2018 ITU Kaleidoscope: Machine Learning for a 5G Future (ITU K), Santa Fe, 2018, pp. 1-8.

[91] M. Abderrahim, A. Ben Letaifa, A. Haji and S. Tabbane, "How to use MEC and ML to Improve Resources Allocation in SDN Networks ?," 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA), Krakow, 2018, pp. 442-447.

[92] S. Yu, X. Wang and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, 2017, pp. 1-6.

[93] T. Hou, G. Feng, S. Qin and W. Jiang, "Proactive Content Caching by Exploiting Transfer Learning for Mobile Edge Computing," GLOBECOM 2017 - 2017 IEEE Global Communications Conference, Singapore, 2017, pp. 1-6.

[94] Chanh Nguyen Le Tan, C. Klein and E. Elmroth, "Location-aware load prediction in Edge Data Centers," 2017 Second International Conference on Fog and Mobile Edge Computing (FMEC), Valencia, 2017, pp. 25-31.

[95] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang and X. S. Shen, "Content Popularity Prediction Towards Location-Aware Mobile Edge Caching," in IEEE Transactions on Multimedia.

[96] T. He, E. N. Ciftcioglu, S. Wang and K. S. Chan, "Location Privacy in Mobile Edge Clouds," 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), Atlanta, GA, 2017, pp. 2264-2269.

[97] P. Yang, N. Zhang, S. Zhang, L. Yu, J. Zhang and X. Shen, "Dynamic Mobile Edge Caching with Location Differentiation," GLOBECOM 2017 - 2017 IEEE Global Communications Conference, Singapore, 2017, pp. 1-6.

[98] E. Pencheva, I. Atanasov, K. Kassev and V. Trifonov, "Location service in mobile edge computing," 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), Milan, 2017, pp. 617-622.

[99] Luong, Nguyen Cong et al. “Applications of Deep Reinforcement Learning in Communications and Networking: A Survey.” *CoRR* abs/1810.07862 (2018).

[100] Wanrong Huang, Yanzhen Wang, and Xiaodong Yi. 2017. Deep Q-Learning to Preserve Connectivity in Multi-robot Systems. In Proceedings of the 9th International Conference on Signal Processing Systems (ICSPS 2017). ACM, New York, NY, USA, 45-50.

[101] W. Huang, Y. Wang and X. Yi, "A deep reinforcement learning approach to preserve connectivity for multi-robot systems," 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Shanghai, 2017, pp. 1-7.

[102] K. Gilly, S. Filiposka, A. Mishev, "Supporting Location Transparent Services in a Mobile Edge Computing Environment," Advances in Electrical and Computer Engineering, vol.18, no.4, pp.11-22, 2018.

[103] C. Dong, C. C. Loy, K. He, and X. Tang, “Image super resolution using deep convolutional networks,” IEEE Trans. Pattern Anal. Mach. Intell, vol. 38, pp.295–307, 2016.

[104] J. Kim, J. Lee, and K. Lee, “Accurate Image Super-Resolution Using Very Deep Convolutional Networks,” IEEE Conf. Computer Vision and Pattern Recognition, pp.1646-1654, 2016

[105] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.770–778, 2016.

[106] J. Kim, J. Lee, and K. Lee, “Deeply-recursive convolutional network for image super-resolution,” IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.1637–1645, 2016.

[107] Y. Tai, J. Yang and X. Liu, “Image super-resolution via deep recursive residual network,” IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.2790–2798, 2017.

[108] T. Tong, G. Li, X. Liu, and Q. Gao, “Image super-resolution using dense skip connections,” IEEE Int’l Conf. Computer Vision (ICCV), pp.4809–4817, 2017.

[109] Y. Zhang, Y. Tian, Y. Kong, B. Zhong, and Y. Fu, “Residual dense network for image super-resolution,” IEEE Conf. Computer Vision and Pattern Recognition (CVPR), June 2018.

[110] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S.Guadarrama, and K. Murphy, “Speed/accuracy trade-offs for modern convolutional object detectors,” IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.3296–3297, 2017.

[111] R. Timofte, E. Agustsson, L. V. Gool, M. H. Yang, L. Zhang, et al., “Ntire 2017 challenge on single image super-resolution: Methods and results,” IEEE Conf. Computer Vision and Pattern Recognition (CVPR) Worshops, pp.1110–1121, 2017.

[112] S. Park, L. Zhang, and S. Chakraborty, “Battery assignment and scheduling for drone delivery businesses,” in Proc. IEEE/ACM ISLPED, 2017.

[113] A. Couture-Beil and R. T. Vaughan, “Adaptive mobile charging stations for multi-robot systems,” in Proc. IEEE IROS, 2009.

[114] M. P. Wellman, W. E. Walsh, P. R. Wurman, and J. K. MacKie-Mason, “Auction protocols for decentralized scheduling,” Games and Economic Behavior, vol. 35, 2001.

[115] P. Corcoran and S. K. Datta, “Mobile-Edge Computing and the Internet of Things for Consumers: Extending cloud computing and services to the edge of the network,” *IEEE Consum. Electron. Mag.*, vol. 5, no. 4, pp. 73–74, Oct. 2016.

[116] M. Satyanarayanan, “Edge computing for situational awareness,” in *2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, 2017, pp. 1–6.

[117] Y. Xiao and C. Zhu, “Vehicular fog computing: Vision and challenges,” in *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, 2017, pp. 6–9.

[118] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, “Vehicular Fog Computing: A Viewpoint of Vehicles as the Infrastructures,” *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.

[119] J. Ni, A. Zhang, X. Lin, and X. S. Shen, “Security, Privacy, and Fairness in Fog-Based Vehicular Crowdsensing,” *IEEE Commun. Mag.*, vol. 55, no. 6, pp. 146–152, 2017.

[120] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, “Resource Provisioning for IoT Services in the Fog,” in *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, 2016, pp. 32–39.

[121] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, “Zenith: Utility-Aware Resource Allocation for Edge Computing,” in *2017 IEEE International Conference on Edge Computing (EDGE)*, 2017, pp. 47–54.

[122] J. Du, L. Zhao, J. Feng, and X. Chu, “Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems With Min-Max Fairness Guarantee,” *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1594–1608, Apr. 2018.

[123] Y. Li, Y. Chen, T. Lan, and G. Venkataramani, “MobiQoR: Pushing the Envelope of Mobile Edge Computing Via Quality-of-Result Optimization,” in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 1261–1270.

[124] S. Sardellitti, G. Scutari, and S. Barbarossa, “Joint Optimization of Radio and Computational Resources for Multicell Mobile-Edge Computing,” *IEEE Trans. Signal Inf. Process. Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[125] Y. Liu, M. J. Lee, and Y. Zheng, “Adaptive Multi-Resource Allocation for Cloudlet-Based Mobile Cloud Computing System,” *IEEE Trans. Mob. Comput.*, vol. 15, no. 10, pp. 2398–2410, Oct. 2016.

[126] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. S. Quek, “Offloading in Mobile Edge Computing: Task Allocation and Computational Frequency Scaling,” *IEEE Trans. Commun.*, vol. 65, no. 8, pp. 3571–3584, Aug. 2017.

[127] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, “Optimal Workload Allocation in Fog-Cloud Computing Toward Balanced Delay and Power Consumption,” *IEEE Internet Things J.*, vol. 3, no. 6, pp. 1171–1181, Dec. 2016.

[128] J. Feng, Z. Liu, C. Wu, and Y. Ji, “AVE: Autonomous Vehicular Edge Computing Framework with ACO-Based Scheduling,” *IEEE Trans. Veh. Technol.*, vol. PP, no. 99, pp. 1–1, 2017.

[129] Mininet simulator. Available at: http://mininet.org/.

[130] Open Virtual Switch. Available at: https://www.openvswitch.org/.

[131] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,”IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pp.2818–2826, 2016.

[132] L. Cavigelli, P. Hager, and L. Benini, “Cas-cnn: A deep convolutional neural network for image compression artifact suppression,” IEEE Int’l Joint Conf. Neural Networks(IJCNN), pp. 752–759, 2017.

[133] Y. Hu, X. Gao, J. Li, Y. Huang, and H. Wang, “Single image super-resolution via cascaded multi-scale cross network,” Computing Research Repository (CoRR), vol.abs/1802.08808, 2018.

[134] S. Li *et al.*, “Joint Admission Control and Resource Allocation in Edge Computing for Internet of Things,” *IEEE Netw.*, vol. 32, no. 1, pp. 72–79, Jan. 2018.

[135] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeaeski, “Fog Following Me: Latency and Quality Balanced Task Allocation in Vehicular Fog Computing,” in *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, 2018, pp. 1–9.

[136] W. Hu, Z. Feng, Z. Chen, J. Harkes, P. Pillai, and M. Satyanarayanan, “Live Synthesis of Vehicle-Sourced Data Over 4G LTE,” 2017.

[137] “Small Cell Network White Paper.”

[138] J. Redmon and A. Farhadi, “YOLO9000: Better, Faster, Stronger,” *ArXiv161208242 Cs*, Dec. 2016.

[139] F. Hembecker, H. S. Lopes, and W. Godoy, “Particle Swarm Optimization for the Multidimensional Knapsack Problem,” in *Adaptive and Natural Computing Algorithms*, 2007, pp. 358–365.

[140] J. Kennedy, “Particle Swarm Optimization,” in *Encyclopedia of Machine Learning*, Springer, Boston, MA, 2011, pp. 760–766.

[141] A. Salman, I. Ahmad, and S. Al-Madani, “Particle swarm optimization for task assignment problem,” *Microprocess. Microsyst.*, vol. 26, no. 8, pp. 363–371, Nov. 2002.

[125] H. Cha, and S.-L. Kim, "A reinforcement learning approach to dynamic spectrum access in Internet-of-Things networks," will be presented in 2019 IEEE International Conference on Communications (ICC 2019), Shanghai, China 2019.

[126] V. V. Phansalkar and M. A. L. Thathachar, “Local and global optimization algorithms for generalized learning automata,” Neural Computer, vol. 7, no. 5, pp. 950–973, Sep. 1995.

[127] E. Jeong, S. Oh, H. Kim, J. Park, M. Bennis, and S. -L. Kim, "Communication-efficient on-device machine learning: federated distillation and augmentation under non-IID private data," in Proc. NIPS Workshop on Machine Learning on the Phone and other Consumer Devices (MLPCD), Montréal, Canada, December 2018. Available at: https://arxiv.org/abs/1811.11479.