UNIVERSITY OF CATANIA

DEPARTMENT OF ELECTRICAL, ELECTRONIC AND COMPUTER ENGINEERING

PhD IN SYSTEM, ENERGY, COMPUTER AND TELECOMMUNICATIONS ENGINEERING

XXXV CYCLE

On the exploitation of ML in Infrastructureless networks

JOANNES SAM MERTENS JOSEPH THATHEYUS

Coordinator: Prof. Paolo Pietro Arena

Tutor: Prof. Laura Galluccio

ACKNOWLEDGEMENTS

With immense pleasure and deep sense of gratitude, I wish to express my sincere thanks to my supervisor **Dr. Laura Galluccio**, without her motivation and continuous encouragement, this research would not have been successfully completed.

I express my sincere thanks to **Dr. Giacomo Morabitto** for motivating me to carry out research in the University of Catania and also for providing me with infrastructural facilities and many other resources needed for my research.

I wish to extend my profound sense of gratitude to **my parents** for all the sacrifices they made during my research and also providing me with moral support and encouragement whenever required.

Place: Catania

Date: 24/11/2022 **JOANNES SAM MERTENS JOSEPH THATHEYUS**

Contents

ACKNOWLEDGEMENT i

List of Terms and Abbreviations **v**

List of Figures **vi**

List of Tables **ix**

Abstract **1**

1 Introduction **2**

2 Wireless Sensor Networks **6**

2.1 Introduction 6

2.2 Federated Learning (FL) 6

2.2.1 Federated learning in WSN 10

2.2.2 Gossiping for FL 12

2.3 MGM-4-FL: Combining Federated Learning (FL) and Model Gossiping 13

2.3.1 Objectives and rationale 13

2.3.2 MGM-4-FL Protocol Details 16

2.3.3 Experimentation Scenario 19

2.3.4 Numerical results 23

2.3.5 MGM-4-FL with GANs 26

2.4 CGL: Centrality aware Gossiping 27

2.4.1 Centrality in WSN 29

2.4.2 CGL Design objectives 32

2.4.3 CGL Protocol Details 33

2.4.4 ML Algorithms 35

2.4.5 CGL: Numerical Results 37

2.5 i-WSN: Clustered Distributed learning 43

2.5.1 Overview of operations and characteristics of the hardware platforms . . . 45
2.5.2 i-WSN Protocol Details . . . 46
2.5.3 i-WSN Performance evaluation . . . 49
2.5.4 i-WSN Scenario . . . 51
2.5.5 i-WSN Numerical Results . . . 53
2.6 Applying SDN to WSN with Data Mules . . . 58
2.6.1 SDN-(UAV)ISE Scenario and Overview . . . 60
2.6.2 New Features of SDN-(UAV)ISE . . . 61
2.6.3 SDN-(UAV)ISE in action . . . 64
2.6.4 SDN-(UAV)ISE Performance evaluation . . . 65
2.6.5 SDN-(UAV)ISE Numerical results . . . 68

3 Vehicular Networks 71
3.1 Introduction . . . 71
3.2 ML for Vehicular Safety Applications . . . 71
3.2.1 ML for driving behaviour analysis . . . 72
3.2.2 ML for Autonomous Vehicles . . . 73
3.3 A Case Study: Detection of anomalies in cycling behavior with CNN 75
3.3.1 Bicycle Dataset . . . 76
3.3.2 Transfer Learning with CNN based Auto-encoders . . . 77
3.3.3 Numerical Results . . . 78
3.4 Achieving Layer Separation by Sequential training . . . 82
3.4.1 V2I Scenario . . . 83
3.4.2 Sequential Training Algorithm . . . 84
3.4.3 Numerical Results . . . 86

4 Internet of Underwater Things 92
4.1 Introduction to Underwater Test-beds . . . 92
4.2 System Description . . . 94
4.3 Experimental Setup: Underwater Sub-System . . . 96
4.4 Experimental Setup: LoRa Sub-System . . . 101
4.5 Android application . . . 104
4.6 Web application . . . 108
4.7 Numerical analysis . . . 110
4.7.1 Underwater sub-system . . . 110
4.7.2 LoRa Sub-System . . . 112
4.8 ML in Internet of Underwater Things . . . 116

5 Conclusion **118**

List of publications **120**

References **122**

List of Terms and Abbreviations

ACF	Avggregate Channel Feature
ADAS	Avanced Driver-Assistance System
AI	Artificial Intelligence
CCTV	Closed Circuit Telivision
CAE	Convolutional Auto Encoder
CH	Cluster Head
CNN	Convolutional Neural Network
CM	Cluster Member
DTC	Diagnostic Trouble Codes
FL	Federated Learning
GAN	Generative Adversarial Network
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IoT	Internet of Things
IoWT	Internet of Underwater Things
LIDAR	Light Detection and Ranging
LSTM	Long Short Term Memory
KNN	K Nearest Neighbor
ML	Machine Learning

MSE	Mean Square Error
NN	Neural Network
OBD	On Board Diagnostics
OBU	On Board Unit
OS-ELM	Online Sequential Extreme Learning Machine
PID	Parametric Identifiers
PM	Particulate Matter
RNN	Recurrent Neural Network
RPM	Revolutions Per Minute
RSU	Road Side Unit
SDN	Software Defined Networking
SGD	Stochastic Gradient Descent
SSADNET	imultaneous Segmentation and Detection Network
TEE	Trusted Execution Environments
UAV	Unmanned Aerial Vehicle
V2I	Vehicle to Infrastructure
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network

List of Figures

1.1 Structure of this thesis 5

2.1 Federated Learning mechanism. 8
2.2 Wireless network scenario 11
2.3 MGM-4-FL protocol in action 19
2.4 Wireless network scenario for the simulation. 20
2.5 Illustration of the WSN scenario in the form of Tree Topology for standard FL 20
2.6 Training Data Characterization 21
2.7 Model of an autoencoder. 22
2.8 Average Loss vs. No. of Iterations for different values of epochs (MGM-4-FL) 24
2.9 Average Loss vs. No. of Iterations for different values of α (MGM-4-FL) 24
2.10 Standard deviation of the Loss vs. No. of Iterations (MGM-4-FL) 25
2.11 Number of times each node has broadcast its model parameters . 25
2.12 Average Loss vs. the Number of Packets Transmitted 27
2.13 Generative Adversarial Networks 28
2.14 Average Loss vs Iterations 28
2.15 CGL protocol in action 36
2.16 Functioning of LSTM network 37
2.17 Average Loss vs Number of iterations for different values of α . . 39
2.18 Average Loss vs Number of iterations for different values of Epochs 40
2.19 Average Loss vs. Data Transmitted for different centrality measures (Autoencoders) 41
2.20 Average Loss vs. Data Transmitted for different centrality measures (LSTM network) 41
2.21 Average Loss vs. Data Transmitted (Autoencoders) 42
2.22 Average Loss vs. Data Transmitted (LSTM network) 42

2.23 Number of times each node has been trained (Case 1: Autoencoder) 43
2.24 Number of times each node has been trained (Case 2: LSTM-Prediction) 43
2.25 Clustering Scenario. 47
2.26 i-WSN League protocol in action (Cluster Heads) 51
2.27 i-WSN League protocol in action (inside Cluster 4) 51
2.28 Clustering cases considered in the experiments 52
2.29 Average Loss vs Number of Iteration for different values of epochs 53
2.30 Average Loss vs Number of Iteration for different values of α . . . 54
2.31 Average Loss vs Number of Iterations in each case. 55
2.32 Standard Deviation Loss vs Number of Iterations in each case. . . 55
2.33 Average Train count vs Number of Epochs 56
2.34 Average Data chunk transmissions vs Number of Epochs 56
2.35 Average number of times each Cluster Head has been trained . . . 57
2.36 Average number of times each Cluster Member has transmitted data chunks 57
2.37 Proposed scenario. 60
2.38 Sensors protocol stack. 61
2.39 Controller Application Layer block diagram. 62
2.40 Standard deviation of the Loss vs. No. of Iterations (MGM-4-FL) 66
2.41 Variation of criticality when drone moves randomly and for different values of α 69
2.42 Expected Utility with respect to the duration of the forecast window. 70

3.1 Bicycle Path 76
3.2 Training Loss vs Epochs 79
3.3 Confusion matrix 80
3.4 Recall vs Threshold 81
3.5 Precision vs Threshold 81
3.6 F-score vs Threshold 82
3.7 V2I scenario 84
3.8 Neural network architecture. 85
3.9 Start-up Training 87
3.10 Average Loss vs Number of iterations for different values of Epochs 89
3.11 Average Loss vs Input-Time Stamp 90
3.12 Average Loss vs Number of Samples 91

4.1 The IoUT System 94

4.2 Functional View of the System 94
4.3 The Underwater sub-system components. 97
4.4 The Underwater components during marine tests. 98
4.5 LoRa End Node 105
4.6 LoRa Gateway 105
4.7 Setting camera angle using the Android application. 108
4.8 Dashboard of the web application 109
4.9 Catania Port setting 111
4.10 Transmission Time (s) Vs. Image Quality-Aquarium setting 113
4.11 Transmission Time (s) Vs. Image Quality (WEBP)- Catania Port setting 113
4.12 Transmission Time (s) Vs. Image Quality (JPEG)- Catania Port setting 114
4.13 Sample image collected during the marine tests. 114
4.14 Packet Delivery Rate (PDR) as a function of the SF being used. . 115

List of Tables

2.1 Summary of relevant papers. 14
2.2 Fairness Index for the number of training events and the number of times a node is chosen as best node for different values of α and epochs. 29
2.3 Fairness Index for regarding the number of times Cluster Heads have been trained for different values of α and epochs in Case 1 and Case 2 58

4.1 Achieved Bit Rate depending on the Location in Catania Port setting112
4.2 Transmission Delay using the LoRa/LoRaWAN technology for a SF=7, Bandwidth 250 kHz and a LoRa Gateway located G1. . . . 113
4.3 Packet Delivery Rate as a function of the LoRa Gateway position for SF=11. 115

ABSTRACT

In recent times, Machine Learning algorithms have gained significant interest in the area of infrastructureless networks. Infrastructureless wireless networks are an important class of wireless networks that is best applicable for scenarios where there is temporary and localized telecommunication demand. Machine Learning algorithms could play an important role in providing reliable and energy efficient communication and resource management; indeed it can be beneficial in handling the large volumes of computation and communication as required by evolving wireless applications.

In this context, this thesis was realized with the aim of studying the relevant aspects concerned with application of Machine Learning in infrastructureless networks. In particular, in the first part of this work the application of Machine Learning in the scenario of Wireless Sensor Networks is addressed. More specifically, the application of Federated Learning is studied and combined with model gossiping to provide energy efficient operations. Also, a software defined networking approach is introduced for wireless sensor networks with data mules. A simple machine learning algorithm is implemented to forecast the route of the data mule.

In the second part of this work, a Transfer Learning approach is introduced for detecting anomalies in driving behaviour. The transfer learning approach with convolutional neural networks based auto-encoders is applied to a specific case study in which riding data has been collected from different bicyclists riding along the same path. A layer separation approach is also introduced to partition the Neural Network layers of the model into some layers specific of the user behaviour and others layers specific of the road-environment. The layer separation approach has been experimented with the dataset.

Finally, this thesis explores also the use of ML in specific challenging infrastructureless networks, namely underwater networks. Specifically, a hybrid acoustic/LoRa system is designed and developed to collect data from underwater in real time. The collection of multimedia data in real time could pave the way for utilization of Machine Learning algorithms for evolving underwater applications.

Keywords: *5G*, *Machine Learning*, *Federated Learning*, *Transfer Learning*, *Layer Separation*, *Underwater Communication.*

Chapter 1

Introduction

In the recent years, there has been a drastic growth in the number of connected devices and in the number of Internet users. Therefore the requirement for next generation wireless networks is huge since they should provide energy efficient, low latency communication and intelligently control the connected devices in real time. Sensor measurements from Wireless Sensor Networks (WSNs), sensor readings from autonomous cars and other infrastructureless networking applications generate large volumes of data that must be collected and processed in real time. Infrastructureless networks are wireless networks that do not rely on an infrastructure such as base stations or access points. The communication and computing requirements can only be achieved through the inclusion of Machine Learning (ML) techniques in infrastructureless networks. In the recent past, ML algorithms have gained significant attention among the wireless networking and communication research communities. ML-based algorithms and techniques can enable wireless network analysis and resource management and can be beneficial in handling the large volume of communication and computation for evolving networking applications.

Machine learning (ML) techniques can dramatically increase the effectiveness and efficiency of *wireless sensor networks* (WSN) [1]. Both the academic and industrial R&D communities are showing increasing interest in the development of solutions for Machine Learning (ML) in Wireless Sensor Networks (WSNs). For example, ML can be exploited to optimise networking operations [2, 3] and locally process the data in order to extract the relevant information from the collected measurements[4]. Therefore, it will be sufficient to send significant information only, instead of large amounts of data, thus saving communication and energy resources. Striking evidence of the interest of companies on the use of ML for WSNs is that Google Brain has recently released *TensorFlow Lite*[5], an open-source ML framework that converts pre-trained TensorFlow models. These models can be thus executed in Android or iOS-based mobile phones, Linux-based embedded devices, and even micro-controllers.

Nowadays, the use of TensorFlow Lite is in line with the widespread vision regarding the use of ML in WSN: the models are executed in the sensor nodes (*inference*), while *training* is performed in some resource-rich servers. Indeed, model training implies heavy computations which cannot be executed by the platforms usually employed to realize WSN nodes. However it is expected that future generation WSN devices will be able to perform this in-network training since embedded devices are becoming more and more powerful (e.g. new Raspberry Pi platforms support the full TensorFlow 1.9 and, thus, are able to do training). Also, under the point of view of distributed learning techniques, novel techniques are being developed which ask for lower amount of resources in those nodes which take part to the learning process [6]. Many communities are working in this direction and the support for training on-device is in the TensorFlow Lite development roadmap[1]. So executing the **training process in** the sensor nodes will be not only possible, but will represent the ultimate horizon of the *smart dust* concept, which has now reached its adulthood [7].

This thesis aims to analyze aspects related to the exploitation of ML in infrastructureless networks. In particular, the main objectives of the thesis are:

– Wireless Sensor Networks: In this context, the first work regards the design and development of distributed ML algorithms in WSN. The application of well known Federated Learning (FL) mechanism in WSN is studied. Protocols were designed by combining FL and Model Gossiping to save energy and communication resources and to overcome the funnelling effect. Additionally, centrality metrics were exploited in achieving the convergence quickly. Also, a hybrid clustering framework was considered in which nodes have different ML training capabilities. The learning protocols were experimented with different ML algorithms such as auto-encoders, Long Short Term Memory (LSTM) network and Generative Adversarial Networks (GANs). The second work is about applying Software Defined Networking (SDN) to WSN with data mules. The movement of the data mule is forecast by the SDN controller and the forecast positions are considered to generate the flow table entries to be installed in the sensor nodes and schedule their applications. To this purpose, it is expected that the drone will move to the locations where abnormal conditions are observed by the sensors. A simple

[1]http://www.tensorflow.org/lite/guide/roadmap

and efficient decision tree algorithm is implemented, which takes the values measured by the sensors as inputs, to forecast the route of the data mule.

– Vehicular Networks: In this context, the first work is about monitoring and analysing the driving behaviour in real time. This can be helpful in detecting anomalies and warning the driver and nearby road users. Convolutional Neural Networks (CNN) based auto-encoders were applied for this task. Additionally, the transfer learning approach was incorporated to considerably reduce the data and time required for training the ML models. The transfer learning approach was applied to a specific case study in which data was collected from different bicyclists riding along the same path in the city of Catania. The second work is about achieving layer separation in deep neural networks for road-user interaction analysis in smart road environments. ML can play an effective role in analysing the driving behaviour, which could be helpful in detecting anomalies and thus, warning the driver timely. However, it is very demanding and crucial to build models for all the users which would be effective in all kinds of road conditions. In fact, it is obvious that every driver has her own driving style which would vary depending on the road-environment. To address this issue efficiently, a V2I framework is designed based on a training technique that partitions the Neural Network (NN) layers of the model into some layers specific of the user behaviour and others layers specific the road-environment. specific of the road-environment where the vehicle is currently located.

– Internet of Underwater Things: This work is about real time monitoring of underwater historical sites and the marine life. One of the most critical issues is indeed associated to the need for developing a network of devices which can connect and remotely deliver data to a front-end elaboration center. To this purpose, the need for increasing network lifetime and improving the quality of the monitored parameters, while guaranteeing real time control of the network, calls for the design of tools for remotisation, actuation and control. In this work, an integrated acoustic/LoRa system has been designed and developed for transmission of multimedia sensor data over the Internet of Underwater Things. The developed real time monitoring system paves way for the application of ML algorithms to various intelligent underwater applications.

This thesis work is structured as shown in Fig. 1.1. Chapter 2 will discuss the application of ML solutions in WSN scenarios. More specifically, the chapter ad-

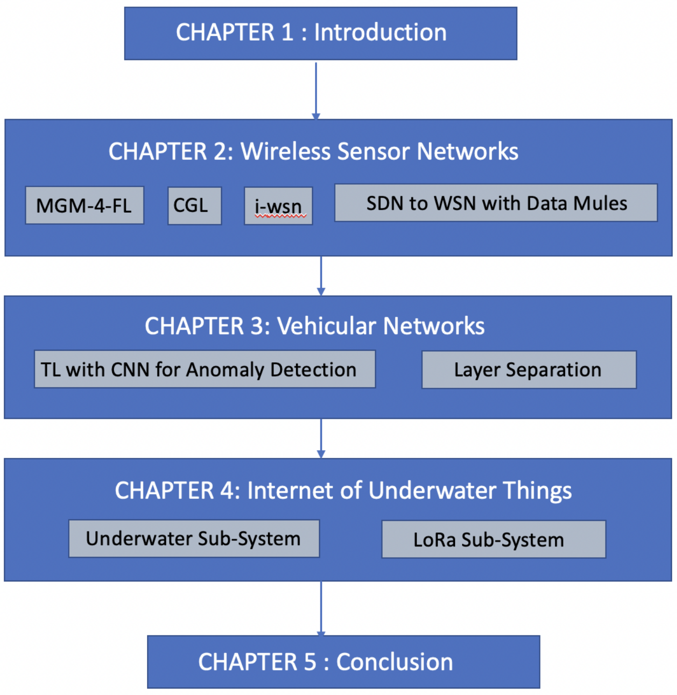

CHAPTER 1 : Introduction
CHAPTER 2: Wireless Sensor Networks
MGM-4-FL
CGL
i-wsn
SDN to WSN with Data Mules
CHAPTER 3: Vehicular Networks
TL with CNN for Anomaly Detection
Layer Separation
CHAPTER 4: Internet of Underwater Things
Underwater Sub-System
LoRa Sub-System
CHAPTER 5 : Conclusion

Figure 1.1: Structure of this thesis

dresses the challenges and solutions in the design of distributed and centralized ML in WSNs. Similarly chapter 3 will discuss the application of ML solutions in vehicular scenarios. In chapter 4, the underwater communication scenario is presented where ML can play an important role in the communication of multimedia data. Finally, in Chapter 5 the conclusions of this thesis will be drawn.

Chapter 2

Wireless Sensor Networks

2.1 Introduction

This chapter discusses the application of ML in the WSNs. In this context, the first set of works carried out were focused on the design and development of distributed ML protocols for WSNs. The concept of FL in WSNs and the challenges and issues in it were studied. To overcome the issues in applying FL in WSNs, several gossiping based distributed ML protocols were developed. The topological information of the network was considered in enhancing the protocols. Also, the application of distributed ML was studied in scenarios where the nodes exhibit different training and transmission capabilities. The developed protocols were experiment with various ML algorithms such as the auto-encoders, GANs and LSTM recurrent neural networks. The second work in this chapter was focused on applying SDN to WSN with data mules. The movement of the data mule is forecast by the SDN controller and the forecast positions are considered to generate the flow table entries to be installed in the sensor nodes and schedule their applications. To this purpose, it is expected that the drone will move to the locations where abnormal conditions are observed by the sensors. The well known decision tree algorithm was exploited which takes the values measured by the sensors as inputs, to forecast the route of the data mule.

2.2 Federated Learning (FL)

Conventional *centralized* ML approaches require to have data delivered to a central entity where such data is used for training a ML model. This may not be feasible in several scenarios because of the consequent privacy issues (e.g. in case of biomedical applications or chemical monitoring applications), as well as due to the resulting communication overhead and cost implied by the transmission of large data-sets [8], [9].

Deep Neural Networks are an example of ML application in a centralized network. Here the internal parameters of the neural network, i.e. the weights, are optimized to minimize a loss function, which measures the difference between the prediction done by the neural network and the real data [10]. An iterative procedure called *training* is executed to minimize the loss; indeed at each step, the system computes the gradient of the loss function w.r.t. the system weights and the training data set, and the result is used to update the weights correspondingly. This basic procedure is called Stochastic Gradient Descent (SGD) [11] which is so far the standard technique employed in Neural Networks.

However in real wireless network settings, nodes capabilities are limited and distributed *decentralized* approaches are more suitable. In fact, in distributed approaches the result of the local training is sent to a central server instead of the data. This methodology allows to split the computational load among multiple devices and reduce the amount of information transmitted by individual devices.

The most popular decentralized ML approach is *Federated Learning* (FL), which has been proposed by Google [12], and is experiencing a huge success especially in the framework of edge computing and caching. FL overcomes the limitations of traditional ML approaches that require centralizing the training data on one single machine. FL has indeed been proposed originally by Google [12, 13] as a decentralized mechanism using the **on-device processing capabilities** and the collected data to train the model in a distributed manner [12]. FL enables the involved nodes to collaboratively learn a shared model while keeping all the training data on device.In FL, each member of a group of *federated learners* trains its model locally, using its data only and sends it to some entities, called *aggregation points*, which are able to aggregate local models in a unique one. Then this aggregated model can be sent back to the distributed federated learners for further training and use.

Federated Learning has been proficuously used also in the framework of medical healthcare. For example, in [14] a survey on application of FL to healthcare is presented. Indeed the use of AI would imply centralized data collection and processing. This can be impossible due to relevant scalability issues associated to current healthcare networks and because of increasing data privacy concerns. FL exhibits in general also the advantage that data is kept where it was generated, so improving privacy.

Use of FL for smart healthcare was also addressed in [15] where it allows to co-

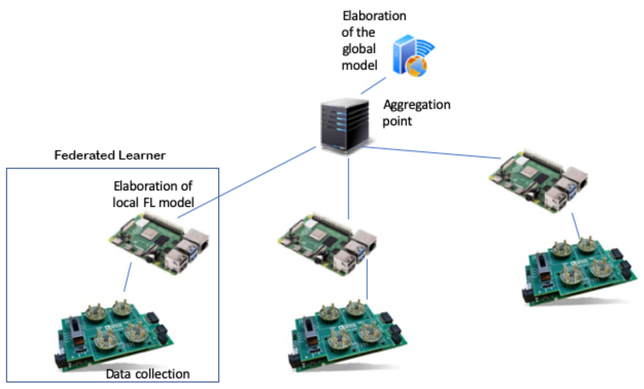

Elaboration of the global model
Aggregation point
Federated Learner
Elaboration of local FL model
Data collection

Figure 2.1: Federated Learning mechanism.

ordinate multiple clients (e.g., hospitals) to perform AI training without requiring data dissemination and sharing. In [15] the application of FL to study tumor segmentation is discussed. Also in [16] differentially private FL is applied for analysis of histopathology images, which represents the most complex and largest type of medical images. As compared to traditional techniques, this distributed training achieves comparable performance to conventional training, but with good privacy guarantees. FL offers also the advantage that models can be exchanged instead of large data-sets and this improves the communication resources efficiency.

To overcome the limitations of standard implementations of FL [8, 17, 18], while exploiting the advantageous features of multihop communications typical of WSNs, solutions have been recently proposed that further distribute learning. This is possible by enabling model aggregation at all network nodes (and not only in the aggregation point) that, therefore, exchange their models with their one hop neighbors.

In FL there are several *federated learners* that maintain a local part of the data-set and use it to train their models locally. Local data-sets can be different in size and features.

As sketched in Fig. 2.1, the locally elaborated models are sent to a server, called *aggregation point*, which aggregates them; the aggregated model is then sent to the learners again that might train it further, using the local data.

Let us assume to have a large population of wireless sensor nodes that possess raw data and have to use it to train their ML model. In order to avoid privacy issues and reduce bandwidth and energy consumption, these nodes do not disclose their raw data to the aggregation point. Indeed, at each iteration t, FL aims at solving a distributed optimization problem in the form [19]:

$$\min_{\mathbf{w}} F(\mathbf{w}) \tag{2.1}$$

being

$$F(\mathbf{w}) = \sum_{k=1}^{K} p_k F_k(\mathbf{w}) \tag{2.2}$$

where $F(\mathbf{w})$ is the objective function, i.e., the global loss function, $\mathbf{w}$ is the vector containing the model parameters, K is the number of network nodes, p_k is the weight of the k-th learner such that $\sum_{k=1}^{K} p_k{=}1$ and $F_k(\mathbf{w})$ is the loss function for the k-th node when the model is $\mathbf{w}$.

As an example, if we assume to weight in the same way all training data samples and denote the number of training data samples of the k-th federated learner as n_k and the number of training samples across the K federated learners in the network as n, i.e., $n = \sum_{k=1}^{K} n_k$, then p_k is defined as $p_k = \frac{n_k}{n}$. The local loss function for each learner k, $F_k(\cdot)$, needed in eq. (2.2), can be calculated as:

$$F_k(\mathbf{w}) = \frac{1}{n_k} \sum_{j=1}^{n_k} f(\mathbf{w}; \mathbf{X}_{k,j}) \tag{2.3}$$

where $\mathbb{X}_k{=}(\mathbf{X}_{k,1} \dots \mathbf{X}_{k,n_k})$ represent the n_k data samples available at the k-th learner and $f(\cdot,\cdot)$ is the per sample loss function using the trained model parameter $\mathbf{w}$ estimated for example in terms of mean square error or cross entropy. This function is obtained upon applying the model $\mathbf{w}$ and input data $\mathbb{X}_k$ where it is assumed to have non IID data.

At the generic epoch t, the k-th federated learner tries to minimize $F_k(\cdot)$ given in eq. (2.3) and sends the resulting model parameters $\mathbf{w}_k$ at each iteration to the aggregation point where the global model $\mathbf{w}$ is built as

$$\mathbf{w} = \sum_{k=1}^{K} p_k \cdot \mathbf{w}_k \tag{2.4}$$

Note that at each learner the process can be repeated for $t \in E$ epochs locally before sharing the models with the aggregation point in R iterations. The aggregation point sends the global model to the federated learners that will execute for another set of epochs the inference and then at next iteration send their $\mathbf{w}$ till the global loss converges as desired.

The FL mechanism is efficient in terms of accuracy and convergence of the model, although many researchers have recently focused on the associated cost for executing the different iterations [19].

Federated learning has been deployed in practice by major companies. Examples taken from [8] are:

- Google is applying FL in the Gboard mobile keyboard as well as in Pixel phones [20] and in Android Messages.
- Apple is using cross-device FL in iOS 13 for applications like the vocal classifier for "Hey Siri".
- doc.ai is developing cross-device FL solutions for medical research [21]
- Snips has explored cross-device FL for hotword detection [22].

Other relevant applications of FL have recently emerged in the field of smart city sensing [23]. In contexts relevant to our scenario, Liu *et al.* in [24] proposed a hybrid aerial-ground air quality sensing framework that exploits FL among Unmanned Aerial Vehicles (UAVs). The central server sends the copy of the global model to all the UAVs and each UAV improves the global model by learning on the local data-set.

Federated learning has been also recently proposed for privacy-preserving mobile scenarios. As an example, in [25], in order to limit privacy leakage in federated learning for mobile systems, authors proposed to use Trusted Execution Environments (TEE). Specifically the latter were employed in clients for local training as well as on servers for secure aggregation. Each model was trained inside the trusted area until convergence, while preserving privacy and not leading to serious system overhead. Another novel application of FL is presented in [26]. Here collaborative modeling is addressed by presenting the Federated Learning as a Service (FLaaS) paradigm while also providing a proof of concept by implementing it on a mobile phone setting.

2.2.1 Federated learning in WSN

In this section, the reason for why FL cannot be applied as it is in WSN is discussed.

A WSN is considered consisting of 8 nodes as sketched in Fig. 2.2, and let us assume that FL is applied and that node A is the aggregation point, whereas all other nodes denoted as L1, L2,..., L7 are the federated learners.

Models generated by the federated learners will traverse several hops before arriving at the aggregation point A. For example, the model generated by learner

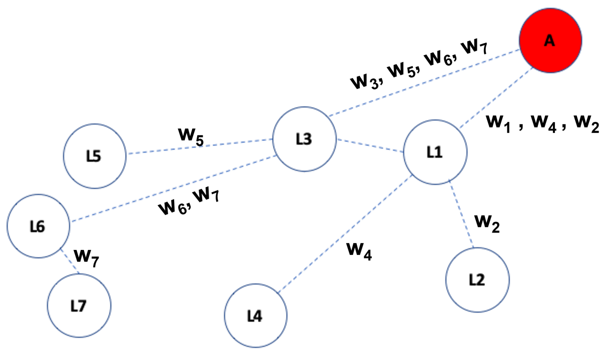


Figure 2.2: Wireless network scenario

L7 will pass through L6 and L3 before reaching A. This will require a large number of transmissions by wireless sensor nodes. More specifically, if we consider the tree spanning all network nodes, whose root is node A, we can calculate the number of model transmissions needed at each learning iteration as

$$\mathcal{N} = \sum_{l=1}^{H} N_{D_l} \cdot l \tag{2.5}$$

where H is the maximum depth of the aforementioned tree and N_{D_l} is the number of nodes with depth l in the spanning tree. In the example in Fig. 2.37, the number of model transmissions will be $\mathcal{N} = \sum_{l=1}^{3} N_{D_l} \cdot l = 2 \cdot 1 + 4 \cdot 2 + 1 \cdot 3 = 13$

Furthermore, it is clear that the nodes that are close to the root, A, will be involved in several relaying operations. For example in Fig. 2.37, node L3 is responsible for forwarding models $\mathbf{w}_3$, $\mathbf{w}_5$, $\mathbf{w}_6$, and $\mathbf{w}_7$ whereas nodes L2, L4, L5 and L7 will be responsible for the transmission of their models only. This is the well known funneling effect problem which is the reason for unfairness and may result in a rapid exhaustion of batteries in nodes close to the aggregation point A and, thus, as a consequence, can lead to a significant reduction in network lifetime. Funnelling effect refers to the intense utilization of communication resources and energy by the nodes located at close proximity to the aggregator node, due to the delivery of packets from the other nodes. Hence the nodes closer to the aggregator node consume the resources at a faster rate than the rest of the nodes [27],[28].

One obvious way to overcome this funneling effect is to exploit the multihop communication paradigm by aggregating models at intermediate nodes. For example, learner $L7$ sends its model parameters to its parent node $L6$. Instead of forwarding model $\mathbf{w}_7$ to its parent L3, node L6 performs an average of its own model parameters $\mathbf{w}_6$ and those sent by L7, $\mathbf{w}_7$, thus issuing $\mathbf{w}_6^{(A)}$. Only the resulting aggregated model will be forwarded up-words in the spanning tree towards the aggregation point A. In other terms, a node will wait for the models coming from all its clients in the routing tree. The node will aggregate them with its own model and will send only the result of such operation to its parent node. In our example, L3 will wait for the models coming from L5 and L6 and will send the aggregation of w_3, w_5 and $w_6^{(A)}$ to the aggregation point.

Such an approach would reduce the transmission significantly. However, it involves that the aggregated model is built in the aggregation point and then flooded in the network. Furthermore, it relies on knowledge of a route towards the aggregation point in each node, which is not always the case. To solve such issues, gossiping has been proposed as explained in the next Section 2.2.2.

2.2.2 Gossiping for FL

In this section, the major relevant research contributions and highlight how MGM-4-FL goes beyond the state of art is discussed.

Gossiping was initially conceived to solve the -so called- *consensus problem* by exploiting the computing resources at each node to reduce the amount of data that needs to be transmitted in the network. Therefore, gossiping can be used to save energy and communication resources, so extending network lifetime, and reducing latency [29–31].

Gossiping thus captures the condition where a set of network agents must achieve a shared *opinion* through exchanges of local information with neighbors. In WSN, gossiping has applications in distributed inference and detection [32, 33].

Recently, the use of gossiping has been investigated in the context of FL as well, where consensus has to be achieved regarding the ML model. Therefore it is assumed that each node has a value or set of values, which in our case are the model weights. The objective of gossiping is to allow all nodes to achieve shared estimation on the *average* of all models, which is what FL tries to do at each iteration.

Early examples of such schemes are presented in [34], [35], [36]. Nodes exploit the locally observed data and collaborate with their one hop neighbors to collectively learn a model that best fits the data collected by the entire network.

One of such schemes is proposed in [37] which is supported by theoretical results regarding its performance. In [38], a fully distributed scheme is proposed that adapts the transmission rates of individual nodes to control network density while keeping communication time required to exchange the models below appropriate thresholds. Significant step towards the practical applicability of gossiping is achieved in [39] where the network constraints are taken into account. A novel class of FL algorithms are introduced to improve convergence and a prototype implementation is presented. In [40], a thorough comparison of FL and Gossip Learning is provided and the authors examine the aggregated cost of machine learning in both the cases. Experiments are performed with different distributions of training data and the authors show that Gossip learning performs better than FL in all the scenarios where the training data is uniformly distributed over the devices. Similarly in [41], a segmented gossip approach is proposed which fully utilizes node to node bandwidth. The proposed segmented approach also has good training convergence. Experiment results show that the training time is considerably reduced when compared to the standard FL approach

Table 2.1 summarizes the relevant literature on Gossiping for FL discussed above.

2.3 MGM-4-FL: Combining Federated Learning (FL) and Model Gossiping

In this section, the MGM-4-FL (Model Gossiping Method for Federated Learning) protocol is introduced which is based on combining FL and model gossiping.

More specifically in Section 2.3.1, the design objectives and rationale of MGM-4-FL are discussed. Then in Section 2.3.2,the operations of MGM-4-FL are illustrated.

2.3.1 Objectives and rationale

Objective of MGM-4-FL is to support distributed learning by applying gossiping in energy constrained WSN, based on the multihop wireless communication paradigm. Therefore, under certain circumstances that will be described later, a given node will transmit its model parameters to its one hop neighbors. Such neighbors will use the received parameters to update their own model and will further train it. Once said j the node that has transmitted its model parameters, $\mathbf{w}_j$, its generic neighbor k will update its model as

$$\mathbf{w}_k = \alpha \cdot \mathbf{w}_j + (1 - \alpha) \cdot \mathbf{w}_k \tag{2.6}$$

where α is a weight parameter that we set larger than 0.5 as discussed later.

Relevant Related work				
Ref.	Year	ML Algorithm	Theoretical Analysis	Description
[36]	2021	Auto-Encoders	No	Gossiping combined with FL to improve the efficiency of communication resources use in WSNs. Simulation results show that gossiping can help in improving significantly the performance in terms of resource efficiency
[37]	2019	Deep Neural Network and Linear Regression	Yes	Fully decentralized framework in which nodes communicate only with their one-hop neighbours to learn the shared model. The proposed work is supported by theoretical results.
[38]	2020	Convolutional Neural Networks	Yes	Decentralized learning approach that adapts the transmission rates of individual nodes to control network density while keeping communication time required to exchange the models below appropriate thresholds.
[39]	2020	Convolutional Neural Networks and 2 Layers Neural Network	Yes	Consensus-Based fully Federated averaging approach to improve convergence with a prototype implementation.
[40]	2019	Logistic Regression	No	Comparison of Federated Learning and Gossip learning algorithms. Simulations results show that gossip learning is a suitable decentralized alternative for Federated Learning.
[41]	2019	Convolutional Neural Networks	No	A segmented gossiping approach that fully utilizes node-to-node bandwidth for good training convergence.

Table 2.1: Summary of relevant papers.

Note that if the loss function $F_k(\mathbf{w})$ is convex, the operation in eq. (2.16) is performed only in the case $F_k(\mathbf{w_j}) < F_k(\mathbf{w_k})$. When the loss function $F_k(\mathbf{w})$ is not convex, we will update $\mathbf{w}_k$ with a probability which decreases when the difference between $F_k(\mathbf{w_j})$ and $F_k(\mathbf{w_k})$ increases.

Such sequence of actions are denoted as *protocol iteration* or *iteration*, in short.

A further constraint of MGM-4-FL is that each node should exploit local information only, i.e., information produced by the node itself and the one hop neighboring nodes. As a consequence, MGM-4-FL operations must be asynchronous and uncoordinated.

Given the energy limitations characterizing WSNs, MGM-4-FL operations must be as effective as possible; thus, model transmissions and consequent training iterations, should be executed when the resulting expected reduction in the overall loss, $F(\mathbf{w})$ in eq. (2.2), is significant. To this goal,

- at the end of a protocol iteration, each involved node will broadcast its updated loss value;
- nodes store the values of loss broadcast by their neighbors. Therefore, each node, k, maintains updated values of $F_j(\mathbf{w}_j)$, for all $j \in \Phi_k$, where Φ_k represents the set of neighbors of k;
- each node, k, schedules the transmission of its model parameters after a time interval of duration proportional to $F_k(\mathbf{w}_k)/\max_{j\in\Phi_k} F_j(\mathbf{w}_j)$.

As a result, iterations are expected to be triggered by nodes that have a high value of the ratio $\max_{j\in\Phi_k} F_j(\mathbf{w}_j)/F_k(\mathbf{w}_k)$ and thus the MGM-4-FL scheme is expected to avoid iterations that are likely to bring low loss reductions.

Furthermore, the communication and computing load must be distributed between all network nodes as fairly as possible. This is necessary to avoid that a few nodes are overloaded and, thus, their batteries are exhausted rapidly, so reducing the lifetime of the entire network. To achieve this goal, the MGM-4-FL scheme exploits the *boost factor* which has a high value for nodes that have received the model parameters sent by one of their neighbors, say k, and is reset to one for node k. As a result, the boost factor is expected to be high for nodes that did not transmit their model parameters in the recent past and low for the nodes that, instead, did transmit their model parameters recently. Each node schedules the transmission of its model parameters after a time interval of duration reciprocal to its boost factor. Note that since the boost factor is increased by nodes that have just received the model parameters by one of their neighbors, its use facilitates the spread of fitting models.

2.3.2 MGM-4-FL Protocol Details

In this section more details regarding the MGM-4-FL operations and how they work in a simple scenario is discussed. More specifically, a network of four nodes deployed according to a linear topology is considered as shown in Fig. 2.3 (leftmost corner).

Algorithm 1 represents the pseudocode for the protocol run by the generic node k. In the pseudocode LL_k represents the maximum loss of the neighboring nodes of k, that is $LL_k = \max_{j \in \Phi_k} F_j(\mathbf{w}_j)$

At the startup, the MGM-4-FL node initializes the boost factor and the maximum loss of the neighboring nodes to one, i.e., $B_k = 1$ and $LL_k = 1$; furthermore, the model is trained using the local data X_k and starting from random initial conditions, RND. The training operation is executed for a given number of epochs, locally, at each learner. Therefore, at the end of the initialization phase, which is sketched in lines 1-5 of Algorithm 1, node k will get a local model $\mathbf{w}_k$ and an estimate of its current loss $F_k(\mathbf{w}_k)$. Using such initial parameters the node sets a timeout at time $t_k^{(\mathrm{next})} = t + T \cdot F_k(\mathbf{w}_k)/(B_k \cdot LL_k)$, where t represents the current time and T is a constant value which is set in such a way that protocol operations are sparse in time[1].

For example, in Fig. 2.3 the loss values for the four nodes at the end of the initialization phases are $F_1(\mathbf{w}_1) = 312.76$, $F_2(\mathbf{w}_2) = 52.20$, $F_3(\mathbf{w}_3) = 29.16$, and $F_4(\mathbf{w}_4) = 48.57$ and therefore, the four nodes will set their timeouts at times $t_1^{(\mathrm{next})} = 312.76 \cdot T$, $t_2^{(\mathrm{next})} = 52.20 \cdot T$, $t_3^{(\mathrm{next})} = 29.16 \cdot T$, and $t_4^{(\mathrm{next})} = 48.57 \cdot T$, respectively. Note that Fig. 2.3 was obtained assuming that the startup time is $t = 0$ for all nodes.

When the initialization phase is completed, MGM-4-FL executes the regular operations, sketched in lines 6-34 of the Algorithm 1, which are event based. More specifically, three types of events can occur:

1. The local timeout elapses, i.e., current time is $t_k^{(\mathrm{next})}$: in this case, the node executes the operations sketched in lines 9-16 of Algorithm 1.

2. The model parameters, $\mathbf{w}_j$, of the neighboring node j are received: in this case the node will execute the operations sketched in lines 17-28 of Algorithm 1, including retraining of the model for a given number of epochs.

3. The new loss value, $F_j(\mathbf{w}_j)$ of the neighboring node j is received: in this case the node will execute the operations sketched in lines 29-34 of Algorithm 1.

[1]In our experiment, we have set T equal to a constant proportional to the average time required to train the model local data.

Algorithm 1 MGM-4-FL protocol

```
 1: /* Protocol initialization */
 2: Initialize $B_k$=1
 3: Initialize $LL_k$=1
 4: Initialize $\mathbf{w}_k$=train_model($\mathbb{X}_k$, RND)
 5: $t_k^{(\text{next})} = t + T \cdot F_k(\mathbf{w}_k)/(B_k \cdot LL_k)$
 6: /* Regular operations */
 7: while Protocol initialization do
 8:     Wait for Event
 9:     if Event.Type==time-out then
10:         /* Node k will send its model parameters $\mathbf{w}_k$ */
11:         Broadcast($\mathbf{w}_k, F_k(\mathbf{w}_k)$)
12:         Wait for the updated loss values
13:         $LL_k = \max_{j\in\Phi_k} F_j(\mathbf{w}_j)$
14:         $B_k = 1$
15:         $t_k^{(\text{next})} = t + T \cdot F_k(\mathbf{w}_k)/(B_k \cdot LL_k)$
16:     end if
17:     if Event.Type == received $\mathbf{w}_j$ then
18:         /* Node k is receiving the model parameters $\mathbf{w}_j$ from neighbor node j
    */
19:         $B_k = B_k + 1$
20:         $LL_k = \max_{j\in\Phi_k} F_j(\mathbf{w}_j)$
21:         $t_k^{(\text{next})} = \min\{t_k^{(\text{next})}, t + T \cdot F_k(\mathbf{w}_k)/(B_k \cdot LL_k)\}$
22:         Evaluate $F_k(\mathbf{w}_j)$
23:         if $F_k(\mathbf{w}_j) \leq F_k(\mathbf{w}_k)$ then
24:             $\mathbf{w}_k = \alpha \cdot \mathbf{w}_j + (1-\alpha) \cdot \mathbf{w}_k$
25:             $\mathbf{w}_k$ = train_model($\mathbb{X}_k, \mathbf{w}_k$)
26:             Evaluate and Broadcast $F_k(\mathbf{w}_k)$
27:         end if
28:     end if
29:     if Event.Type == $F(\mathbf{w}_j)$ received then
30:         /* Node k is receiving the updated loss value $F(\mathbf{w}_j)$ from node j */
31:         $LL_k = \max_{j\in\Phi_k} F_j(\mathbf{w}_j)$
32:         $t_k^{(\text{next})} = \min\{t_k^{(\text{next})}, t + T \cdot F_k(\mathbf{w}_k)/(B_k \cdot LL_k)\}$
33:     end if
34: end while
```

As reported in lines 9-16 of Algorithm 1, when the timeout elapses, the node will broadcast its model parameters $\mathbf{w}_j$ and then will wait for receiving the new values of loss estimated by the neighboring nodes. In fact, the neighboring nodes upon receiving the model parameters, under certain circumstances, will send updated values of their loss. This will be explained later and is sketched in line 26 of Algorithm 1.

In the case shown in Fig. 2.3, node 3 has the lowest value of the timeout parameter, i.e. $t_3^{(\text{next})} = 29.16 \cdot T$. Therefore, as reported in line 9, it will broadcast a message containing its model parameters $\mathbf{w}_3$. Such message will be received by nodes 2 and 4 which will execute the operations sketched in lines 17-27 of Algorithm 1. More specifically, each of them will first increase the boost factor by one, i.e., $B_k = B_k + 1$, and update the LL_k as well as the corresponding $t_k^{(\text{next})}$ value as shown in lines 19-21. Then it will evaluate the loss it would achieve by exploiting the model $\mathbf{w}_3$ (see line 22). In other words, nodes 2 and 4 will evaluate $F_2(\mathbf{w}_3)$ and $F_4(\mathbf{w}_3)$, respectively.

If the estimated value of loss is higher than the current one, no other actions will be executed and the node will wait for a new event. This is the case for node 4 which sets $B_4 = 2$, $LL_4 = 29.16$, and updates the $t_4^{(\text{next})}$ value accordingly, i.e.,

$$t_4^{(\text{next})} = t + T \cdot F_4(\mathbf{w}_4)/(B_4 \cdot LL_4) =$$

$$= T29.16 + T \cdot 48.57/(2 \cdot 29.16) = 29.99 \cdot T \tag{2.7}$$

Instead, if the estimated value of loss, $F_k(\mathbf{w}_j)$, is lower than or equal to the current one, $F_k(\mathbf{w}_k)$, the node executes the operations in lines 23-27. This is the case of node 2, instead, which updates its model parameters $\mathbf{w}_2$ as $\mathbf{w}_2 = \alpha \cdot \mathbf{w}_3 + (1 - \alpha) \cdot \mathbf{w}_2$ and then trains the model using local data, evaluates the new value of the loss $F_2(\mathbf{w}_2)$ and broadcasts this.

In the example shown in Fig. 2.3, the resulting value of $F_2(\mathbf{w}_2)$ is 16.42. Since $LL_2 = 29.16$ and $B_2 = 2$, the timeout is set at time $t_2^{(\text{next})} = 29.44 \cdot T$.

Note that the message broadcast by node 2 containing the new value of $F_2(\mathbf{w}_2)$ is received by node 1. The operations that node 1 executes as a consequence of such event are sketched in lines 29-34 of Algorithm 1. More specifically, node 1 will only update $LL_1 = 16.42$ and set the timeout accordingly, $t_1^{(\text{next})} = 48.21 \cdot T$.

Note that the next node that will send its own model parameter is node 2 that at time $t_2^{(\text{next})} = 29.44 \cdot T$ will broadcast its model parameters, $\mathbf{w}_2$, which will be received by nodes 1 and 3. Then it will be the turn of node 3 that, at time $t_3^{(\text{next})} = 29.61 \cdot T$, will transmit its new model parameters, $\mathbf{w}_3$.

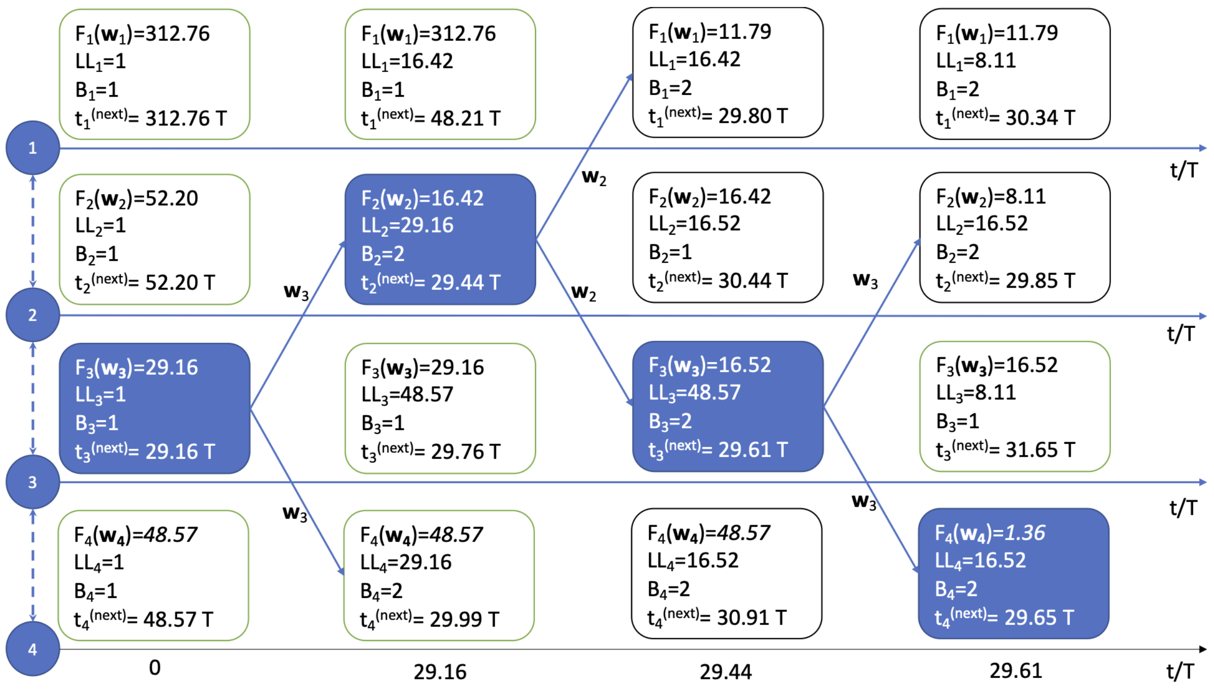

F1(w1)=312.76 LL1=1 B1=1 t1(next)= 312.76 T
F1(w1)=312.76 LL1=16.42 B1=1 t1(next)= 48.21 T
F1(w1)=11.79 LL1=16.42 B1=2 t1(next)= 29.80 T
F1(w1)=11.79 LL1=8.11 B1=2 t1(next)= 30.34 T
F2(w2)=52.20 LL2=1 B2=1 t2(next)= 52.20 T
F2(w2)=16.42 LL2=29.16 B2=2 t2(next)= 29.44 T
F2(w2)=16.42 LL2=16.52 B2=1 t2(next)= 30.44 T
F2(w2)=8.11 LL2=16.52 B2=2 t2(next)= 29.85 T
F3(w3)=29.16 LL3=1 B3=1 t3(next)= 29.16 T
F3(w3)=29.16 LL3=48.57 B3=1 t3(next)= 29.76 T
F3(w3)=16.52 LL3=48.57 B3=2 t3(next)= 29.61 T
F3(w3)=16.52 LL3=8.11 B3=1 t3(next)= 31.65 T
F4(w4)=48.57 LL4=1 B4=1 t4(next)= 48.57 T
F4(w4)=48.57 LL4=29.16 B4=2 t4(next)= 29.99 T
F4(w4)=48.57 LL4=16.52 B4=2 t4(next)= 30.91 T
F4(w4)=1.36 LL4=16.52 B4=2 t4(next)= 29.65 T
w2
w2
w3
w3
w3
w3
1
2
3
4
t/T
0
29.16
29.44
29.61

Figure 2.3: MGM-4-FL protocol in action

Note that with just three transmissions of model parameters, the values of the loss in the four nodes become $F_1(\mathbf{w}_1) = 11.79$, $F_2(\mathbf{w}_2) = 8.11$, $F_3(\mathbf{w}_3) = 16.52$, and $F_4(\mathbf{w}_4) = 1.36$. Such values are much lower than the initial ones and, as a result, the average loss, which was equal to $(312.76{+}52.2{+}29.16{+}48.57)/4{=}110.67$ at time 0, rapidly decreases to $(11.79{+}8.11{+}16.52{+}1.36)/4{=}9.44$ at time 29.61 T.

2.3.3 Experimentation Scenario

In this section, the MGM-4-FL is applied to a specific case study. Then, in Section 4.7, the numerical results are presented.

For the experiments, the data-set considered contained the values of temperature, humidity, and concentrations of several pollutants (i.e., PM1, PM2.5, PM10) measured by 56 low cost sensors deployed in the city of Krakow (Poland) in 2017[2].

Since, some of the sensors do not have complete data, among these 25 sensors were selected which provide a complete dataset. The topology of the sensor network is shown in Fig. 2.4. The above topology has been constructed considering the real position of the sensors and assuming that each of them is equipped with a wireless interface giving a radio coverage of 4 km.

[2]https://www.kaggle.com/datascienceairly/air-quality-data-from-extensive-network-of-sensors

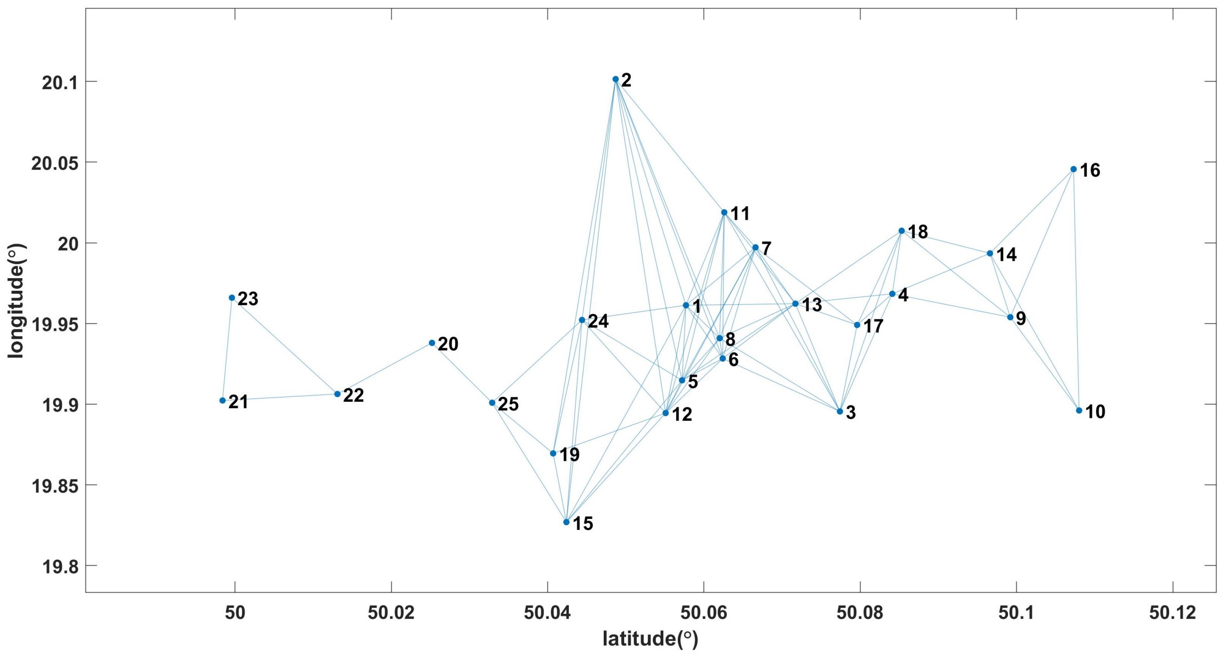


Figure 2.4: Wireless network scenario for the simulation.

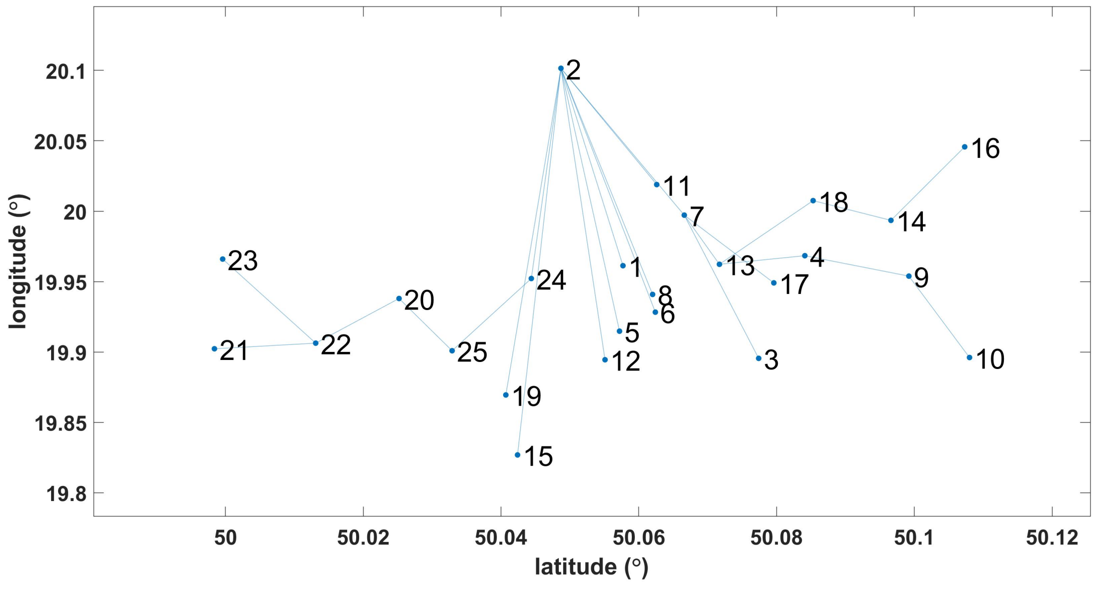


Figure 2.5: Illustration of the WSN scenario in the form of Tree Topology for standard FL

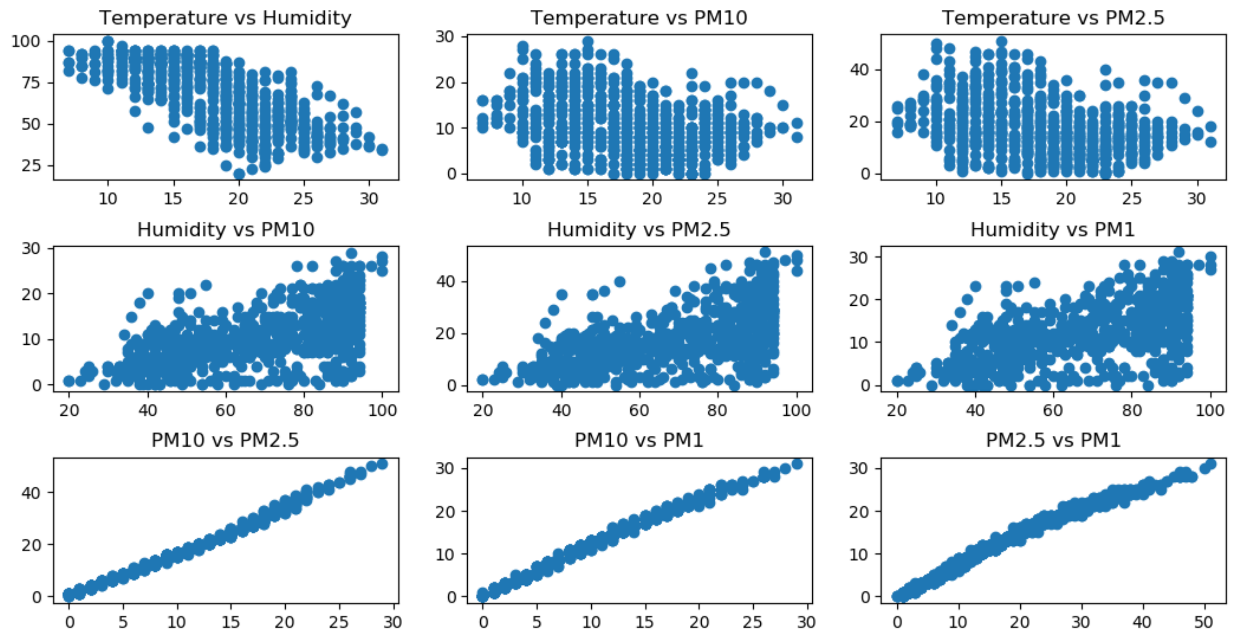


Figure 2.6: Training Data Characterization

In Fig. 2.5, the WSN scenario is shown in the form of a tree topology. If standard FL is employed, and assuming that node 2, which is the most central, performs the aggregation of the model parameters, nodes 24 and 7 will experience the funnelling effect as the model parameters from nodes 20, 21, 22, 23, and 25 have to pass through 24 and model parameters from nodes 3, 4, 9, 10,13,14, 16, 17, and 18 have to pass through node 7. On the other hand, nodes 1, 5, 6, 8, 11, 12 and 15 will send just one packet to the sink per each iteration. Thus the nodes 24 and 7 will consume their resources at a a faster rate than the rest of the nodes. As a result fairness in energy consumption is violated and this leads to a reduction in network lifetime.

In Fig. 2.6 the data features are characterized through some scatter plots. Specifically, in this figure we represent the values of temperature, humidity, PM1, PM2.5 and PM10 in different combinations.

MGM-4-FL can be applied whatever is the ML approach utilized. Nevertheless, in the experiments conducted, an assumption was made that each sensor node executes an autoencoder [42] which represents a traditional methodology for anomaly detection [43].

An autoencoder is a type of artificial neural network that operates in an unsupervised way to learn efficient data encoding. In fact, backpropagation is used to achieve a set of target values as close as possible to the input ones.

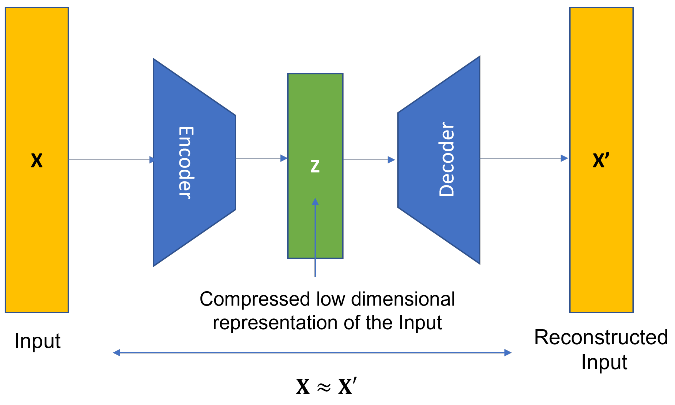


Figure 2.7: Model of an autoencoder.

In Fig. 2.7 we present the architecture of an autoencoder. Besides learning an efficient representation of data, autoencoders can denoise and decorrelate data.

The focus was not on the specific ML approach, therefore, in the experiments conducted, the simplest type of autoencoder was employed, the - so called - *Vanilla autoencoder* which consists of one hidden layer, only. Accordingly, in the experiments, the autoencoder consists of a three layer neural network, thus having one input, one hidden and one output layer. In the encoding process, the autoencoder first converts the input vector $\mathbf{X}$ into a hidden representation Z using an appropriate weight matrix $\mathbf{w}'$. During the decoding, instead, the autoencoder maps the hidden representation back to obtain the original format and obtains $\mathbf{X}'$ through another weight matrix $\mathbf{w}''$. The model parameter optimization is aimed at minimizing a loss measure which is proportional to the average reconstruction error, i.e., the difference between $\mathbf{X}$ and $\mathbf{X}$, given the input distribution. More specifically, we have that $L = \psi(\mathbf{X}, \mathbf{X}')$. Often Mean Square Errors (MSE) is utilized to provide an estimation of the reconstruction accuracy [43], i.e. $\psi(\cdot,\cdot) = MSE(\cdot,\cdot)$.

Since the sensors collect one value for each parameter every hour, the data-set for one month (i.e. 30 days) considered for each of the 25 sensors consists of $n_k = 24 \cdot 30 = 720$ entries of 7 values, with $k = 1, 2, ..., 25$. These 7 values represent the day, time,temperature, humidity, and PM1, PM2.5 PM10 parameters measured in the time period July 1-30, 2017.

The $u-th$ entry in the data-set of the $k-th$ sensor, denoted as $\mathbf{X}_{k,u}$, is:

$$\mathbf{X}_{k,u} = (X^{(0)}_{k,u}, X^{(1)}_{k,u}, , ..., X^{(4)}_{k,u}) \tag{2.8}$$

Note that the input size of the auto-encoder used in the experiments is 5, i.e., the values of temperature, humidity, PM1, PM2.5 and PM10. The intermediate size is 3 which is also known as code or compressed dimension.

2.3.4 Numerical results

In this section, the numerical results are reported that are obtained in the scenario setting described in the previous Section by applying the MGM-4-FL protocol. Several simulation campaigns were conducted to estimate the average loss, how many times any node has been trained, and how many times a node has broadcast its model parameters to its one hop neighbors. Note that the average loss is obtained by the average of the mean square loss of the sensor nodes. The mean square loss is evaluated from the data reconstructed by the autoencoder and the input data fed into the autoencoder according to the eq. (3.5), i.e.,

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2 \tag{2.9}$$

where y is the input data fed into the autoencoder, $\hat{y}$ is the reconstructed output data and N is the length of the input data.

A number of 10 simulations were executed for different values of α (i.e. α =0.5, 0.7, 0.9 and 1), and different number of *epochs* (i.e., 10, 15 and 20). Note that epochs denote the number of passes the machine learning algorithm will work through the entire local data-set every time training is triggered.

The average loss with respect to the number of iterations for the MGM-4-FL mechanism when α is 0.7, is shown in Fig. 2.8. We explicitly observe that in MGM-4-FL each iteration involves the broadcast of the model parameters of one node. In Fig. 2.8, we observe that, as soon as we increase the number of iterations, the average loss significantly decreases and starting from approximately 20 iterations, the impact of a different choice in epochs is minimal. This means that we can easily keep minimum the number of times the machine learning algorithm works through the complete data-set while preserving accuracy.

Fig. 2.9 shows the average loss with respect to the number of iterations for different values of α and 15 epochs. Once fixed the value of the epochs, we note that the impact of the parameter α is minimal which witnesses the good stability of the algorithm independently of how much we trust into neighbor models.

Also the standard deviation of the loss is reported with respect to the number of iterations for the MGM-4-FL mechanism when α is 0.5 and when α is 0.9, in Figs. 2.10a and 2.10b, respectively, and for different numbers of iterations. Note that, as soon as we increase the weight α in eq. (2.16) we increase the relevance given to neighbor model parameters. Specifically observe that an increase in α leads

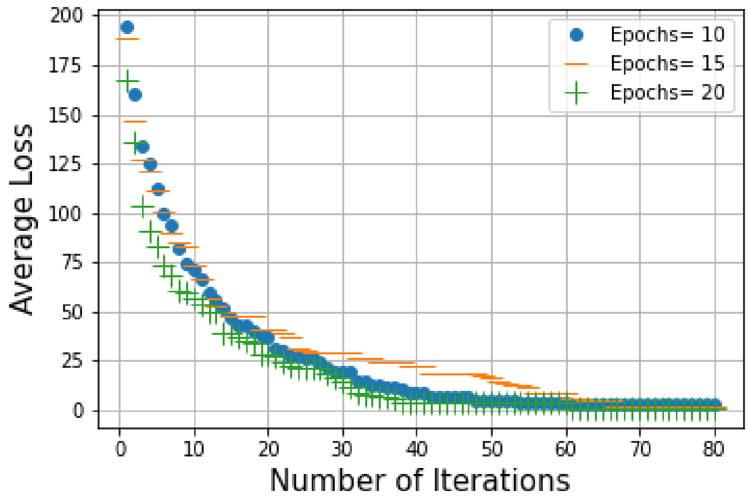


Figure 2.8: Average Loss vs. No. of Iterations for different values of epochs (MGM-4-FL)

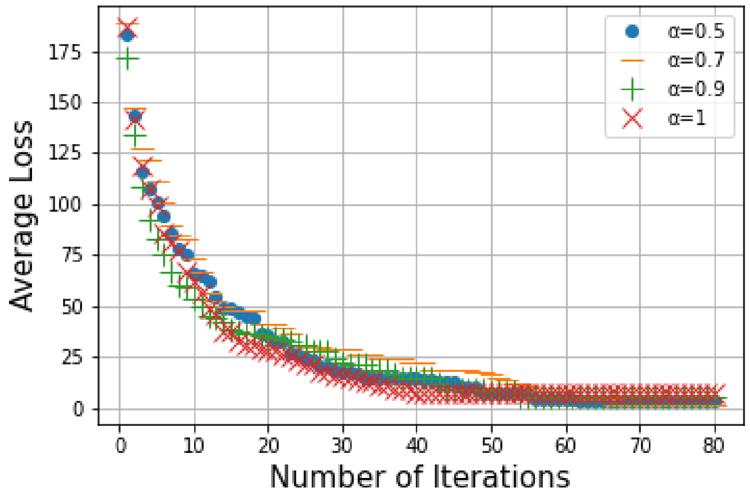
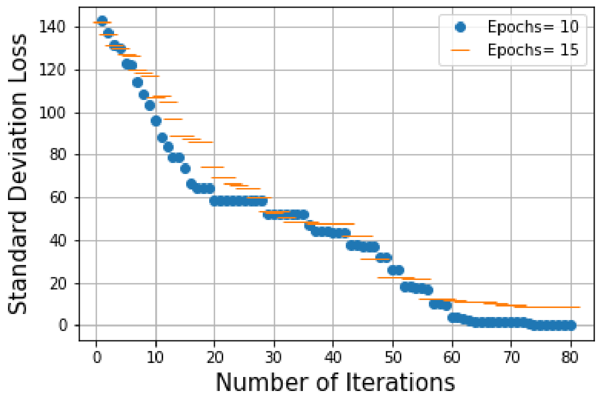
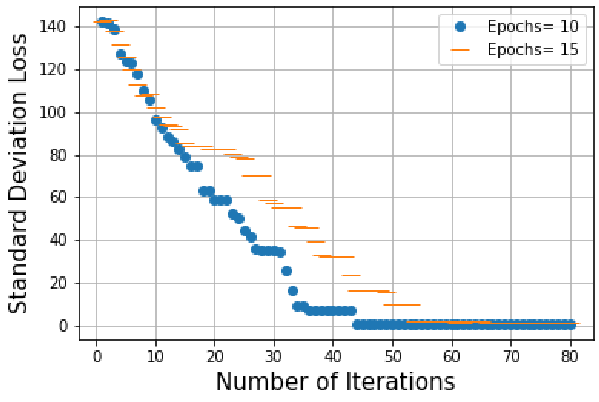


Figure 2.9: Average Loss vs. No. of Iterations for different values of α (MGM-4-FL)



(a) $\alpha = 0.5$



(b) $\alpha = 0.9$

Figure 2.10: Standard deviation of the Loss vs. No. of Iterations (MGM-4-FL)

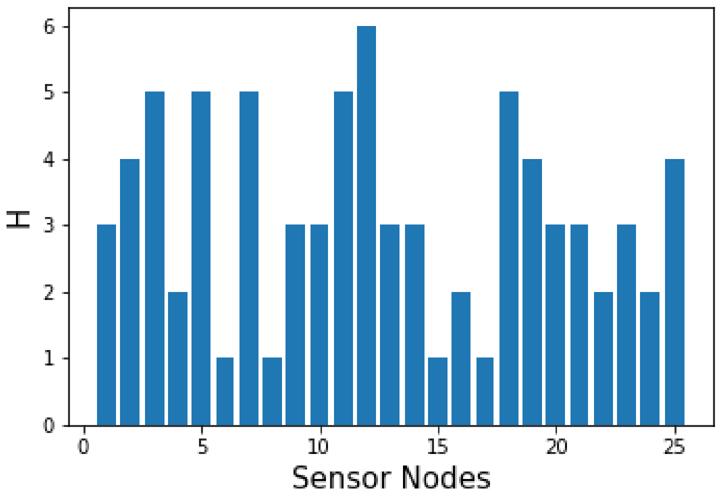


Figure 2.11: Number of times each node has broadcast its model parameters

to a faster convergence for a lower number of iterations. Also, higher values of α imply more coherence in the standard deviation, independently of the number of epochs.

Similarly, the number of times(H) each node broadcasts its model parameters is reported in Fig. 2.11. Note that in this plot nodes are not sorted out in terms of how many times they have been selected but based on their IDs as reported in Fig. 2.4. Observe how central nodes, like Node 3, 5, 7, 12, 18 and 25 broadcast their model parameters more often than the others. This is because, due to their centrality, they rapidly obtain models with good performance. On the other hand, nodes like 6, 8, 15, 17, 22 and 24 which are placed at the periphery of the network, obtain good models later.

In order to assess fairness, the Jain's Fairness index was calculated for both the number of times the nodes have been trained and the number of times they have broadcast their model parameters.

The Jain's Fairness index is calculated as [44]:

$$\mathcal{F}(x_1, x_2, ...x_n) = \frac{(\sum_{i=n}^{n} x_i)^2}{n \cdot \sum_{i=n}^{n} x_i^2} \tag{2.10}$$

for $n = 25$ sensors. In Table I we report the Jain's Fairness index for different values of α and the number of epochs.

Observe that the main fluctuations in terms of fairness are obtained upon varying the α parameter. Upon changing the number of epochs, however, we note that the number of times a node is trained or broadcasts its model parameters remains basically stable because the algorithm is reliable and quickly converges.

A comparison was made between MGM-4-FL with plain model gossiping in which each node periodically broadcast its model parameters (*Standard Gossiping*). In Fig. 2.12 the average loss versus the number of times a node broadcasts its model parameters is reported. The performance improvement given by MGM-4-FL when compared to Standard Gossiping is evident. For example, after 55 model parameters broadcasting events the loss achieved by MGM-4-FL is 5.313, whereas applying standard gossiping the loss would be 57.680 at 55, so more than 10 times higher. This assesses the good learning features of the methodology.

2.3.5 MGM-4-FL with GANs

: In this section, the MGM-4-FL algorithm is experimented with the ML algorithm "Generative Adversarial Networks (GANs)". GANs is a generative modeling ML algorithm unlike the Auto-encoders. Generative modeling is an unsu-

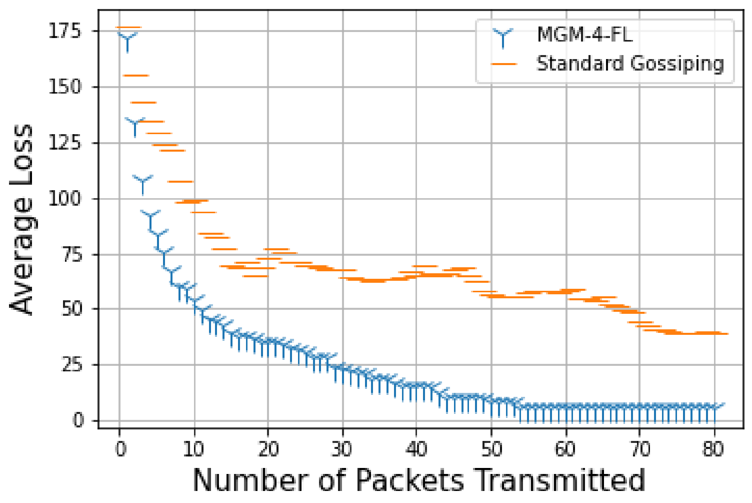


Figure 2.12: Average Loss vs. the Number of Packets Transmitted

pervised learning algorithm that learns the patters in the input data, so that the model can be used to generate output data that is very similar to the original input data. GANs basically comprises of two neural networks that compete with each other to analyse the replicate the patterns in the dataset [45, 46].

As shown in the figure 2.13, there is a Generator and Discriminator in GANs. The task of the Generator is to generate fake samples of data to fool the Discriminator and the data can be image, audio or numerical data. Whereas, the task of the Discriminator is to distinguish between the real and fake data.

The experiments were conducted with the same setup as mentioned in section 2.3.3. The average loss with respect to the the number of iterations is shown in figure 2.14. As shown in the figure, the average loss of all the 25 nodes reduces from 1.4 to 0.7 in 50 iterations by exploiting the MGM-4-FL protocol.

2.4 CGL: Centrality aware Gossiping

In recent times, federated learning learning solutions have emerged to enable collaborative synthesis of neural network models. These approaches cannot be applied in *wireless sensor networks* (WSNs) as they are because they involve a large volumes of network traffic to reach convergence, which is costly in terms of communication and computing resources that are particularly scarce resources in WSNs. Gossiping, which naturally fits with the multihop communication paradigm featured in most WSN scenarios, can reduce the amount of traffic significantly and therefore, quite a few solutions have been recently proposed that exploit it.

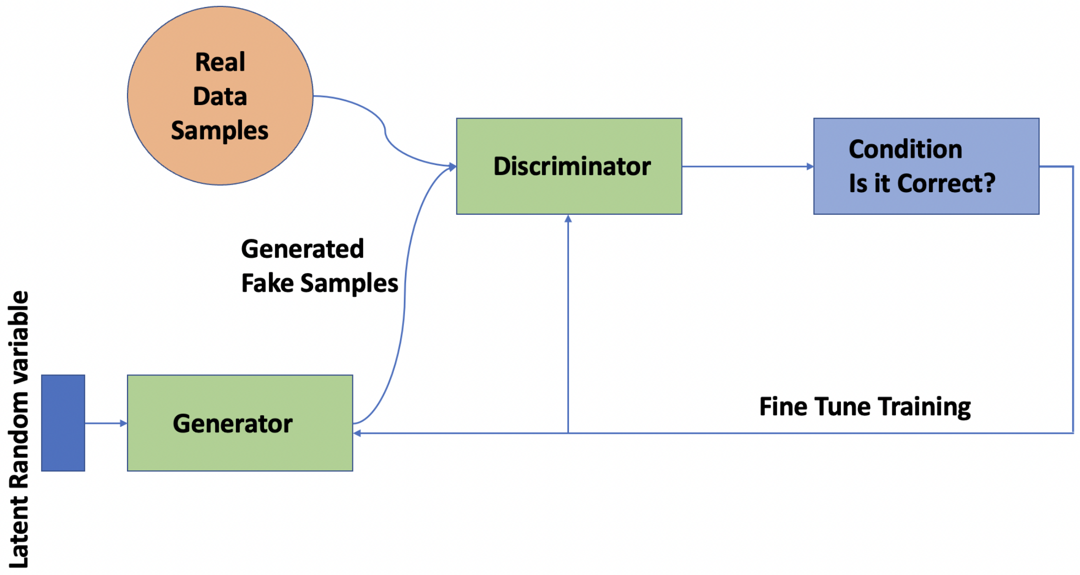


Figure 2.13: Generative Adversarial Networks

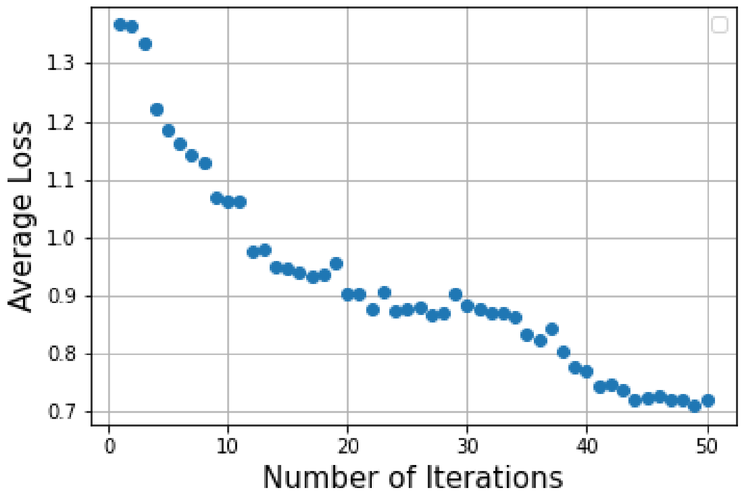


Figure 2.14: Average Loss vs Iterations

α	Epochs	Fairness in Train. Events	Fairness Broadc. Events
0.5	10	0.71379	0.77949
0.5	15	0.71909	0.76911
0.5	20	0.72692	0.77998
0.7	10	0.70785	0.81575
0.7	15	0.71143	0.80103
0.7	20	0.70974	0.80482
0.9	10	0.69520	0.81388
0.9	15	0.687528	0.80684
0.9	20	0.674327	0.82902
1	10	0.675843	0.80486
1	15	0.66990	0.80329
1	20	0.673902	0.80471

Table 2.2: Fairness Index for the number of training events and the number of times a node is chosen as best node for different values of α and epochs.

It is known that gossiping can be executed more efficiently if the characteristics of network topology are taken into account. Accordingly, this work investigates to what extent and in which conditions the exploitation of topological information can improve the convergence of the learning process so making it more efficient in terms of network resource consumption.

More specifically, a gossiping-based distributed learning protocol which exploits the centrality information to reduce the consumption of network resources is introduced. The proposed *centrality-aware gossiping-based learning protocol* (CGL) is assessed considering different centrality measures and machine learning techniques.

2.4.1 Centrality in WSN

Centrality characterizes the topological *importance* of a node in a network and its relationship with its neighbouring nodes. More specifically, it measures how a node impacts on other network nodes. In this section, we illustrate some of the most well known centrality metrics typically proposed in WSNs [47],[48].

The most commonly used centrality metrics are:

- **Betweennes Centrality**: It assesses the number of times a node lies on the shortest path between other pairs of nodes. It is calculated by identifying all the shortest paths in a network and then counting how many times each node falls on one of these shortest paths. Therefore the node with the highest betweennes centrality measure is the node that is located in the maximum number of shortest paths in the network. Hence, more information passes through the node with the maximum betweenness. Betweennes centrality is largely used for analysing communication dynamics and plays a crucial role in characterizing the network connectivity.

Formally, betweenness centrality of a node x, denoted as $CE_b(x)$, can be calculated as:

$$CE_b(x) = \sum_{s \neq x \neq t} \frac{\tau_{st}(x)}{\tau_{st}} \tag{2.11}$$

where τ_{st} is the total number of shortest paths from node s to node t and $\tau_{st}(x)$ is the number of paths that pass through the node x.

- **Eigen Centrality**: It assesses a node impact by considering the importance of its one-hop neighbours. More specifically, it measures a node influence based on the number of links it has to the other nodes in the network. Additionally, it also takes how well connected those other nodes are and how many connections they have in the network into account. Thus the eigen centrality of a node is assigned based on its connections and the connections of connections.

  Google Pagerank and Katz centrality [49] are variants of the Eigen centrality.

- **Pagerank Centrality**: As mentioned before, pagerank centrality is a variant of the eigen centrality. Similarly to the eigen centrality, pagerank assigns a score to a node based on its connections and its connections of connections. However, Pagerank also considers the direction of the link and the connection weight [50]. It uses the indegree as the major measure to calculate the influence. Hence Pagerank is used for directed networks. Page rank is the ranking algorithms behind the Google search engine.

- **Degree centrality**: Degree centrality is the simplest centrality measure that assesses the number of links held by each node in the network. More specifically, it denotes the one-hop connections each node has with its neighbouring nodes. Degree centrality is often used for selecting a cluster head and/or the sink node in a network. The degree centrality is calculated as:

$$CE_{cd}(x) = \frac{\sum_v e(v,x)}{N-1} \tag{2.12}$$

where N is the number of nodes in the graph and $e(v, x)$ is equal to 1 if there is a direct connection between nodes v and x and 0 otherwise.

- **Closeness Centrality**: The Closeness centrality of a node is the geodesic distance between itself and all other nodes in the network. The closeness centrality score is assigned to a node in the network based on its closeness to other nodes. More specifically, it assesses how short the shortest paths are between the node and all other nodes.

  The closeness centrality of a node x is calculated as follows:

  $$CE_c(x) = \frac{N}{\sum_v d(v, x)} \tag{2.13}$$

  where N is the number of nodes in the graph and d(v,x) is the distance between nodes v and x.

Centrality metrics have been widely studied and applied in designing algorithms for WSNs [48].

Centrality information is used in Cluster Head selection problems in WSN scenarios for enhancing network lifetime. This is crucial in scenarios that exploit clustering to reduce resource consumption. For example, [51] proposes a Cluster Head selection approach based on Degree centrality and Closeness centrality metrics. As compared to approaches that do not exploit centrality metrics, the authors show that the proposed approach performs better in terms of network lifetime. Similarly in [52], the authors propose a Cluster Head selection approach for large scale WSN scenarios using the Closeness centrality measure and they observe significant reduction in energy consumption .

Another critical problem in WSNs is energy efficient routing since it is necessary for prolonging the network life. Centrality metrics have been studied and applied for designing efficient routing algorithms. As an example, in [53] the authors propose a new centrality measure, called *Sink Betweenness*, which they use to support a tree-based routing algorithm. In the above work simulation results show that their routing algorithm improves routing overlap, thus facilitating data fusion. In [54], the authors propose a connectivity-based routing algorithm in which the connectivity factor of a node is evaluated using Betweenness centrality measure. In that work efficient routes are found using fuzzy logic optimization.

Centrality-aware protocols have shown to be energy efficient in various scenarios as mentioned in the previous paragraphs. In our work, we study and design a centrality-aware gossiping protocol for distributed learning which enables achieving quick convergence, thus resulting in significantly increased network lifetime.

In Section 2.4.2, the design objectives of CGL are discussed. Then, in Section 2.4.3, the protocol operations are discussed in detail.

2.4.2 CGL Design objectives

Tthe target of CGL is to support the nodes in collaboratively learning the ML model, $\mathbf{w}$, by exploiting the gossiping mechanism in a WSN. In such a scenario, a given node will transmit its model parameters to its one-hop neighbouring nodes that will update their own model with the received parameters and train it further.

For instance if j is the node that has transmitted its model parameters, $\mathbf{w}_j$, its generic neighbor k will update its model as

$$\mathbf{w}_k = \alpha \cdot \mathbf{w}_j + (1 - \alpha) \cdot \mathbf{w}_k \tag{2.14}$$

where α is a weight parameter the setting of which will be discussed later. Note that node k performs the operation in eq. (2.16) if the model loss obtained by using $\mathbf{w}_j$ is lower than the loss obtained using its current model, i.e., $F_k(\mathbf{w_j}) < F_k(\mathbf{w_k})$. We call this sequence of actions "*communication round*" or "*iteration*", in short.

Another characteristic of CGL is that, each node in the network should exploit its local information only, i.e., the centrality information and the data generated by the node itself. Thus, the operations of CGL should be asynchronous and uncoordinated. Due to the energy limitations in WSNs, the operations of CGL should be as effective as possible, with the objective to reduce the global loss function (see eq. (2.2)) by requiring minimum number of model transmissions.

To achieve the above goals, each node, k, of the network knows and stores its centrality metric and at each iteration schedules the transmission of its model parameters after a time interval, τ_k, with average that is proportional to ratio

$$E\{\tau_k\} = F_k(\mathbf{w}_k)/(B_k \cdot CE_k) \tag{2.15}$$

where CE_K is the centrality of the node and B_k is called *boost factor* and is described in the following.

Furthermore, the communication and model training load must be distributed between all network nodes as fairly as possible. This is necessary to have good fairness among nodes and avoid overloading of a limited number of nodes. To achieve this goal, CGL exploits the *boost factor*. A node k increments its *boost* *factor* whenever it receives the model parameters sent by one of its neighbors, say j. Instead, the *boost factor* is reset to one when the node broadcasts its model parameters in the CGL mechanism. Given that the boost factor is at the denominator of eq. (2.15), the value of τ_k will be lower for nodes that did not transmit their model parameters in the recent past and higher for nodes that transmitted their model parameters recently.

2.4.3 CGL Protocol Details

In this section, more details regarding the operations of CGL are provided and how it works in a toy exemplary WSN scenario is illustrated.

Let us consider a network consisting of four nodes deployed according to a linear topology as shown in Figure 2.15.

Algorithm 2 reports the pseudo-code for the protocol operations executed by a generic node k. In the pseudocode CE_k represents the centrality measure of node k.

At setup, any node initializes the boost factor $B_k = 1$. Furthermore, the model is trained using the local data X_k. The training operation is executed for a given number of epochs, at each node separately. Epochs denote the number of times the machine learning algorithm will work through the entire dataset during the training.

Also at setup, each CGL node is initialized by calculating its centrality measure $CE(k)$. To this purpose, observe that several distributed algorithms exist that allow such calculation efficiently (see, for example, [55–58]).

At the end of the initialization phase which is illustrated in lines 1-5 of Algorithm 2, the CGL node k will have a local model with parameters $\mathbf{w}_k$ and an estimate of its current loss $F_k(\mathbf{w}_k)$. Using these initial parameters the node sets a timeout at time $t_k^{(\text{next})} = t + T \cdot F_k(\mathbf{w}_k)/(B_k \cdot CE_k)$, where t represents the current time and T is a constant value which is set in such a way that protocol operations are sparse in time[3].

For example in Figure 2.15, the loss values for the four nodes at the end of the initialization phase are $F_1(\mathbf{w}_1) = 226.73$, $F_2(\mathbf{w}_2) = 92.11$, $F_3(\mathbf{w}_3) = 76.34$, and $F_4(\mathbf{w}_4) = 70.42$, respectively.

Let us consider the Degree centrality for this example. Therefore the centrality values for the four nodes are $CE_1 = 1$, $CE_2 = 2$, $CE_3 = 2$, and $CE_4 = 1$. Consequently, the four nodes will set their timeouts at times $t_1^{(\text{next})} = 226.73 \cdot T$,

[3]In our experiment, we have set T equal to a constant proportional to the average time required to train the model local data.

$t_2^{(\text{next})} = 46.05 \cdot T$, $t_3^{(\text{next})} = 38.17 \cdot T$, and $t_4^{(\text{next})} = 70.42 \cdot T$, respectively.

Note that Figure 2.15 was obtained assuming that the setup time is $t = 0$ for all nodes.

When the initialization phase is completed, CGL executes the regular operations, sketched in lines 7-27 of the Algorithm 2. More specifically,

1. when the timeout $t_k^{(\text{next})}$ elapses, the node executes the operations sketched in lines 9-15 of Algorithm 2;

2. when a given node receives the model parameters, $\mathbf{w}_j$, from the neighboring node j, it executes the operations sketched in lines 17-25 of Algorithm 2, including retraining the model for a given number of epochs.

As reported in lines 9-15 of Algorithm 2, when the timeout elapses, the node will broadcast its model parameters $\mathbf{w}_j$ to its neighbouring nodes. In the example shown in Figure 2.15, node 3 has the lowest value of the timeout parameter, i.e. $t_3^{(\text{next})} = 38.17 \cdot T$.

Therefore, as reported in line 11, it will broadcast a message containing its model parameters $\mathbf{w}_3$. Note that the time at which a node disseminates its model parameters depends on the timeout value. As an example, the first dissemination performed by node 3 occurs at time $t = 38.17 \cdot T$. The disseminated message will be received by nodes 2 and 4 which will execute the operations sketched in lines 17-25 of Algorithm 2.

Note that the nodes 2 and 4 first increase the boost factor by one, evaluate the loss they would achieve by exploiting the model $\mathbf{w}_3$ (i.e. $F_2(\mathbf{w}_3)$ and $F_4(\mathbf{w}_3)$). If the estimated value of loss is higher than the current one, no other actions will be executed and the node will wait for a new event. Instead, if the estimated value of loss, $F_k(\mathbf{w}_j)$, is lower than or equal to the current one, $F_k(\mathbf{w}_k)$, the node executes the operations in lines 21-24. Thus, node 2 and 4 update their model parameters $\mathbf{w}_2$ and $\mathbf{w}_4$ as $\mathbf{w}_2 = \alpha \cdot \mathbf{w}_3 + (1-\alpha) \cdot \mathbf{w}_2$ and $\mathbf{w}_4 = \alpha \cdot \mathbf{w}_3 + (1-\alpha) \cdot \mathbf{w}_4$ and, then, train the model using their local data and evaluate new loss values. Finally, they update the $t_k^{(\text{next})}$ value.

In the example shown in Figure 2.15, the resulting value of $F_2(\mathbf{w}_2)$ at time $t = 38.17 \cdot T$ is 86.12. Since $CE_2 = 2$ and $B_2 = 2$, the timeout is set at time $t_2^{(\text{next})} = 59.7 \cdot T$. Similarly, the resulting value of $F_4(\mathbf{w}_4)$ at the same time is 54.11. Since $CE_4 = 1$ and $B_4 = 2$, the timeout is set at time $t_4^{(\text{next})} = 65.22 \cdot T$.

In the next round, i.e., at time $t = 59.7 \cdot T$, node 2 broadcasts its model parameters to nodes 1 and 3.

Then it will be the turn of node 3 that will transmit its new model parameters, $\mathbf{w}_3$.

In just three transmissions of model parameters, the values of the loss in the four nodes become $F_1(\mathbf{w}_1) = 90.84$, $F_2(\mathbf{w}_2) = 79.74$, $F_3(\mathbf{w}_3) = 44.33$, and $F_4(\mathbf{w}_4) = 36.44$. Such values are much lower than the initial ones and, as a result, the average loss, which was equal to (226.73+92.11+76.34+70.42)/4=116.4 at time 0, rapidly decreases to (90.84+79.74+44.33+36.44)/4=62.81.

Algorithm 2 CGL protocol

```
1: /* Protocol initialization */
2: Initialize $B_k$=1
3: Calculate Centrality metrics $CE_k$
4: Initialize $\mathbf{w}_k$=train_model($\mathbb{X}_k$, RND)
5: $t_k^{(\text{next})} = t + T \cdot F_k(\mathbf{w}_k)/(B_k \cdot CE_k)$
6: /* Regular operations */
7: while Protocol initialization do
8:     Wait for Event
9:     if Event.Type==time-out then
10:        /* Node k will send its model parameters $\mathbf{w}_k$ */
11:        Broadcast($\mathbf{w}_k, F_k(\mathbf{w}_k)$)
12:        Wait for the updated loss values
13:        $B_k = 1$
14:        $t_k^{(\text{next})} = t + T \cdot F_k(\mathbf{w}_k)/(B_k \cdot CE_k)$
15:    end if
16:    if Event.Type == received $\mathbf{w}_j$ then
17:        /* Node k is receiving the model parameters $\mathbf{w}_j$ from neighbor node j */
18:        $B_k = B_k + 1$
19:        Evaluate $F_k(\mathbf{w}_j)$
20:        if $F_k(\mathbf{w}_j) \leq F_k(\mathbf{w}_k)$ then
21:            $\mathbf{w}_k = \alpha \cdot \mathbf{w}_j + (1 - \alpha) \cdot \mathbf{w}_k$
22:            $\mathbf{w}_k$ = train_model($\mathbb{X}_k, \mathbf{w}_k$)
23:            Evaluate $F_k(\mathbf{w}_k)$
24:            $t_k^{(\text{next})} = t + T \cdot F_k(\mathbf{w}_k)/(B_k \cdot CE_k)$
25:        end if
26:    end if
27: end while
```

2.4.4 ML Algorithms

In this section, we apply CGL protocol to the same WSN scenario considered in section 2.3.3.

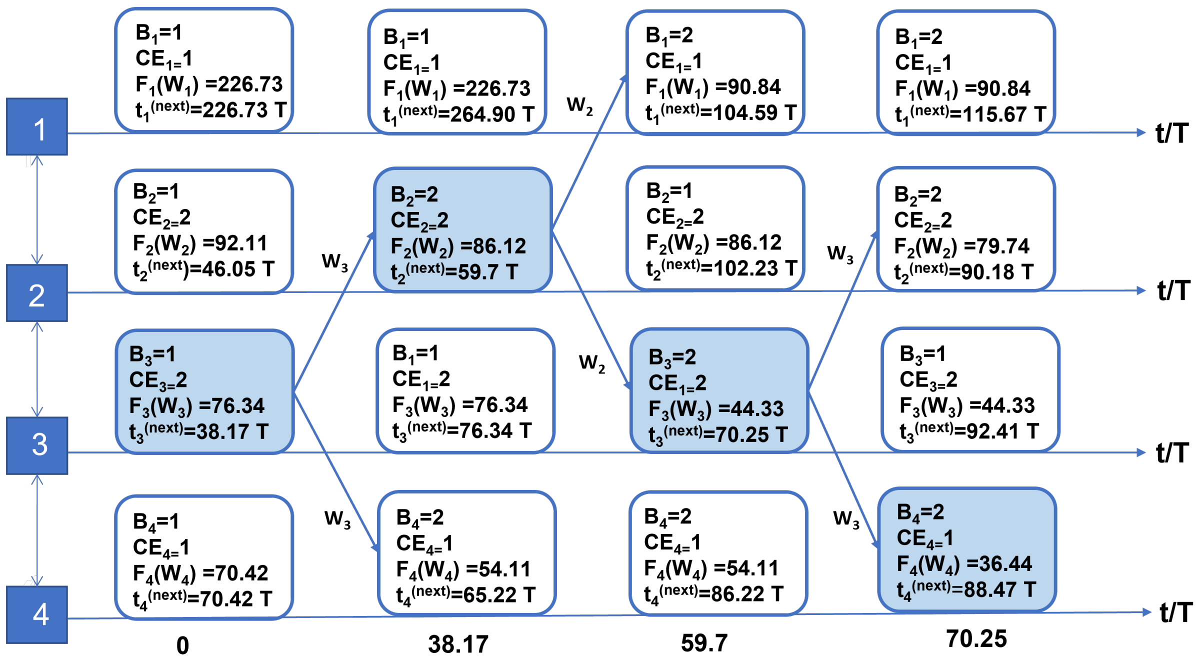


Figure 2.15: CGL protocol in action

CGL can be applied whatever is the ML approach utilized. For our experiments we consider two cases with different machine learning approaches. In Case 1, we assume that each sensor node executes an autoencoder [42] which represents a traditional methodology for dimension reduction, denoising, and anomaly detection [43]. In Case 2, we assume each sensor node executes a LSTM (Long Short-Term Memory) neural network for time series prediction.

For the centrality metrics, we considered betweenness, closeness, pagerank, degree and eigen centrality metrics.

Autoencoders

Concerning the ML approach, in our experiments we have employed the simplest type of autoencoder, the - so called - *Vanilla autoencoder* which was described in the section 2.3.3.

LSTM Network

The LSTM neural networks are a type of *recurrent neural network* (RNN) approach that is trained through the back-propogation mechanism over time. LSTM neural networks are applied for forecasting tasks [59],[60],[61].

The key difference between LSTM and other neural networks is that LSTM do not have neurons, but they have memory blocks that are connected through

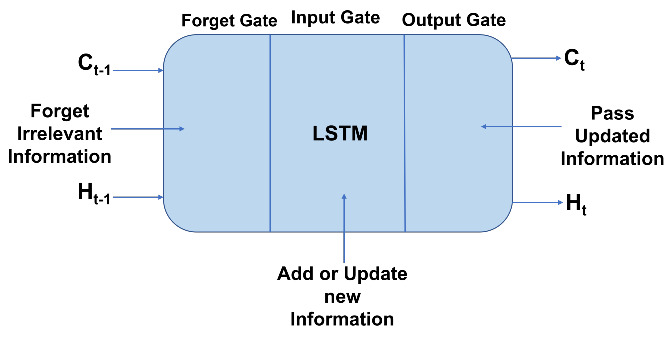


Figure 2.16: Functioning of LSTM network

layers. Each block has gates that take care of the block state and input. The three types of gate within a block are *forget gate*, *input gate* and *output gate* as shown in figure 2.16. The forget gate decides which information should be discarded. More specifically, it decides whether the information from the previous timestamp is to be remembered or to be forgotten. The input gate decides the input information to update the memory state. The output gate decides the output based on the input and memory of the block. Each gate in a block has a sigmoid function to control if they are activated or not.

LSTM has a hidden state where $H_{(t-1)}$ indicates the hidden state of the previous timestamp amd H_t denotes the hidden state of the current time stamp. Addition to the hidden state, LSTM also has a cell state denoted by C_t and $C_{(t-1)}$ for current and previous stamp respectively. Note that cell state is known as long term memory and hidden state is known as short term memory.

In the case study, the LSTM is utilised to predict the next hour PM1 values using the regression approach. More specifically, the PM1 values from the dataset are used to create two columns of data: the first column containing the PM1 values (t) and the second column containing the next hour PM1 values (t+1), to be predicted. Thus the LSTM model trains on the two columns to map a function for predicting the next hour PM1 values.

2.4.5 CGL: Numerical Results

In this section, the numerical results obtained by applying the CGL protocol in the scenarios described in previous Section 2.4.4 are discussed. We conducted a simulation campaign to observe the average loss of the sensor nodes obtained using different centrality metrics in both the cases, i.e. autoencoder and LSTM.

More specifically, 10 simulations were executed in both the cases for different values of α (i.e. α =0.5, 0.7, 0.9 and 1), different number of *epochs* (i.e., 10, 15 and 20) and different centrality metrics. Epochs denote the number of times the machine learning model will work through the entire dataset during the training.

In figure 2.17, the average loss obtained for different values of α in both Auto-encoders and LSTM network cases is reported. Note that α is the weight parameter in the eq. 2.16. As seen in the figures, we observed that with higher values of α, the convergence is quicker in both the cases.

Similarly in figure 2.18, the average loss obtained for different values of epochs in both the Auto-encoders and LSTM network cases are reported. The observation was that with higher number of epochs in training, the convergence is achieved quickly in both the cases. Note that, the closeness centrality metric was considered for these plots.

In order to select which centrality measure performs better, in Figure 2.19, the average loss obtained with respect to the overall amount of data transmitted by network nodes in the autoencoders case when α is 0.7 and the number of epochs is set to 15 were reported. It was observed that the average loss in case of the Betweenness centrality converges faster when compared to the other centrality metrics. We also observe that the cases with Degree centrality and Page rank centrality had relatively slower convergence rates. Note that each iteration in the CGL protocol corresponds to one packet transmission.

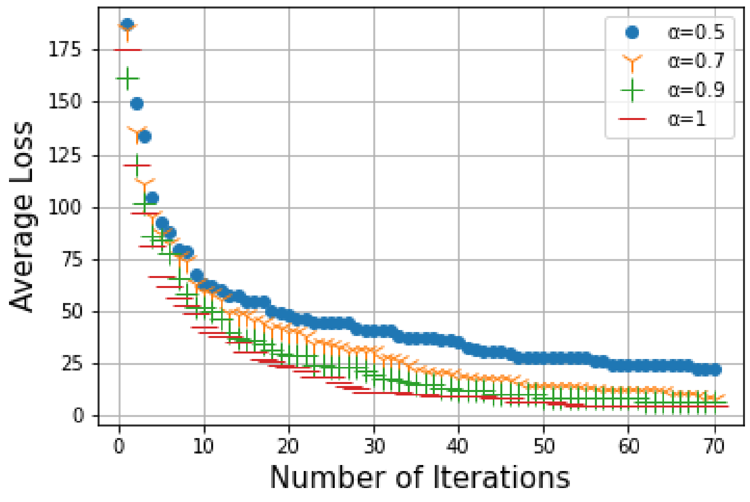
Similarly in Figure 2.20 the average loss obtained with respect to the amount of data trasmitted in the LSTM prediction case when α is 0.7 and the number of epochs is 15 were reported. Similarly to the autoencoder case, it was observed that faster convergence when considering the Betweenness centrality metrics and relatively slower convergence with Degree and Pagerank centrality metrics.

Since considering Betweenness centrality gives better performance in both the cases, in the following we focus on such centrality metric to make a comparison with the case in which distributed learning is based on a gossiping scheme which does not consider the centrality measures of nodes.

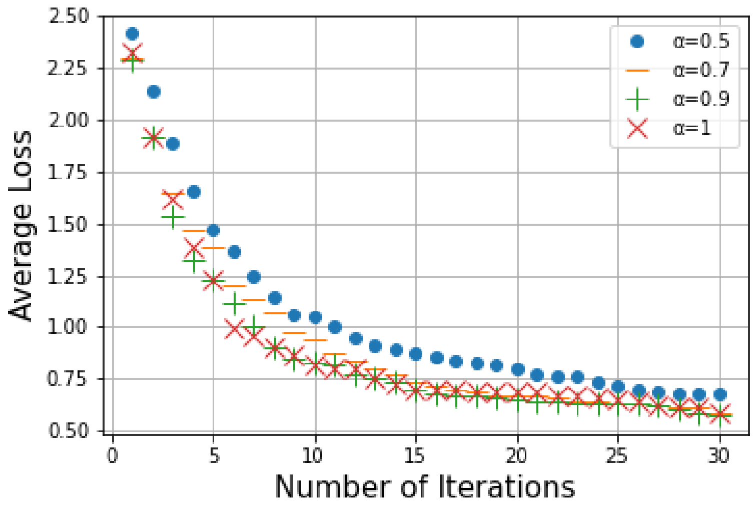
In Figure 2.21, the average loss with respect to the amount of data transmitted in the autoencoder case is reported. It was observed that a significant improvement in performance in the betweeness centrality case when compared to the case without any centrality metrics.

Also, a similar comparison was done in Case 2 and a similar behavior was noticed as shown in Figure 2.22. This assesses that the Betweenness centrality plays an important role in reducing the number of communication rounds required to collaboratively learn a fitting model in a WSN.

Furthermore, in Figure 2.23 we illustrate the number of times a sensor node

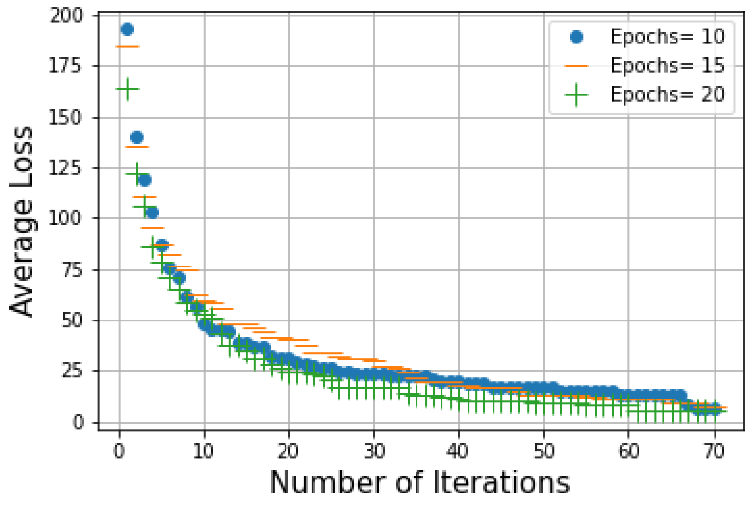


(a) Auto-encoders

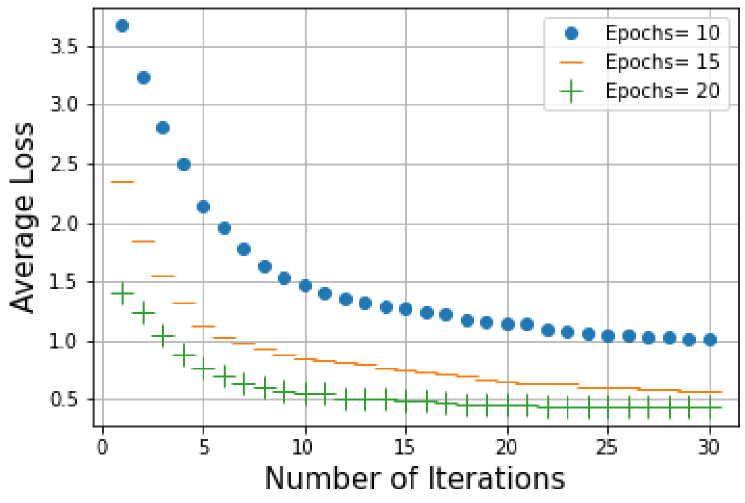


(b) (LSTM network)

Figure 2.17: Average Loss vs Number of iterations for different values of α



(a) Auto-encoders



(b) (LSTM network)

Figure 2.18: Average Loss vs Number of iterations for different values of Epochs

has been trained using CGL in Case 1, estimated until the loss converges and stabilizes. In the figure, nodes are sorted out in decreasing order of number of training cycles (i.e., not based on their IDs). We observed that the majority of nodes were trained approximately 20 times or more; on the other hand, few sets of nodes were trained less than 10 times. This implies that the training event depends on how influential a node is in the network.

Similar observation was made in the case of LSTM-prediction (Case 2), as shown in Figure 2.24. In this case, the majority of nodes were trained 8 times or more, whereas a set of 7 nodes were trained 5 times or less. Similarly to case 1, nodes

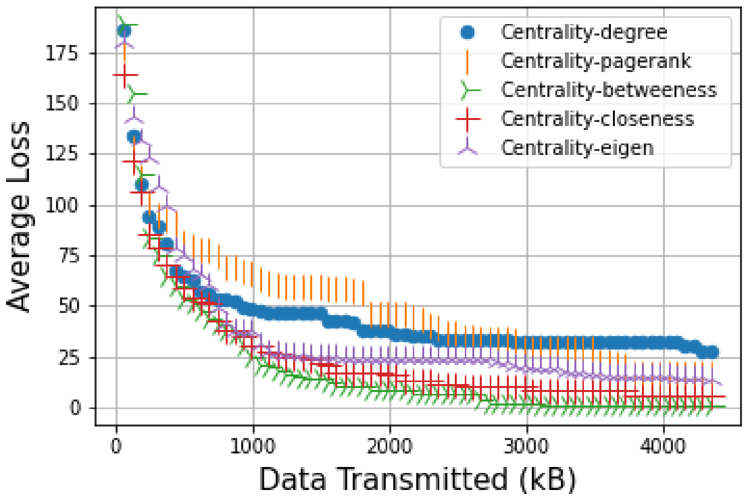


Figure 2.19: Average Loss vs. Data Transmitted for different centrality measures (Autoencoders)

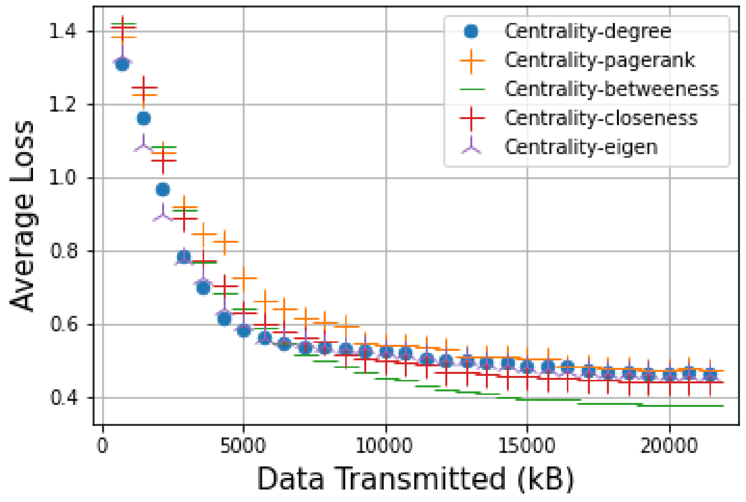


Figure 2.20: Average Loss vs. Data Transmitted for different centrality measures (LSTM network)

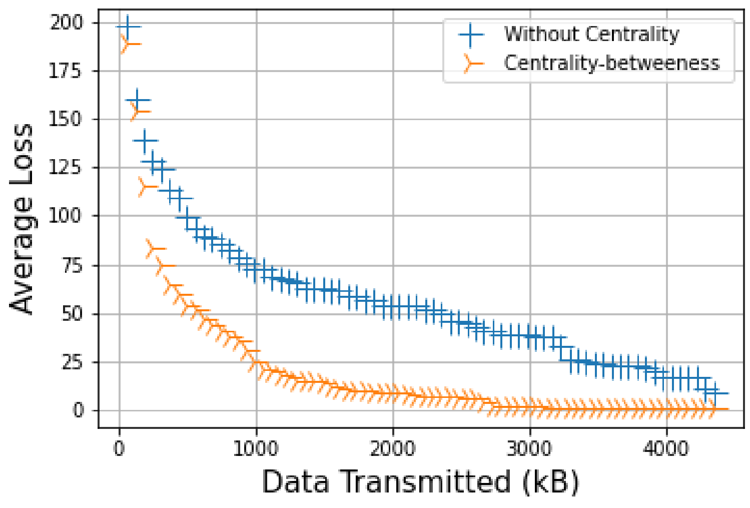


Figure 2.21: Average Loss vs. Data Transmitted (Autoencoders)

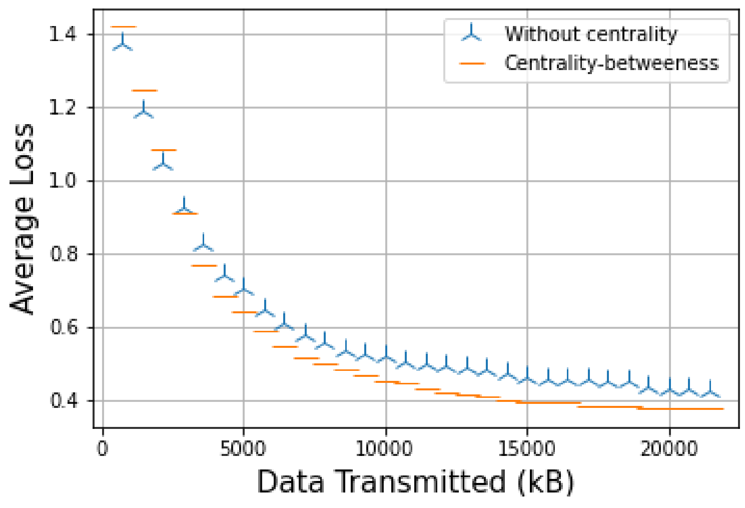


Figure 2.22: Average Loss vs. Data Transmitted (LSTM network)

which were less trained, are those exhibiting lower Betweeness centrality. From both these cases, we notice that the centrality metric of a node influences the training event in both the cases.

Thus, numerical analysis assesses how the centrality metric can significantly impact gossiping and enhance the collaborative learning approach of machine learning models in a WSN scenario.

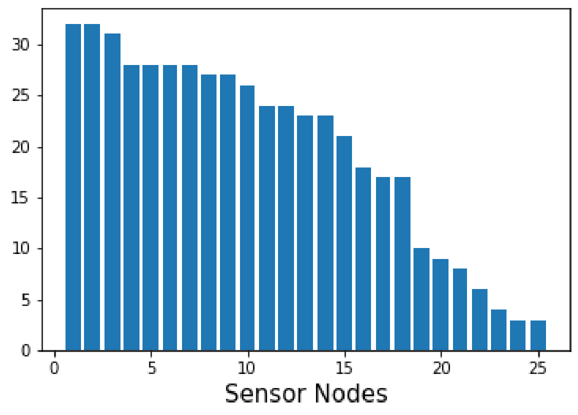


Figure 2.23: Number of times each node has been trained (Case 1: Autoencoder)

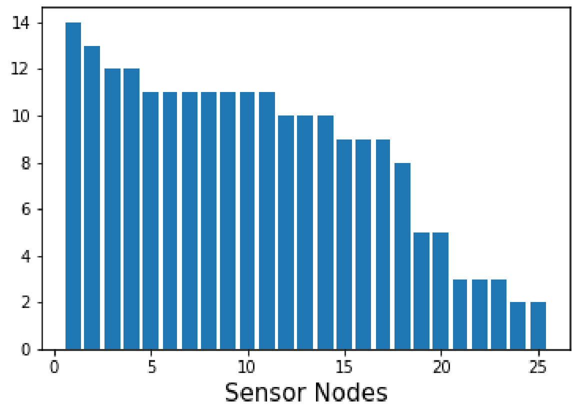


Figure 2.24: Number of times each node has been trained (Case 2: LSTM-Prediction)

2.5 i-WSN: Clustered Distributed learning

The execution of *machine learning* (ML) algorithms into small and low power devices has attracted the attention of researchers and has opened the path to new use cases in the context of wireless sensor networks WSNs. The introduction and success of tools, like Tensorflow Lite[4], represent both the evidence of the interest of the ML community towards such scenarios and a fundamental step towards their realization. However, performing on-device training requires energy, memory, and computing capabilities which are not available in most hardware platforms employed for WSNs.

Therefore, in most current solutions models are trained is some resource rich server outside or at the edge of the WSN. Such approach, however, envisions the transmission of the data available at the WSN nodes needed to train the ML model to the above server, which involves two types of problems:

- such transmissions might require the use of a large amount of communication and energy resources;
- there might be security and privacy issues as the data transmitted by the nodes can be the target of attacks in its way towards the server.

Such issues are addressed and a framework is proposed for the realization of intelligent wireless sensor networks (i-WSN)s which minimizes the exchange of information between WSN nodes. It is assumed that the WSN includes some nodes that have enough resources to execute some ML model training [62]. All network nodes are divided in clusters and in each cluster there is a resource rich node which is in charge for the training of the ML model that will be used by all nodes in the cluster. Such node is, thus, a Cluster Head and executes training by using only the data which is locally available. The resulting model is sent to all nodes in the cluster which will execute it to evaluate some fitness metric. Nodes that obtain a low value of such fitness metric will transmit a part of their data to the Cluster Head which will use it to re-train the model.

Clusters exchange their models in peer-to-peer manner and, thus, cooperate forming a *league*. That is why we call the proposed solution *i-WSN League*.

In Section 2.5.1 we give an overview of the i-WSN League operations. In this context we will also characterize the major features of the hardware platforms that are used for head nodes and common nodes. Then, in Section 2.5.2 we will present the i-WSN League protocol in details.

[4]`https://www.tensorflow.org/lite`

2.5.1 Overview of operations and characteristics of the hardware platforms

As mentioned earlier, a WSN is considered that consists of nodes some of which, referred to as *head nodes*, have computing capabilities sufficient to perform both on-device training and inference and others, referred to as *common nodes*, that can perform inference only due to resource limitations.

Head nodes are equipped with two wireless interfaces. One of them allows connection to a wide area network, e.g., LoRa [63] or IEEE 802.11, the other enables short range communications, e.g., BLE or IEEE 802.15.4. Common nodes are equipped with the short range wireless communication interface only.

Nodes will be divided into Clusters, each of which has a *Cluster Head* and several *Cluster Members*. Head nodes can be Cluster Heads, common nodes are always Cluster Members. Cluster Heads can communicate using both the wireless interfaces, Cluster Members can communicate using the short range wireless communication interface, only. Thus, head nodes that are Cluster Members will maintain their wide area network interface turned off.

Cluster Heads use their long range wireless communication interface to create a mesh network, which we call *CH-network*.

Cluster Heads execute model training using their own data. The *fitness* of the model will be evaluated through an appropriate *loss* function[5]. Each Cluster Head broadcasts its loss function throughout the CH-network after it calculates it at the end of every training execution. The Cluster Head that attains the lowest loss will transmit its model parameters to its one-hop neighbors in the CH-network using the long range wireless interface.

The generic k-th Cluster Head upon receiving the model by the j-th Cluster Head will use the received parameters, $\mathbf{w}_j$, to update its own model, $\mathbf{w}_k$, as follows

$$\mathbf{w}_k = \alpha \cdot \mathbf{w}_j + (1 - \alpha) \cdot \mathbf{w}_k \tag{2.16}$$

where α is a weight parameter that we set larger than 0.5 as detailed later. After updating their models, the Cluster Heads will train the model obtained applying eq. (2.16) using the data they have available locally.

For what concerns Cluster Members, observe that in most cases ML models are too complex to be used for inference by common nodes. Therefore, the generic j-th Cluster Heads will create a compressed version of its model, $\widehat{\mathbf{w}}_j$, in such a way

[5]Several loss functions have been proposed for different application scenarios. Interested reader can refer to the overview provided in [64].

that it can be used for inference by common nodes as well. How the compressed model, $\widehat{\mathbf{w}}_j$ can be used for inference by common nodes is outside the scope of this paper as it depends on the tool utilized for model compression.

The Cluster Head will broadcast the obtained model along with the corresponding value of loss to all members of its Cluster. Cluster Members will not train the received model as they do not have the capabilities required for training. Instead, the Cluster Members will evaluate the loss they achieve with the updated model using their own data. If the difference between the loss obtained by the Cluster Member and the broadcasting Cluster Head is greater than a certain threshold, that particular Cluster Member will send its data to its Cluster Head in order to train and transmit the updated model back.

The above sequence of actions are denoted as *protocol iteration* or *iteration*, in short.

Given the energy limitations characterizing WSNs, i-WSN League operations must be as effective as possible and thus model transmissions and the consequent training iterations should be executed when the resulting expected reduction in the overall loss is significant. To this goal, at the end of each protocol iteration, each involved Cluster Head will broadcast its updated loss value throughout the CH-network. This value will be used by each Cluster Head to compare the fitness of its own model to the fitness of the other Cluster Heads.

Furthermore, the communication and computing load must be distributed between all Clusters as fairly as possible. This is necessary to avoid that a few Cluster Heads are overloaded. In fact, even if Cluster Heads are resource rich platforms, they are powered by batteries and processing and communication overload of a few of them might result in the exhaustion of their batteries, so reducing the lifetime of the entire network. To achieve this goal, i-WSN League exploits a parameter called *boost factor* which is updated in such a way that it is expected to be high for Cluster Heads that did not transmit their model parameters in the recent past and low for the Cluster Heads that, instead, did transmit their model parameters recently. Further details will be provided in the following Section 2.5.2.

2.5.2 i-WSN Protocol Details

In this section, the details of the i-WSN League protocol are presented and, for the sake of clarity, its operations are discussed in a the simple scenario depicted in Figure 2.25. This comprises 8 common nodes grouped into 4 Clusters. Nodes 1,

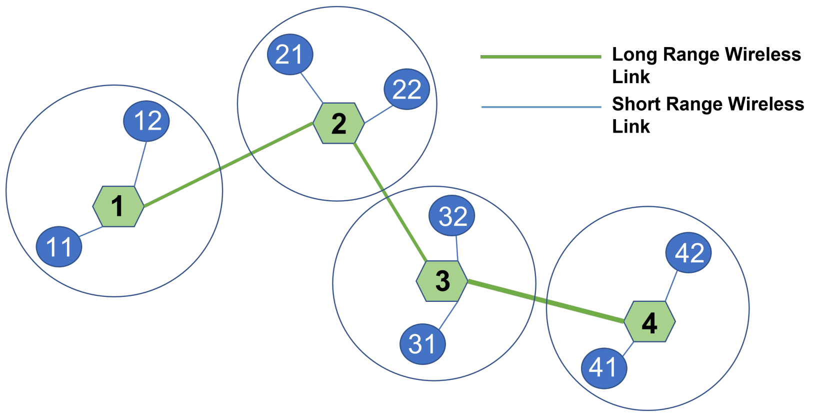


Figure 2.25: Clustering Scenario.

2, 3 and 4 are Cluster Heads[6] equipped with two wireless interfaces. Accordingly, the CH-network consists of 4 nodes connected according to a linear topology. All the other nodes are Cluster Members, each of which belongs to one Cluster only. Observe, that communication in each Cluster can happen in a multi-hop manner, when needed.

Algorithm 3 represents the pseudocode for the protocol run by the generic Cluster Head CH. The resulting actions executed by the Cluster Heads are represented in Figure 2.26.

At the startup sketched in lines 1-6 of Algorithm 3, Cluster Head CH initializes the boost factor B_{CH} to 1. Furthermore, the model is trained using the local data X_{CH} and starting from random initial conditions, RND. The training operation is executed for a given number of epochs. The *scaled loss* parameter, denoted as, SL_{CH} is calculated from the training loss and the boost factor as in eq. (2.17),i.e.

$$SL_{CH} = F_{CH}(\mathbf{w}_{CH})/B_{CH} \tag{2.17}$$

where $F_{CH}(\mathbf{w}_{CH})$ is the loss of the Cluster Head, CH, and B_{CH} is its current boost factor. Observe that the scaled loss is supposed to decrease for Cluster heads that have fitting models and did not transmit their model parameters recently. For example, in Figure 2.26, the loss values for the four Cluster Heads at the end of the initialization phase are $F_1(\mathbf{w}_1) = 69.11$, $F_2(\mathbf{w}_2) = 26.72$, $F_3(\mathbf{w}_3) = 66.32$, and $F_4(\mathbf{w}_4) = 98.75$. Therefore, given that the boost factors for all them is equal to 1, the corresponding scaled losses are $SL_1(\mathbf{w}_1) = 69.11$, $SL_2(\mathbf{w}_2) = 26.72$, $SL_3(\mathbf{w}_3) = 66.32$, and $SL_4(\mathbf{w}_4) = 98.75$.

At the end of the initialization phase, the Cluster Heads broadcasts their model

[6]the Cluster is denoted with the identifier of the corresponding Cluster Head, therefore, if say Cluster 1 is mentioned, the Cluster for which node 1 is the Cluster Head is meant

parameters to their own Cluster Members. Also, the Cluster Heads broadcast their scaled loss in the Cluster Head network. In this way Cluster Heads identify of them has the lowest scaled loss and thus will broadcast its model parameters.

When the initialization phase is completed, Cluster Heads execute the regular operations sketched in lines 7-30 of the Algorithm 3, which are event based. More specifically, three types of events can occur:

1. **Broadcast**: In this event, the Cluster Head broadcasts its model parameters, i.e, the Cluster Head with the lowest scaled loss broadcasts its model parameters to its Cluster Members and its neighboring Cluster Heads. In this case, the node executes the operations sketched in lines 10-18 of Algorithm 3.

2. **Receive Model Parameters**: In this event, the Cluster Head receives the model parameters $\mathbf{w}_k$. When the receive, the Cluster Head executes the operations sketched in lines 19-27 of Algorithm 3, including retraining of the model for a given number of epochs.

3. **Receive Data Chunk**: In this event, the Cluster Head receives a chunk of data from one of its Cluster Member for retraining. The Cluster Head will execute the operations sketched in lines 28-31 of Algorithm 3.

At the end of the initialization phase t_1 and at the end of each iterations, each Cluster Head broadcasts its scaled loss to all other Cluster Heads. Thus, each Cluster Head has a the scaled losses of other Cluster Heads to make a comparison and realize if it has the lowest scaled loss. If this is the case, it will perform the operations reported in lines 10-18 of the Algorithm 3. The Cluster Head will broadcast the model parameters $\mathbf{w}_{CH}$ to the one hop neighbours in the CH-network and compresses the model to broadcast the compressed model parameters $\widehat{\mathbf{w}}_{CH}$ to its own Cluster Members. In our experiments, the Tensor-flow model is compressed into a Tensor-flow lite model.

In the case shown in Figure 2.26, at the end of the initialization phase, Cluster Head 2 has the lowest scaled loss which is 26.72. Hence it broadcast its model parameters, $\mathbf{w}_2$ to its neighboring Cluster Heads 1 and 3, then it compresses its model and transmits the resulting compressed parameters, $\widehat{\mathbf{w}}_2$, along with the value of the loss, $F_{CH}(\widehat{\mathbf{w}}_{CH})$, to its Cluster Members 21 and 22. The operations executed by the Cluster Members when they receive the compressed model parameters are sketched in Algorithm 4 and are explained later in this section. After it broadcasts its model parameters and the loss value, CH resets the boost factor B_{CH} to 1.

Lines 19-27 of the Algorithm 3 show the functions of the Cluster Head when it receives the model parameters from another Cluster Head through the CH-network. More specifically, the boost factor is multiplied by two and the weights of their models are updated according to the eq. (2.16). Finally, the resulting model is trained using the data locally available. In our exemplary case, Cluster Heads 1 and 3 have received the model by Cluster Head 2, therefore they train the updated model using their data. At the end of the training their losses become 64.70 and 22.10, respectively. Both their boost factors become equal to 2 and the scaled losses is evaluated accordingly.

Similarly to the first round, the Cluster Head with the lowest scaled loss is identified. Therefore, the Cluster Head 3, which has SL_3=11.05 broadcasts its model parameters.

As it will be explained later, it might happen that the Cluster Head receives a chunk of data by one of its Cluster Members, say j. As reported in lines 28-31 of Algorithm 3, where $\mathbb{DA}_j$ represents the received chunk of data, in this case the Cluster Head trains the model using the received data along with the rest of the data locally available. Then, the new model is sent back to the Cluster Member j.

Algorithm 4 illustrates the functions of the generic Cluster Member when it receives the model parameters from its Cluster Head. Before updating its model, the node evaluates the loss which it would obtain using the model $\widehat{\mathbf{w}}_{CH}$ transmitted by the Cluster Head, $F_j(\widehat{\mathbf{w}}_{CH})$ and compares it to the loss value sent by CH, $F_{CH}(\widehat{\mathbf{w}}_{CH})$. If the the difference between the two is larger than a given threshold σ_{TH}, it means that the dataset available at the Cluster Head is not representative of the data available at the Cluster Member. Accordingly, the Cluster Member sends a chunk of its data to the Cluster Head. As explained earlier the Cluster Head will use such data for training its model. This is the reason why there is a change in the loss in the Cluster Head at second iteration though it has not received model parameters (point A in figure 2.27). Since the Cluster Head has received the data chunk and trained on it, there is a change in the loss from 98.75 to 80.33 in figure 2.27 where we detail what happens in Figure 2.26 at the second iteration in Cluster Head 4.

2.5.3 i-WSN Performance evaluation

In this section, the performance of i-WSN League is assesed by analyzing its behavior in the same case study considered in the previous sections.

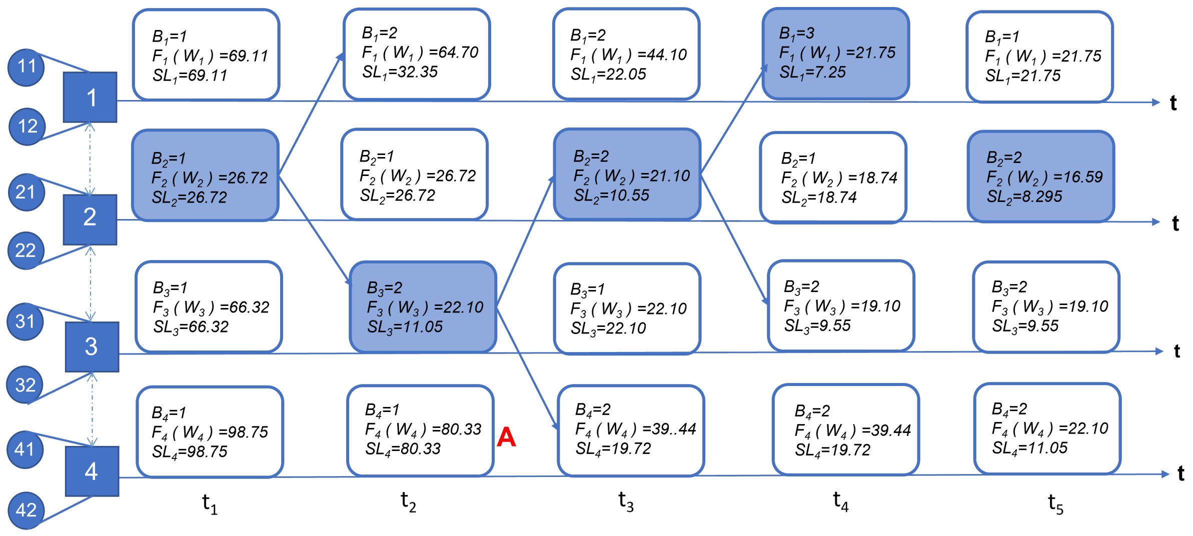
Accordingly, in the following section, i.e., Section 2.5.4, the scenario and the

Algorithm 3 i-WSN protocol-**Cluster Head**

```
1: /* Protocol initialization */
2: Initialize $B_{CH}$=1
3: Initialize $\mathbf{w}_{CH}$=train_model($\mathbb{X}_{CH}$, RND)
4: Calculate $SL_{CH} = F_{CH}(\mathbf{w}_{CH})/B_{CH}$
5: Broadcast($\mathbf{w}_{CH}$) to $CM$
6: Broadcast($\mathbf{SL}_{CH}$) in $CH$ network.
7: /* Regular operations */
8: while TRUE do
9:     Wait for Event
10:    if Event.Type==Broadcast then
11:        /* Cluster-Head $CH$ will send its model parameters $\mathbf{w}_{CH}$ */
12:        Broadcast($\mathbf{w}_{CH}$) in $CH$ network.
13:        Reset $B_{CH} = 1$
14:        Calculate $SL_{CH} = F_{CH}(\mathbf{w}_{CH})/B_{CH}$
15:        Broadcast($\mathbf{SL}_{CH}$) in $CH$ network.
16:        Compress the model
17:        Broadcast the $model_{lite}$ parameters $\widehat{\mathbf{w}}_{CH}$
18:    end if
19:    if Event.Type == Receive Model Parameters $\mathbf{w}_K$ then
20:        /* Node $CH$ is receiving the model parameters $\mathbf{w}_{CH}$ from neighbor
   Cluster-head $K$ */
21:        $B_{CH} = B_{CH} * 2$
22:        Calculate $SL_{CH} = F_{CH}(\mathbf{w}_{CH})/B_{CH}$
23:        $\mathbf{w}_{CH} = \alpha \cdot \mathbf{w}_K + (1-\alpha) \cdot \mathbf{w}_{CH}$
24:        $\mathbf{w}_{CH}$ = train_model($\mathbb{X}_{CH}, \mathbf{w}_{CH}$)
25:        Broadcast($\mathbf{w}_{CH}$) to $CM$
26:        Broadcast($\mathbf{SL}_{CH}$) in $CH$ network.
27:    end if
28:    if Event.Type == Receive Data Chunk $\mathbb{DA}_j$ then
29:        $\mathbf{w}_{CH}$ = train_model($\mathbb{DA}_j, \mathbf{w}_K$)
30:        Broadcast($\mathbf{w}_{CH}$) to $j$
31:    end if
32: end while
```

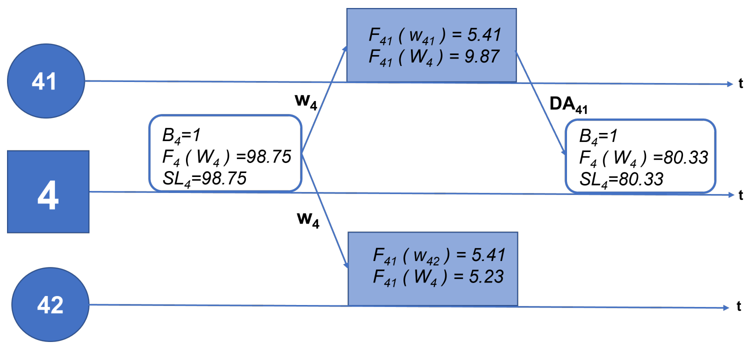
Algorithm 4 i-WSN protocol-**Cluster Member**

```
1: Wait for Event
2: if Event.Type == received $\widehat{\mathbf{w}}_{CH}$ then
3:     $\mathbf{w}_j = \alpha \cdot \widehat{\mathbf{w}}_{CH} + (1-\alpha) \cdot \mathbf{w}_j$
4:     Evaluate $F_j(\mathbf{w}_j)$ and $F_j(\widehat{\mathbf{w}}_{CH})$
5:     if $F_j(\mathbf{w}_j) - F_j(\widehat{\mathbf{w}}_{CH}) \geq TH$ then
6:         /* Sends data to Cluster Head for training */
7:         Transmit $\mathbb{DA}_j$ to $CH$
8:     end if
9: end if
```



11
12
21
22
31
32
41
42
1
2
3
4
B1=1 F1(W1)=69.11 SL1=69.11
B1=2 F1(W1)=64.70 SL1=32.35
B1=2 F1(W1)=44.10 SL1=22.05
B1=3 F1(W1)=21.75 SL1=7.25
B1=1 F1(W1)=21.75 SL1=21.75
B2=1 F2(W2)=26.72 SL2=26.72
B2=1 F2(W2)=26.72 SL2=26.72
B2=2 F2(W2)=21.10 SL2=10.55
B2=1 F2(W2)=18.74 SL2=18.74
B2=2 F2(W2)=16.59 SL2=8.295
B3=1 F3(W3)=66.32 SL3=66.32
B3=2 F3(W3)=22.10 SL3=11.05
B3=1 F3(W3)=22.10 SL3=22.10
B3=2 F3(W3)=19.10 SL3=9.55
B3=2 F3(W3)=19.10 SL3=9.55
B4=1 F4(W4)=98.75 SL4=98.75
B4=1 F4(W4)=80.33 SL4=80.33
A
B4=2 F4(W4)=39..44 SL4=19.72
B4=2 F4(W4)=39.44 SL4=19.72
B4=2 F4(W4)=22.10 SL4=11.05
t
t1
t2
t3
t4
t5

Figure 2.26: i-WSN League protocol in action (Cluster Heads)



41
4
42
F41(W41) = 5.41 F41(W4) = 9.87
W4
B4=1 F4(W4)=98.75 SL4=98.75
DA41
B4=1 F4(W4)=80.33 SL4=80.33
W4
F41(W42) = 5.41 F41(W4) = 5.23
t

Figure 2.27: i-WSN League protocol in action (inside Cluster 4)

data-set considered in our experiments are discussed. Then, in Section 2.5.5, we present and discuss the numerical results.

2.5.4 i-WSN Scenario

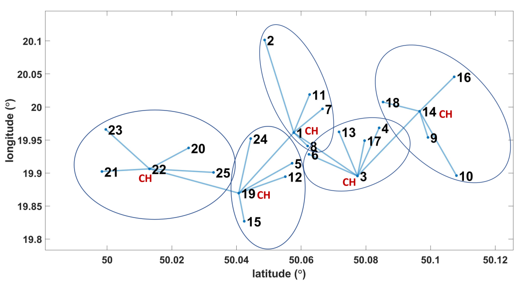
The same data-set and the topology in section 2.3.3 is considered here as well. For the training in Cluster Heads, we consider the data from the first 25 days and we consider the data from the last 5 days for the inference. Similarly, in case of the Cluster Members, the data from the first 25 days is split in chunks, while the last 5 days data is utilised for the inference.

For our experiments, we consider four cases with different clustering strategies which result in the configurations shown in Figure 2.28:

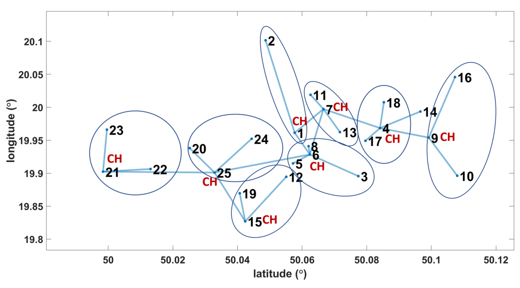
- **Case 1**: The network in this case has 5 Cluster Heads with equal number of cluster Members. Therefore each cluster Head has four Cluster Members. The Cluster Heads are 1,3,14,19 and 22.

- **Case 2**: The network in this case has 8 Cluster Heads. Therefore, the sizes of the clusters are small in this case. The Cluster Heads are 1,4,6,7,9,15,21 and 25.
- **Case 3**: The clustering in this case is similar to Case 1. There are 5 Cluster Heads in the network. But the Cluster Members in each cluster vary and they are not the same as in case 1. The Cluster Heads are 1,10,15,18 and 22.
- **Case 4**: In this case, the network is divided into two large clusters only and the Cluster Heads are 4 and 20.

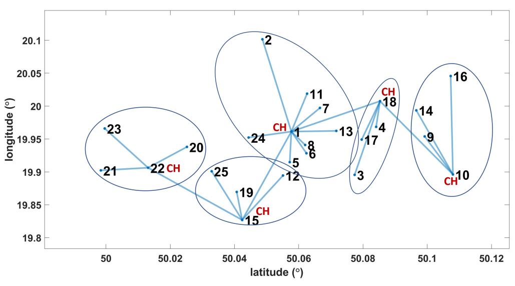
In all the above cases, there are no nodes that belongs to more than one cluster.



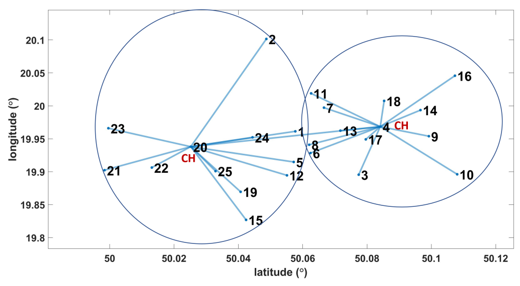
(a) Clustering topology 1



(b) Clustering topology 2



(c) Clustering topology 3



(d) Clustering topology 4

Figure 2.28: Clustering cases considered in the experiments

i-WSN League can be applied whatever is the ML approach utilized. Nevertheless, in our experiments we assume that each Cluster Head executes an autoencoder [42] which is highly utilized for anomaly detection [43]. And all the nodes can perform inference with a trained model.

We utilize the *Mean Square Error* (MSE) to estimate the reconstruction accuracy [43], i.e. $\psi(\mathbf{X}, \mathbf{X}') = MSE(\mathbf{X}, \mathbf{X}')$.

We focus on the values of temperature, humidity, PM1, PM2.5 and PM10, therefore, the input size of the auto-encoder is 5. The intermediate size, which is also known as compressed dimension is equal to 3.

Therefore, the u-th entry in the data-set of the k-th sensor, denoted as $\mathbf{X}_{k,u}$, is:

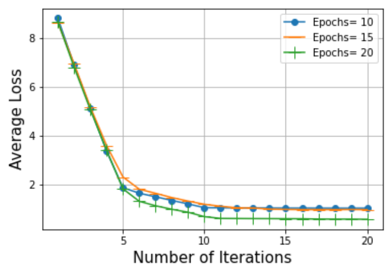
$$\mathbf{X}_{k,u} = (X_{k,u}^{(0)}, X_{k,u}^{(1)}, , ..., X_{k,u}^{(4)}) \tag{2.18}$$

2.5.5 i-WSN Numerical Results

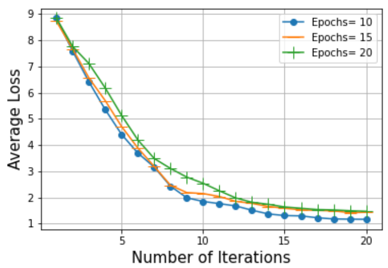
In this section, the numerical results are reported that have been obtained in the scenario described in the previous Section 2.3.3 by applying the i-WSN League approach in all the four cases. A large simulation campaign was conducted to evaluate the relevant performance parameters which we have selected, i.e., the average loss of any node, the number of times a Cluster Head is trained, the number of times Cluster Members have sent their data to their Cluster Heads.

More specifically, 10 simulations in each case were executed for different values of α (i.e. α =0.5, 0.7, 0.9 and 1), and different number of *epochs* (i.e., 10, 15 and 20). Note that epochs denote the number of passes the machine learning algorithm will work through the entire local data-set every time training is executed. An iteration denotes execution of the complete algorithm, that is starting from the broadcast of model parameters by the Cluster Heads to the operations performed by the Cluster Members as discussed in the previous section.

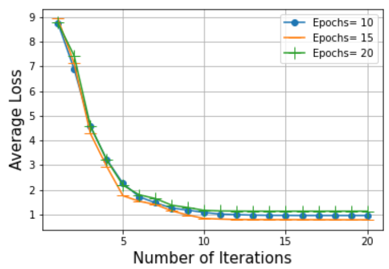
The average loss with respect to the number of iterations for i-WSN League when α is 0.9, is shown in Figure 2.29.



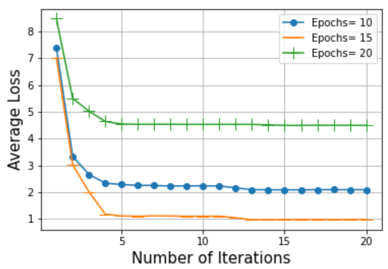
(a) Case 1



(b) Case 2



(c) Case 3

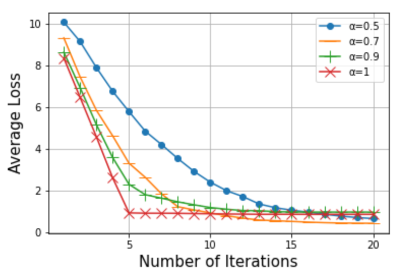


(d) Case 4

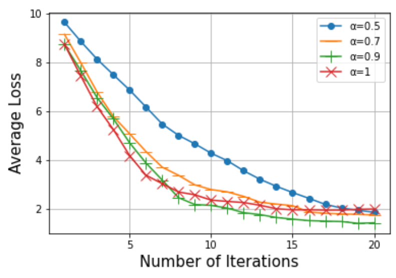
Figure 2.29: Average Loss vs Number of Iteration for different values of epochs

It is explicitily observed that in i-WSN League each iteration involves the broadcast of the model parameters of one Cluster Head. In Figures 2.29, we report the average loss with respect to the number of iterations for different values of epochs in each case. It is observed that, as soon as we increase the number of iterations in each case, the average loss significantly decreases. The loss convergence in Case 1 and Case 3 are quite similar as the clustering is also similar in both the cases. Both Case 1 and Case 3 have 5 clusters each and also it was observed that not only the the loss convergence was similar, but also they were quicker with respect to the iterations. In Case 2, which has 8 clusters, the average loss does not decrease as quickly as in Cases 1 and 3. In Case 4, it was observed that the average loss did not decrease further after 5 iterations. Note that the network in Case 4 has two clusters only, hence there are only two Cluster Heads available to perform training. Thus the performance does not improve after few iterations. In Figures 2.29 we observe how clustering influences the i-WSN League.

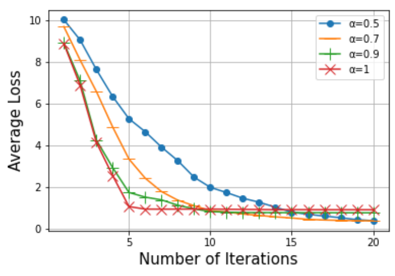
Similarly in Figures 2.30, the average loss is reported with respect to the number of iterations for different values of α and 15 epochs . It was observed that with higher values of α, the convergence is faster in all cases. Also in Figures 2.30, it was observed that the loss convergence for different values of α is quite similar in Cases 1 and 3.



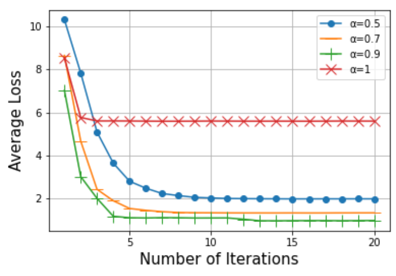
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 2.30: Average Loss vs Number of Iteration for different values of α

In figure 2.31, the average loss is reported with respect to the number of iterations in each case for 20 epochs and when α is 0.9. As mentioned before, the curves in case 1 and case 3 are similar which also shows that the convergence is faster in case 1 and case 3.

In figure 2.32, the standard deviation of the loss is reported with respect to the number of iterations when α is 0.9 and for 15 epochs.

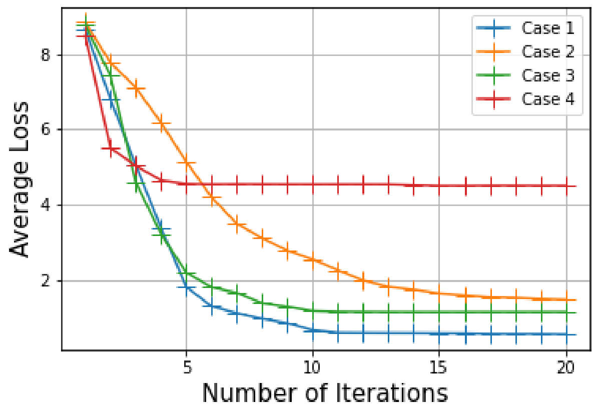


Figure 2.31: Average Loss vs Number of Iterations in each case.

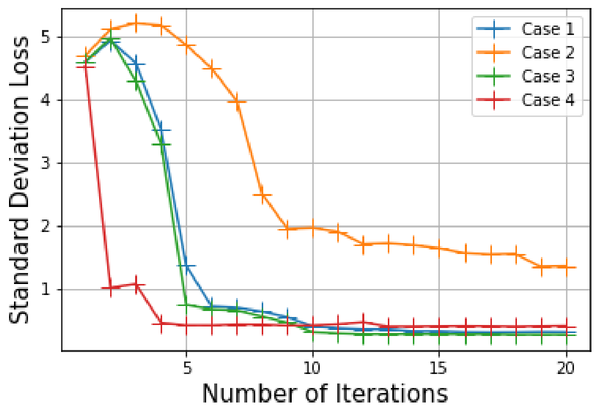


Figure 2.32: Standard Deviation Loss vs Number of Iterations in each case.

The average number of times a Cluster Head gets trained in the 20 iterations has also been reported. Note that, a Cluster Head does not perform training only at the beginning of an iteration, but it also trains on the data chunks sent by its Cluster Members. Figure 2.33 shows the average number of times Cluster Heads get trained in each case for various values of epochs and when α is 0.9. We observe that the average number of times a Cluster Head gets trained is higher in Case 4 and lower in Case 2 and it is quite similar in cases 1 and 3. It is higher in Case 4 because the Cluster Heads have to train on the data chunk sent by its Cluster

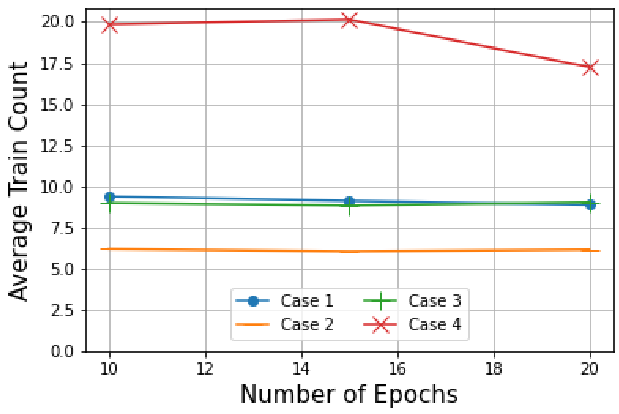


Figure 2.33: Average Train count vs Number of Epochs

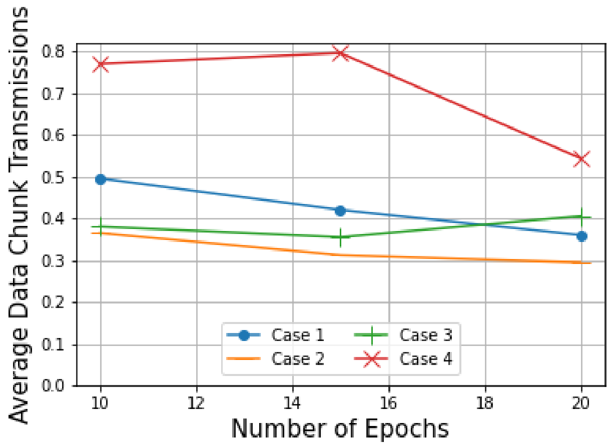


Figure 2.34: Average Data chunk transmissions vs Number of Epochs

Members. Therefore, the two Cluster Heads which have more than 10 Cluster Members will go through a large number of training cycles.

The training count in cases 1 and 3 are similar because the number of clusters in each case is same. Thus, the training load of the Cluster Heads is not as high as case 4 and not as low as case 2. Similarly, Case 2 has the lowest count because each Cluster Head has more or less two Cluster Members and therefore, the training load will be low. However, changing the number of epochs we note that the number of times a Cluster Head is trained is stable because the loss convergence is fast.

Analogously, in Figure 2.34 the average number of times Cluster Members have transmitted their data chunks to the corresponding Cluster Heads when α is 0.9 is reported. Similar observations to those regarding Figure 2.33 can be drawn.

For the sake of completeness, in Figure 2.35 the average number of times each Cluster Head has been trained in Case 1 when the number of epochs is 15 and

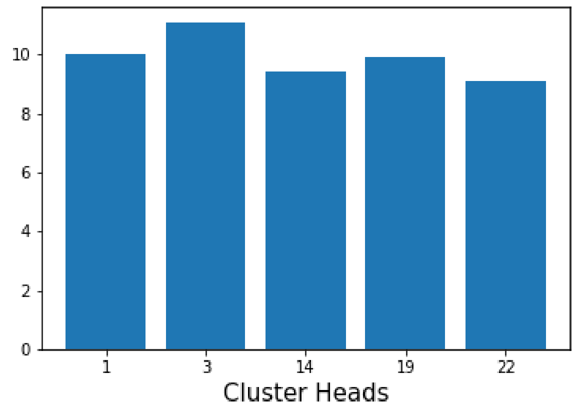


Figure 2.35: Average number of times each Cluster Head has been trained

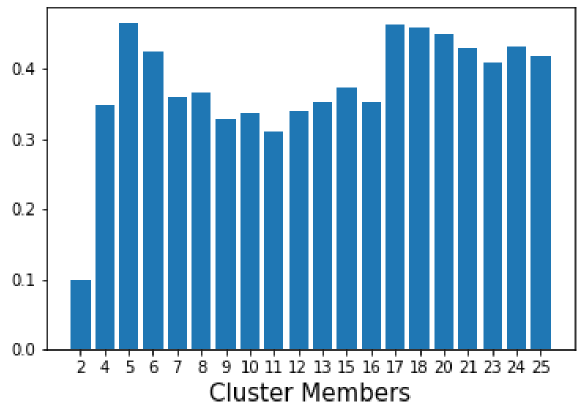


Figure 2.36: Average number of times each Cluster Member has transmitted data chunks

the value of α is 0.9 is reported. The Cluster Heads in Case 1 are nodes 1, 3, 14, 19, and 22.

In Figure 2.36, the average number of times each Cluster Member has sent data chunks to its Cluster Head in Case 1 when the number of epochs is 15 and the value of α is 0.9 is reported.

To assess fairness we calculated the Jain's Fairness index regarding the number of times the Cluster Heads have been trained in Case 1.

The Jain's Fairness index is calculated as [44]:

$$\mathcal{F}(x_1, x_2, ...x_n) = \frac{(\sum_{i=n}^{n} x_i)^2}{n \cdot \sum_{i=n}^{n} x_i^2} \tag{2.19}$$

for $n = 25$ sensors. In Table I we report the Jain's Fairness index for different values of α and the number of epochs in cases 1 and 2. we observed good fairness

α	Epochs	Train. Events (Case 1)	Train Events(Case 2)
0.5	10	0.97660	0.90735
0.5	15	0.96661	0.92125
0.5	20	0.97696	0.91307
0.7	10	0.95835	0.91007
0.7	15	0.96343	0.91550
0.7	20	0.96176	0.88494
0.9	10	0.93305	0.90062
0.9	15	0.90417	0.89184
0.9	20	0.90105	0.89360
1	10	0.92991	0.89283
1	15	0.89929	0.89084
1	20	0.91732	0.88954

Table 2.3: Fairness Index for regarding the number of times Cluster Heads have been trained for different values of α and epochs in Case 1 and Case 2

among the Cluster Heads in both cases.

2.6 Applying SDN to WSN with Data Mules

Wireless sensor networks (WSN)s play a fundamental role in environmental monitoring and have been the subject of extensive research effort for two decades [65]. Recently, several environmental monitoring solutions have been proposed that exploit *unmanned air vehicles* (UAV) in combination with WSNs [66, 67]. This is because from the sensing point of view, drones have features that complement those of WSNs nodes nicely. In fact, while WSNs nodes are usually deployed in large numbers in the monitored environment and are equipped with few low cost sensors, drones are way less numerous due to their higher cost but can be equipped with several expensive sensors. Therefore, in most application scenarios, the WSN nodes are deployed pervasively to monitor basic physical parameters and the drone flies throughout the monitored environment to take more sophisticated measures in areas where the WSNs nodes detect the occurrence of some critical event.

From a communication and networking point of view, the drone can also interplay with WSNs nodes effectively. In fact, by flying at a convenient altitude, drones can establish low attenuation, high data rate communication links with the network infrastructure and therefore are the perfect candidates for acting as the gateways (and thus the *sinks*) of the WSN [68], [69], [70]. Furthermore, since the drones usually fly in the proximity of WSN nodes, detecting the occurrence of critical events with the aid of drones can yield the following two advantages:

- Firstly, drones can immediately forward the data related to the detection of the occurrence of the critical event generated by the WSN nodes to the network infrastructure,hence, resulting in reduced delay and increased reliability;
- Secondly, if the WSN nodes increase the sampling rate to improve the monitoring of the critical event, the consequent packets are gathered by the drones right away without the need to be forwarded by several intermediate nodes. This results in increased energy and communication resource efficiency for the whole WSN.

In this work, a networking solution is desugned, called *SDN-(UAV)ISE*, for the integration of WSNs and UAVs that exploits the *software defined networking* (SDN) approach. SDN-(UAV)ISE extends a recent solution which applies the SDN approach to WSNs, named *SDN-WISE* [71, 72], to integrate UAVs in light of the specific features of their mobility patterns. In fact, we observe that most of the time drones fly according to a linear trajectory on the short term, therefore, it is possible for the SDN controller to predict the resulting topology changes and calculate packet routing paths in advance. Furthermore, since drones are expected to fly in the areas where a critical event is occurring, by observing the data generated by the WSN nodes, it is possible to forecast the medium-long term mobility pattern of the drones and infer the expected topology changes. Accordingly, in this work the two main contributions are:

- The architecture and protocol modifications in the SDN Controller as well as in the nodes, needed to exploit the prediction of the drone mobility are introduced;
- The advantages are assessed that are achieved by exploiting a simple artificial intelligence solution based on *decision tree learning* that exploits the data generated by the sensors to predict the medium-long term mobility of the drone.

In recent years, the application of Artificial Intelligence in the area of UAVs has gained popularity due to their autonomous moving capability and applications in various domains. some of the recent applications of Machine learning algorithms in the area of UAVs are the detection of drones for public safety applications[73], Field Data analysis and Modeling for drone communications [74], Wireless Connectivity and Security of Cellular-Connected UAVs [75]. Whereas, we predict the mobility of the drone in our work using the decision tree machine learning algorithm. And we exploit those predictions to create routes in advance.

2.6.1 SDN-(UAV)ISE Scenario and Overview

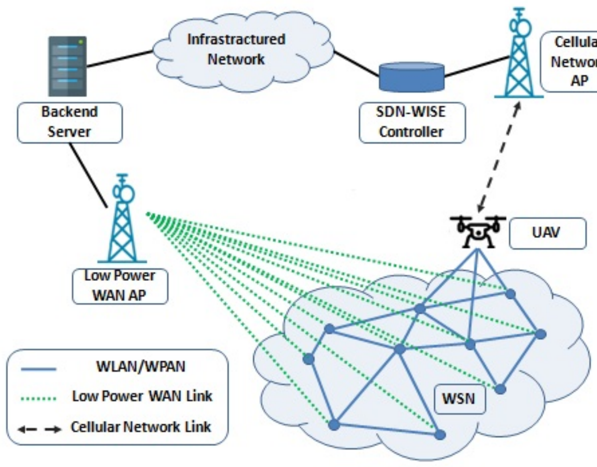


Figure 2.37: Proposed scenario.

The aim of this section is to illustrate step by step the SDN-(UAV)ISE functionalities. All the elements composing the SDN-(UAV)ISE architecture and how they are connected to each other is discussed. Then, in Section 2.6.2, details of all the new features introduced in the proposed approach are explained. Finally, in the Section 2.6.3, how the above components work together is shown.

The scenario depicted in Fig. 2.37 is considered. WSN nodes are equipped with two radio interfaces. The first one exploits a long range, low data rate wireless technology, such as LoRa or Sigfox. Such radio interface, which we denote as *long range wireless interface* is utilized by the nodes to send a few samples of the sensed parameters per day to the back-end server through an access point. The other, instead, exploits a short range, high data rate wireless technology. For such radio interface, which we refer to *short range wireless interface* there are several candidate technologies. The most popular are IEEE 802.15 and IEEE 802.11, but many other solutions exist. WSN nodes exploit the short range wireless interface to establish a multihop wireless network. Such network is connected to the network infrastructure through the drone which acts as a mobile sink.

Accordingly, the most important elements of the proposed scenario are the sensors, the sink deployed on the drone, and the SDN-WISE controller.

- *Sensor:* Sensor nodes, as shown in Fig. 2.38, implement physical and MAC layers which depend on the specific transceiver and micro-control unit (MCU)

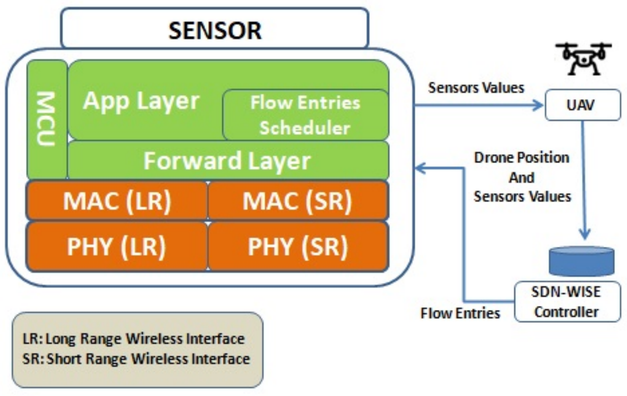


Figure 2.38: Sensors protocol stack.

utilised. The MCU executes the application layer which retrieves the sensor measurements and the Flow Entries Scheduler application. The Forwarding layer that handles the arriving packets as specified in the flow table is directly linked to scheduler.

- *Sink:* The UAV sink is the gateway between the sensor nodes running the Data plane and the Controller implementing the Control plane.

- *SDN-WISE Controller:* The network management logic is dictated by one or several Controller(s). The standard version of SDN-WISE includes a Topology Management (TM) layer which abstracts the network resources so that different logical networks with different management policies set by different Controllers can run over the same set of physical devices. More specifically, the TM layer collects local information from the nodes and provides the Controller(s) with this information in the form of a graph of the network (reporting information related to topology, residual energy level, SNR on the links, etc). In the proposed approach, the Controller runs in its Application Layer, the implemented algorithms and uses the information generated by the nodes to forecast the future UAV's positions in order to reduce the number of destinations based on the WSN topology.

2.6.2 New Features of SDN-(UAV)ISE

In this section, all the policies and mechanisms used by the proposed SDN-(UAV)ISE approach to create back-up routes, to reduce the number of destinations inside the topology as well as to schedule the flow table entries within the sensors are described:

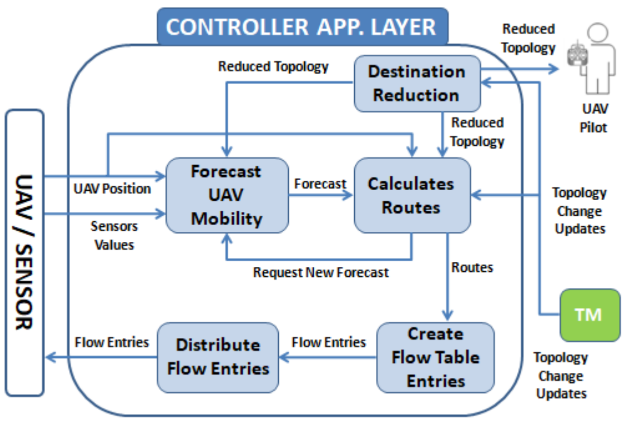


Figure 2.39: Controller Application Layer block diagram.

- *Destinations reduction:* When working with UAVs, energy and saving time are certainly the two most important pressing issues. Misuse of the resources can drastically reduce the operating time of the UAV. For this reason, in scenarios, where a plethora of sensors are deployed, reducing the number of destinations to be reached by the UAV while maintaining the radio visibility with all the sensors is crucial. To achieve this goal, we apply a scheme that identifies a set of positions such that the area surrounding any of the sensors is inside the radio and sensing coverage of the drone, when the drone is in one of the positions identified by our scheme. We will now describe such procedure.

  Starting from the set U of all the GPS coordinates of the deployed sensors, consider all possible subsets of U which we denote $S_1, S_2, ..., S_k$, such that for any $i \leq k$, there is a position, p_i, such that all coordinates in S_i are within the radio and sensing coverage of the drone when this is visiting position p_i. We defined C as a collection of subsets taken from $S_1, S_2, ..., S_k$ whose union is U.

  We want to find the optimal collection C_{opt} with the minimum number of subsets. This problem is a well known problem in the literature and is known as "The Set Cover Problem" and is one of the Karp's 21 NP-complete problems. In general, the problem of determining the optimal C,ie., C_{opt} is NP-complete and an optimal solution can be found using a greedy algorthm, like in [76] or in [77]. In most WSN scenarios, this optimal solution reduces considerably the number of destinations the UAV needs to visit to collect data from the deployed sensors, thus saving time and energy.

  Note that, the set cover greedy algorithm is constantly informed by the TM layer about any topology change. Therefore, C_{opt} might change over time.

Every time a change occurs in the WSN topology, the new optimal solution will be calculated with the aid of the UAV pilot and Forecast UAV Mobility module.

- *Forecast UAV Mobility:* Forecasting the movements of the drone is performed using a decision tree algorithm executed in the Controller. The decision tree is generated by learning the training data that consists of the latest information from the sensors. The training dataset is updated each stage since the data from the sensors keeps varying. The rationale behind such approach is that the pilot will most likely fly the drone to locations where critical events occur and therefore, our forecast scheme considers the drone positions and the data taken from the sensors to forecast the UAV mobility.

- *Calculates Routes:* Based on the predicted movements of the drone, it is possible to forecast the topology changes for a given time window. The Calculate Routes component will evaluate the *optimal* paths between all sensors and the sink (i.e., the drone) for each predicted instance of network topology along with the corresponding time intervals when those optimal paths are supposed to be valid. Observe, that in our current implementation, optimization paths are selected so as to minimize the number of hops, however, different optimization objectives can be chosen based on the specific application requirements.

- *Create Flow Table Entries:* The creation of a Flow Table Entry takes as input the current WSN topology and its predicted changes known to the TM layer which are then shared to the Create Flow Table Entries component as well as. In fact, the Create Flow Table Entries component will create the flow entries that are used to reach the UAV in any possible scenario reducing the number of packets sent into the network.

- *Distribute Flow Entries:* After receiving the flow entries to be distributed, the Distribute Flow Entry component identifies the most effective and efficient way to deliver them to the intended WSN nodes depending on the current network topology. Some of such entries may be dispatched via long range radio interface and others via the short range wireless interface.

- *Flow Entries Scheduler:* The Flow Entries scheduler is a component which manages the incoming packets from the controller and retrieves the time at which a specific flow rule should be applied in the sensor. This table is continuously updated by the Forwarding layer according to the configuration commands sent by the Flow Entries Scheduler. This scheduler operation is managed by the MCU of the WSN nodes and it interacts frequently with the Forwarding layer. It is also responsible for the correct activation of the correct flow table entries in the sensor. Each time a packet arrives from the Controller, the Forwarding layer extrapolates the information as the flow entry and retrieves the time at which the flow rule should be applied and passes it to the scheduler. The scheduler then, applies the corresponding flow entry at that given time. This would reduce the network overhead and the number of control packet inside the network .

2.6.3 SDN-(UAV)ISE in action

The objective of this section is to provide an overview about how the components described in the previous Section 2.6.2 interact with each others.

Let us start by focusing on the Controller whose architecture is shown in Fig. 2.39. Note that, it receives as inputs the updates provided by the TM regarding the sensor network topology as well as the current position of the UAV and the values measured by the sensors.

The sensor network topology updates are utilized by the Calculate Routes component to find the optimal paths as we will explain in the following.

Information regarding the current position of the drone and the values measured by the sensors, are indeed utilized by the Forecast UAV Mobility component to predict the movements of the drone. More specifically, when the drone reaches one of the aggregation points selected by the Destination Reduction component, the Forecast UAV Mobility component applies Decision Tree Learning to predict the following destinations for the drone covering a time window. Duration of such time interval, x, is selected in such a way that the signaling produced by the Controller to update flow tables in the sensors is minimized. We will provide further details about such step in Section 2.6.4.

The predicted movements of the UAV are given as input to the Calculates Routes component that uses such information along with the current topology provided by the TM to evaluate the corresponding predicted topologies. In other terms, as the drone moves its points of attachment with the sensor network, this may lead to topology changes.

For example, suppose, that, according to the prediction of the Forecast UAV Mobility component, the topology of the sensor network should be

- T_1 in the time interval (t_0, t_1);
- T_2 in the time interval (t_1, t_2);
- ...
- T_n in the time interval (t_{n-1}, t_n), where $t_n = x$.

For each of the topologies T_1, T_2, ..., T_n, the Calculates Routes component will calculate the optimal path between each sensor and the sink. In other terms, let $P_{i,j}$ be the optimal path, i.e., the sequence of connected nodes, between the sensor i and the sink when the topology is T_j. The optimal paths, along with the time interval will be given as input to the Create Flow Table Entries component. This component is responsible for creating the resultant Flow Table Entries along with the time intervals when those Flow Table Entries can be considered to be effective. This aforesaid information is given as input to the Distributes Flow Entries component that is responsible for delivering each Flow Table Entry to its intended destination. Observe, that, to achieve this purpose, both the WLAN and WAN communication interfaces can be utilized.

Note, that, when a sensor receives the new Flow Table Entries, these entries will be given to the Flow Entries Scheduler, shown in Fig. 2.38, which will insert the correct Flow Entry in the Flow Table at the intended time instant.

2.6.4 SDN-(UAV)ISE Performance evaluation

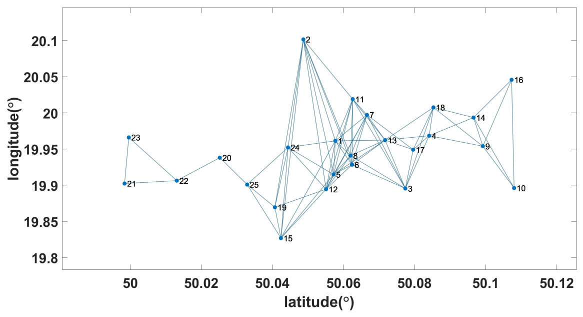
In this section, the experimental setup considered for the assessment of SDN-(UAV)ISE is decribed. Then, in Section 2.6.5, we will discuss the numerical results.

A scenario is considered in which sensors are deployed to measure the quality of air in an urban area. A drone flies over such an area for two purposes:

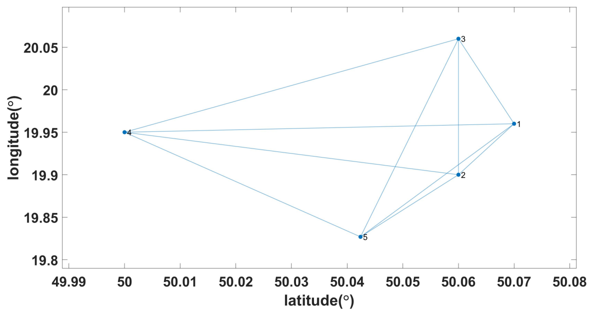
- It acts as a gateway between the wireless sensor network and the network infrastructure;
- It measures further physical features regarding the environment by exploiting some specific sensor it is equipped with, e.g., cameras.

The dataset contains the values of temperature, humidity, pressure, and concentrations of several pollutants (i.e., PM1, PM2.5, PM10) measured by 56 low cost sensors deployed in the city of Krakow (Poland) in 2017[7].

[7]https://www.kaggle.com/datascienceairly/air-quality-data-from-extensive-network-of-sensors



(a) 25 sensor nodes



(b) Reduced Destinations

Figure 2.40: Standard deviation of the Loss vs. No. of Iterations (MGM-4-FL)

The location of the 25 sensor nodes and their corresponding reduced destinations is shown in Figures.2.40a and 2.40b.

The original dataset has been elaborated along two directions, i.e., we thicken the data and add the position of the drone, as described below.

Thickening of the data is carried out in our work because, we note that the original dataset contains one sample per hour. However, we wanted to have more frequent updates about the air quality. Therefore, for each sample, we have generated 100 more samples through linear interpolation and therefore, we have one sample for every 36 seconds.

Similarly, regarding the position of the drone, we observe that there is no experimental data providing information about the movements of a drone in an area covered by a WSN.

Therefore, a mobility model has been developed to generate a synthetic trace of the drone position over time denoted as $v_0(t)$. The i-th reduced destination of sensors is denoted as p_i. The drone will keep moving from one reduced destination to another one. More specifically, at a given time t, the next destination is selected randomly from the reduced destinations, say the i-th with a probability $P_i(t)$

which is proportional to a *weight* function, $W_i(t)$, that is,

$$P_i(t) = W_i(t) / \sum_{j \neq j^*} W_j(t) \tag{2.20}$$

where j^* denotes the calculated reduced destination that the drone has just reached.

The weight $W_i(t)$ is assigned according to the following rationale [78]:

- At each step, any of the other calculated reduced destination can be selected as next destination;
- It is likely that the drone will visit the reduced destination of sensors that are measuring critical values and that have not been visited for some time;
- It is likely that the drone will visit the reduced destinations that are close by instead of wandering from one end of the operation area to the opposite end.

Accordingly, $W(i)$ is evaluated as follows:

$$W_i(t) = \alpha \cdot \mathrm{Destcritic}_i(t) + (1 - \alpha) \cdot \mathrm{proximity}_i(t) \tag{2.21}$$

where α is a parameter that lies in the range [0,1] which is used to model the pilot attitude to consider the values measured by the sensors to decide the next destination for the drone. The function $\mathrm{Destcritic}_i(t)$ is the *Destination criticality* function that calculates the maximum of the criticality values of the sensors corresponding to the i-th destination. The criticality value of the sensor is calculated by the function $\mathrm{critic}_i(t)$ that measures how *abnormal* have been the values measured by the i-th sensor, since the last time, the drone has visited the corresponding reduced destination and the $\mathrm{proximity}_i(t)$ is a function that decays with the distance between the current position of the drone and the i-th reduced destination.

More in detail, the criticality parameter of the i-th sensor is evaluated as follows: $\mathrm{critic}_i(t) =$

$$= \begin{cases} \mathrm{critic}_i(t-1) + 1 & \text{if } x_i(t)) > \sigma \\ critic_i(t-1) & \text{if } x_i(t)) \leq \sigma \end{cases} \tag{2.22}$$

where $x_i(t)$ denotes the air quality measured by the i-th sensor at that moment and σ is the threshold value which is set to be 25. The threshold is obtained by calculating the sum of the standard deviation and the mean of the measured sensor values. The rationale of eq. (2.22) is that the criticality of the sensors increases as long as the drone is not in the range of the reduced destination and is set to 0 otherwise.

The $\text{proximity}_i(t)$ is calculated by the formula suggested in [78], i.e.,

$$\text{proximity}_i(t) = \frac{1}{1 + \kappa d(v_0(t), p_i)} \tag{2.23}$$

where, κ is a constant value which we set to $\kappa = 0.05$.

2.6.5 SDN-(UAV)ISE Numerical results

The criticality is compared between the two cases. In the first case, the drone randomly moves from one observation point to another. The Controller reactively updates the flow tables when it detects a topology modification. In case 2, the drone moves according to the algorithm and the Controller reactively updates the flow tables when it detects a topology modification. Fig. 2.41a shows the variation of criticality when a drone moves randomly and for different values of the parameter α needed in eq. (2.21). As shown in Figures. 2.41a2.41b and 2.41c, the criticality is considerably higher for the case in which the drone moves randomly when compared to the case in which the drone moves according to the algorithm. We highlight that the mobility model is not known to the proposed algorithm.

The forecasting of routes is performed for every 12 minutes using the Decision tree algorithm. The training data-set (X_i,Y_i) offered to the decision tree model, where X_i represents the input attribute set. X_i contains the attributes: temperature, pressure, humidity and pollutants concentrations (PM1, PM2.5, PM10). Y_i represents the objective attribute which is the drone position.

Performance of SDN-(UAV)ISE has been assessed in terms of a *utility function* defined as

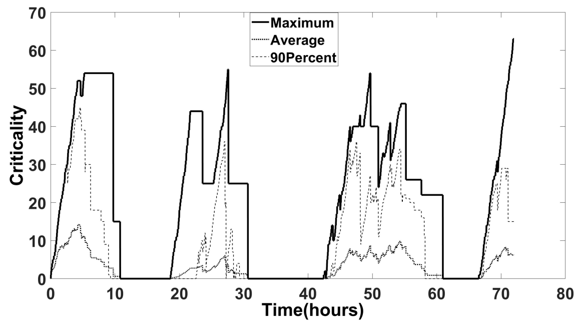
$$u(x) = \zeta(x)/\Delta(x) \tag{2.24}$$

where x is the duration of the forecast window, $\zeta(x)$ is the time interval between two transmissions of flow table updates, and $\Delta(x)$ is the packet size.

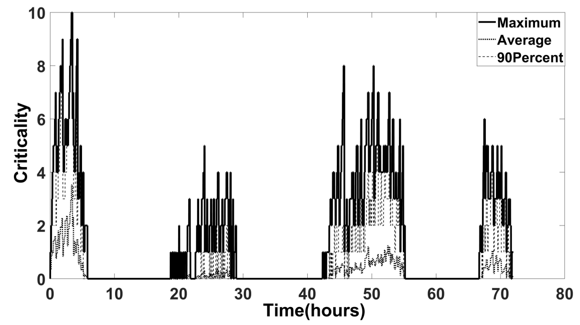
Note. that. $\zeta(x)$ is given by

$$\zeta(x) = \begin{cases} \mathrm{T} & \text{if } \mathrm{T} < x \\ x & \text{if } \mathrm{T} > x \end{cases} \tag{2.25}$$

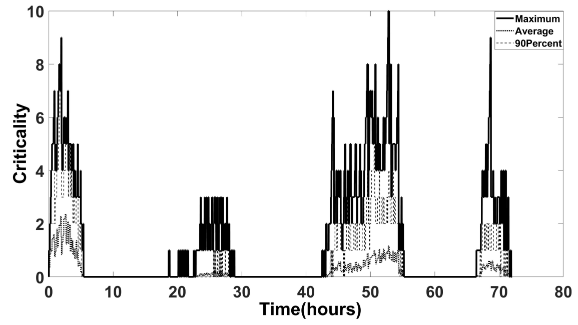
where, T is the time when the first estimation error occurs. In fact, given that the duration of the forecast window is x, the maximum interval between a trans-



(a) Drone moving randomly



(b) $\alpha = 0.5$



(c) $\alpha = 1$

Figure 2.41: Variation of criticality when drone moves randomly and for different values of α

mission and the following one is x. However, if the forecast drone position turns wrong after a time interval T, new flow table entries must be sent.

For what concerns the size of the packets containing the flow table entries, observe that their average size increases linearly with x. In fact, the longer the forecast window duration, the higher the number of flow entries that must be carried by each signaling packet. Note, that, the average packet size is not proportional to x because each packet should include a header.

Accordingly, the average utility function can be calculated as follows

$$E[U(x)] = E[\zeta(x)]/(x + k) \tag{2.26}$$

where k is a constant which weights the contribution of the packet header in the overall packet size.

Note, that, x should be set in such a way that $E[U(x)]$ is maximized. Fig. 2.42 shows the expected utility variation with respect to x. In Fig. 2.42, we report the

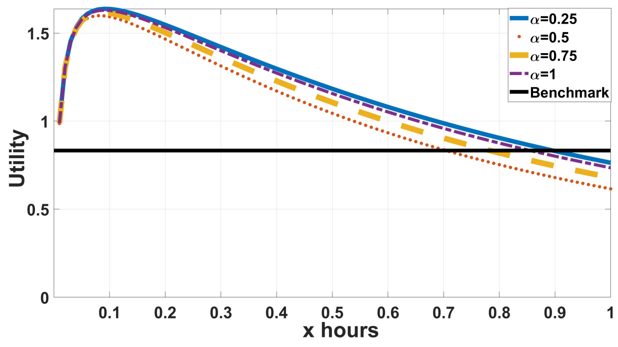


Figure 2.42: Expected Utility with respect to the duration of the forecast window.

utility for different values of α. In the same figure, we show the utility function which would be obtained without exploiting SDN-(UAV)ISE mechanisms. Note that for a wide range of x values, the utility function achieved by SDN-(UAV)ISE is higher, which means it improves resource efficiency.

Chapter 3

Vehicular Networks

3.1 Introduction

This chapter focuses on the exploitation of ML in vehicular safety applications. The first part of the chapter focuses on the application of ML to detect anomalies in driving behaviour. More specifically, a CNN based auto-encoders algorithm was applied to detect the anomalies in real time and warn the vehicle driver. The well known transfer learning approach was applied to a case study in which driving data was collected from different different bicyclists riding along the same path in the city of Catania. The second part of the chapter focused on achieving layer separation in deep neural networks for road-user interaction analysis in smart road environments. For vehicular safety applications, it is not idea to build a common ML model for all the users which would be effective in all kind of road scenarios. The driving behaviour would change with person to person and it would also change depending on the road environments. To address this issue efficiently, a V2I framework was designed based on a training technique that partitions the Neural Network (NN) layers of the model into some layers specific of the user behaviour and others layers specific the road-environment. specific of the road-environment where the vehicle is currently located. This approach was experimented with the same bicycle data-set obtained from different bicyclists.

3.2 ML for Vehicular Safety Applications

The safety of vehicle users is an important consideration while designing and developing intelligent transportation systems. According to the World Health organization, approximately 1.3 million people die each year as a result of road traffic crashes [79]. Monitoring and analysing the driving behaviour in real time can be helpful in detecting anomalies and warning the driver and nearby road users. Generally, the data required for analysing the behaviour of the driver are collected through various on-board sensors installed in the vehicle. Indeed, vehicles manufactured in the last two decades are equipped with variety of sensors installed in the vehicle to monitor speed, temperature, fuel consumption, engine RPM, steering wheel angle and break pedal pressure and more. Note, in fact, that the corresponding values are related to the driver action and reflect the attitude and behaviour of the vehicle. In the recent times, usage of such data has become easier thanks to the introduction of the OBD II protocol which gives access to such sensor values and the vehicle diagnostics [80].

Due to variety of sensor data generated from the vehicles, ML algorithms have been exploited for driving behaviour analysis and other safety applications [81],[82] and anomaly detection [83], [84].

3.2.1 ML for driving behaviour analysis

In this section, the most relevant literature on the application of machine learning scenarios where the road-user interactions are relevant, especially in the area of driving behaviour analysis. Driving behaviour describes the intentional and unintentional characteristics and actions a driver carry out while operating a vehicle. They include driver hand and body movements, foot dynamics, eye gaze dynamics, head rotation and more.

ML algorithms have been widely used in analysing and studying the driving behaviour.

In [81], the authors propose tools based on support vector machines and feed forward neural networks to classify whether the driving behaviour is safe or unsafe. The authors trained the ML model on a publicly available data-set comprising of in-vehicle data collected from 26 hours of driving time. Parameters such as engine speed, vehicle speed, break pedal position are considered for the model training.

In [82], the focus is on the analysis of driving behaviour of truck drivers. A classification framework is proposed consisting of four different machine learning algorithms to classify unsafe driving behaviours such as drunk driving, fatigue driving, speeding and more. Similarly in [85], the objective is the detection of the driving behaviour using GPS sensor data. More specifically, deep neural networks such as recurrent neural networks and long short term memory networks are exploited to characterize the driving behaviours. In [86], the authors utilized the distance to the car in front and also the lane position apart from the commonly used parameters such as speed and steering positions. The authors compared two linear and non-linear ML algorithms for identifying drug induced behaviours.

Apart analysing the driving behaviour of an user, the ability to detect anomalies in the driving behaviour and in the road has the potential to limit the number of traffic accidents and save human lives. In fact, ML techniques have been proved to be effective in detecting anomalies and abnormal driving behaviours.

In [87], the authors propose a ML based method to detect road anomalies by analysing driving behaviours. The ML algorithm is applied on the data collected from smartphone inertial sensors. The proposed approach was able to detect 70 percentage of the swerves and more than 90 percentage of the turns on the road.

In [88], the Online Sequential Extreme Learning Machine (OS-ELM) was applied to detect anomalies in driving behaviour. The OS-ELM is a an efficient neural network model that has high memory efficiency and can perform quick sequential learning with streaming data. Authors compared the application of OS-ELM to hidden Markov model and the traditional Long short term memory (LSTM)-based neural networks and found that OS-ELM has better accuracy and higher learning rate.

Apart from classical ML approaches such as K-Nearest Neighbor (k-NN), Decission Tree and support vector machine, deep neural networks such as auto-encoders, LSTM, and transformers have also been applied for anomaly detection [89–91].

In Section 3.3, a Transfer Learning (Tl) approach with CNN based auto-encoders is proposed for the detection of anomalies in cycling behaviour. The proposed approach is applied to a specific case study and experimental results are obtained. In section 3.4,

3.2.2 ML for Autonomous Vehicles

Ml has been widely used in automotive industry for various applications, thus making self-driving cars a reality. According to the U.S National Highway Traffic Safety Administration (NHTSA), 94 percentage of the road accidents were caused due to human error. Therefore the use of autonomous vehicles could reduce the errors that humans make and a large number of road crashes could be avoided. In case of commercial sector, autonomous vehicles have the benefit of lowering costs. Driver-less delivery will result in reduced labor costs for truck drivers and other delivery drivers.

Though autonomous vehicles are in testing and early prototyping stages, ML is being used in various features of the technology in Advanced Driver-Assistance Systems (ADAS). And ML seems to play a major role in the future developments as well. Some of the aspects of autonomous vehicles in which ML is applied are discussed below.

Perception is an important aspect of autonomous vehicles as they need to perceive the surrounding environment. The perception approach that exploits deep learning for autonomous driving systems that uses Light Detection and Ranging (LIDAR) point cloud information is known as Simultaneous Segmentation and Detection Network (SSADNET) [92]. The SSADNET technique can detect and differentiate both driviable areas and obstacles which are very critical for the point of view of autonomous driving. Deep learning techniques are also used for driving environment perception and localization [93]. Since CNN is a powerful ML algorithm for image processing, it has been used for object recognition, distance estimation and classification of vehicles and pedestrians. In [94], a CNN based multi-task method is proposed to jointly model object detection and distance prediction.

Another important aspect of autonomous driving is motion planning. It is very essential to understand the driving behaviour of nearby vehicles driven by humans and to evaluate and estimate the lane shifting of the autonomous vehicle. The main inputs to the motion planner are lateral acceleration, longitudinal speed and yaw rate.In [95], a deep neural network technique known as Light Gated Recurrent Unit (Li-GRU) is exploited for trajectory estimation in autonomous vehicles. Reinforcement and deep reinforcement learning techniques play a major role in advance forecasting for autonomous vehicles [96].

Pedestrian detection is a crucial task of autonomous driving. However the issue in the exploitation of pedestrian detection techniques for autonomous vehicles is is the extreme amount of data processing. The ML techniques for pedestrian detection includes feature extraction and classifiers and also deep CNN is a widely used technique for this task. The Aggregate Channel Feature (ACF) is a widely recognized pedestrian detection algorithm [97]. However this algorithm fails to detect targets when they are occluded or very small. To overcome this issue, a pedestrian detection algorithm is proposed which is based on a combination of five layer CNN structure and an AdaBoost classifier (CNN–AdaBoost) [98]. The error detection rate of this algorithm was considerably reduced when compared to the well known ACF algorithm

Another important aspect of autonomous driving in ML algorithms are exploited is vehicle cyber-security. The chances of cyber-attacks in autonomous vehicles increases with the application of embedded technologies in connected vehicles and vehicular communication systems. The advancement and success of autonomous vehicle systems depends on the usefulness of sensors being used and the strength of communication technologies deployed. However, the sensors and the communication systems deployed have high security concerns as an attacker could take control of the autonomous of the vehicle by feeding wrong data to the autonomous vehicle systems. In [99], a deep reinforcement learning technique using LSTM-GAN is introduced for autonomous vehicle systems for maintaining safety and security. In this GAN based approach, the adversary tries to feed defective data to the sensor readings or tries to make sure that there is no optimal distance between the autonomous vehicles. On the contrary, the autonomous vehicle will try to deafened itself from such attacks.

Other aspects of autonomous driving in which ML algorithms can be utilized are self-localization, motion-control and automated-parking. While it may not initially be more accurate than vision-based systems, ML algorithms can achieve greater accuracy over time.

3.3 A Case Study: Detection of anomalies in cycling behavior with CNN

Cycling has always been taken as sustainable and healthy transportation mode. With the increasing rate of population, transportation gases and pollution in urban areas, policy makers are pushing bicycle as fare mode of transportation. Even, during Covid-19 period cycling was more encouraging transportation mode by citizens for mobility. Unfortunately, with the growing number of bicyclists the rate of bicycle related road crashes is also increased. Therefore, bicycle safety assessment and management has become a challenge due to the limited availability of crash data and their reactive nature. In this field, artificial intelligence and availability of new data sources in smart cities and Communities offer new opportunities.

Almost 2000 cyclist deaths are caused by road accidents every year in Europe with an increasing trend in urban areas. Traffic Conflict is an alternative proactive approach to the crash data collection and global navigation satellite systems (GNSS) makes easy to track detailed cycling path. In our work, we focus on modeling the cycling behavior of bicyclists to detect anomalies, that can be associated to critical situations requiring an evasive maneuver. ML solutions could play an effective role in handling large amount of heterogeneous data, which could be helpful in applications such as anomaly detection. However, training the model requires large amount of data and different users may need tailored models to

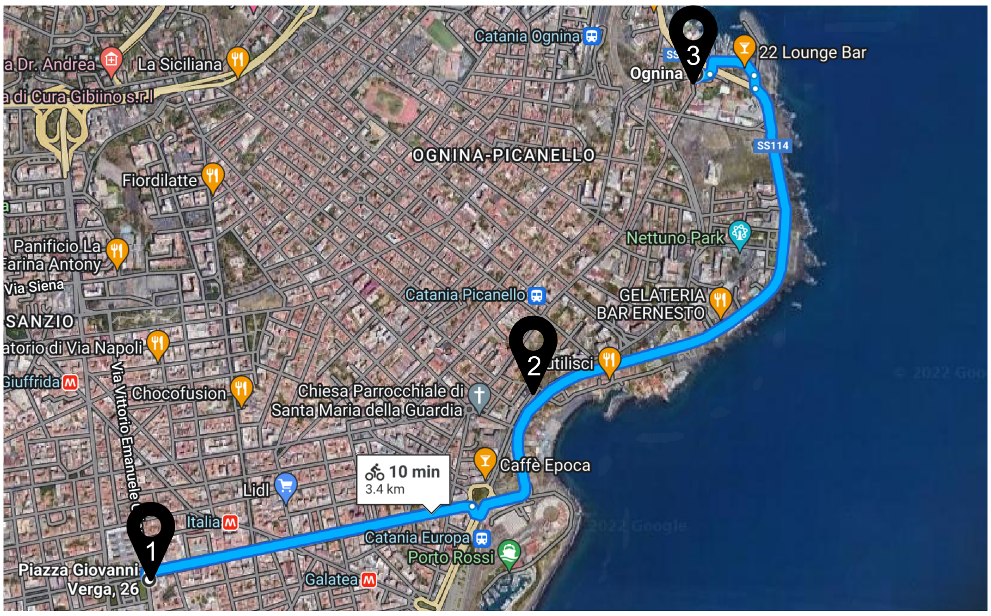


Figure 3.1: Bicycle Path

account for different cycling behaviors. Hence, a CNN based Transfer Learning approach is introduced detect anomalies in cycling behavior.

3.3.1 Bicycle Dataset

In this section, a specific case study is considered to which the Transfer Learning approach can be applied. A bicycle was equipped with sensors to collect video recording and GPS data 10 Hz. Speed and heading have been collected from the NMEA string (standard data format supported by the GNSS), as well as location coordinates. The data was collected by 8 bicyclists riding along the same path in the city of Catania in Italy. After data collection, various python functions were implemented to 1) clean and smooth dataset 2) compute derived parameters 3) create training testing sets for the model. Speed and heading are the basic recoded time series data while we derived longitudinal acceleration, heading rate, transversal acceleration and combined acceleration.

As shown in figure 3.1, the data was collected by 8 bicyclists riding from point 1 to point 3. We mark the path from point 1 to point 2 as scenario 1 and this path has busy traffic. We mark the path from point 2 to point 1 as scenario 2 and this path is along the sea side.

Since the data is time-series, the Convolutional auto-encoder is an ideal choice for the anomaly detection task.

3.3.2 Transfer Learning with CNN based Auto-encoders

The objective is to detect anomalies in real time and warn the bicyclists. Hence the Auto-encoders ML algorithm was considered for the case study. Auto-encoders is an unsupervised ML algorithm that is used for learning the data representation [100–102]. The auto-encoders compress the input into a low dimensional code and then reconstruct the input from this compressed representation. Therefore, an auto-encoder basically comprises of three components: encoder, code and decoder. Auto encoders are used for dimensionality reduction, feature extraction and anomaly detection. In our experimentation, we use auto-encoder for the purpose of learning the representation of the bicycle data and detecting anomalies. There are various types of auto-encoders such as vanilla auto-encoder, stacked auto-encoder, denoising auto-encoder, variational auto-encoder and convolutional auto-encoder.

In our experiments with the bicycle data, we use the convolutional auto-encoder (CAE). CAE combine the advantage of convolutional filtering in convolutional neural networks with unsupervised pretraining of Auto-encoders [103, 104]. The encoder in CAE contains convolutional layers and the decoder contains deconvolutional layers.

The performance of the auto-encoder model is evaluated in terms of mean square loss. The mean square loss is evaluated from the data reconstructed by the autoencoder and the input data fed into the autoencoder according to the eq. (3.5), i.e.,

$$MSE(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y - \hat{y}_i)^2 \tag{3.1}$$

where y is the input data fed into the autoencoder, $\hat{y}$ is the reconstructed output data and N is the length of the input data.

Therefore, in order to detect the anomalies, a threshold value is set. If the loss of the auto-encoder model on the data is greater than the threshold, then it means the data is anomalous and if the loss is lesser than the threshold, then the data is not anomalous.

However, training the model requires large amount of data and different users may need tailored models to account for different cycling behaviors. Hence, the the Transfer Learning approach is used with the auto-encoder model to reduce the computation time and eliminate the requirement of large amount of data for training.

Transfer Learning (TL) is a research solution in Ml in which a model developed for an application or a task is reused as the starting point for a model on a second task. TL has become quite popular in the area of machine learning and deep learning in the recent times. In TL, the pre-trained models are used as the starting point on computer vision and natural language processing applications given the high computation load and time resources required to Ml models [105, 106].

Since, a CNN based auto-ecnoder algorithm is used in this case study for anomaly detection and CNN requires large amount of data for training and high computation time, TL is considered as an ideal solution for learning in applications where CNN is utilised. TL has been used with CNN for computer vision application such as image classification for improved efficiency and saving the training learning time [107–110].

TL has also been used in vehicular safety applications such as driver anomaly detection similar to our case study [111, 112].

3.3.3 Numerical Results

In this section, the numerical results obtained by evaluating the Transfer Learning approach on the bicycle dataset is discussed. For the experimentation, 4 different cases were considered as follows:

- **Case 1**: Case 1 is similar to the standard ML approach. The training is done on each user/bicyclist data and the models were tested on their respective user data. More specifically, 50 percentage of the data in each user was considered for training and the remaining 50 percentage was considered for testing.
- **Case 2**: In case 2, the training was done only on the user 1 data and the trained model was tested on the testing data of all the other 7 users. This case is referred as the benchmark case.
- **Case 3**: Case 3 is the standard Transfer learning case. The model trained on user 1 data was utilised for Transfer learning. More specifically, the trained model was trained again on each user's data before testing on each user's testing data.
- **Case 4**: Case 4 is again a Transfer learning approach like case 3. However in case 4, the inner layers of the Ml model are frozen.

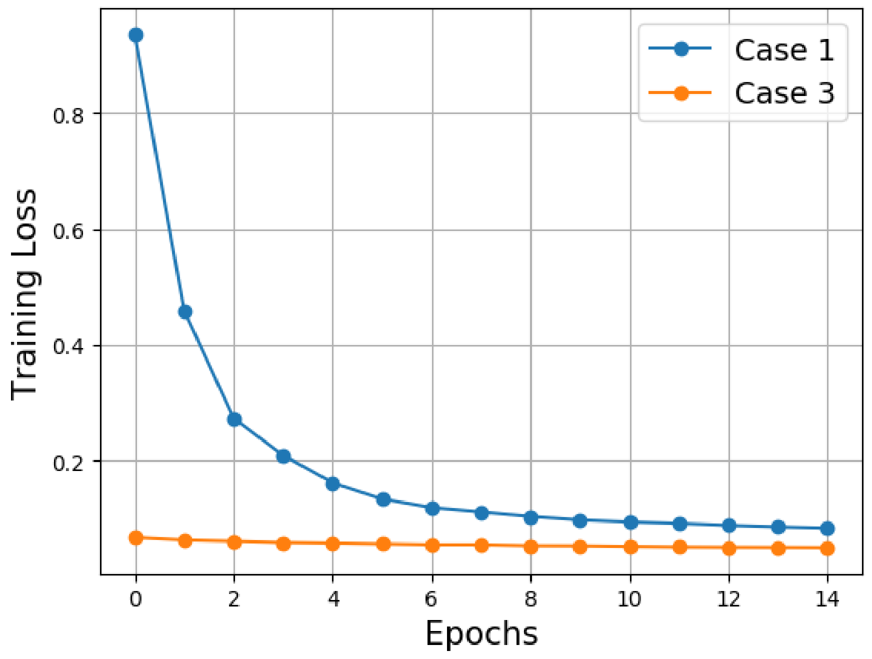


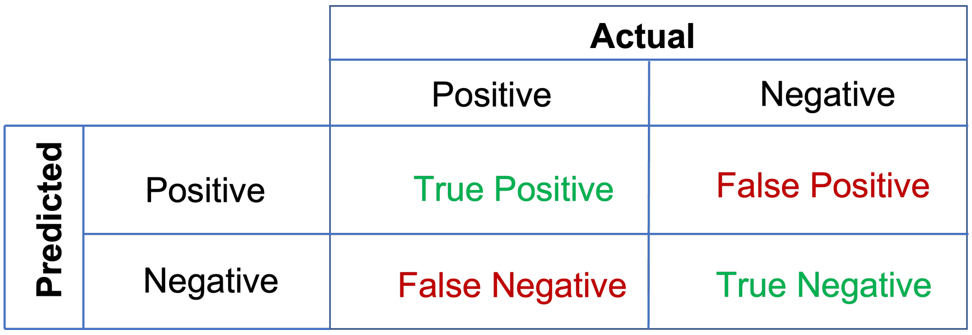
Figure 3.2: Training Loss vs Epochs

Several simulations were conducted to estimate the Training Loss, Recall, F-score and Precision in all the four cases. In figure 3.2, the training loss in case 1 and case 3 are compared to show how transfer learning can prove to be efficient.

As shown in figure 3.2, the convergence is quicker in case 3 (Transfer Learning) when compared to case 1 (Standard case). Note that, in case 3, the loss shown in the graph is the training loss of the pre-trained model. Therefore, by the use of transfer learning, the training time and the cost can be considerably reduced.

Setting the threshold is an important parameter in detecting anomalies using auto-encoders. Generally, the threshold is set as the sum of the average and the standard deviation of the ML loss. The threshold was varied and the Recall, Precision and F-score were compared between the four cases. Note that, it is required to calculate the number of True positives (TP), False positives (FP) and False negatives (FN) to evaluate the Recall, Precision and F-score. They are calculated according to confusion matrix as shown in figure 3.3

The Recall, Precision and F-score are evaluated according the equations 3.2, 3.3 and 3.4. Recall measures the number of positive class predictions made out of all positive data examples in the dataset. Precision measures the number of positive class predictions that actually belong to the positive class. F-Measure denotes a single score that balances both the concerns of precision and recall in one value.



		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Figure 3.3: Confusion matrix

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.3}$$

$$Fscore = \frac{(1 + \beta^2) * (Precision * Recall)}{\beta^2 * (Precision + Recall)} \tag{3.4}$$

Two commonly used values for β are 2 and 0.5. The value 2 weighs recall higher than precision and the value 0.5 weighs recall lower than precision.

The variation of the Recall with respect to the threshold values for all the four cases is shown in figure 3.4. As seen in the figure, the recall values for case 3 and 4 (Transfer learning approaches) are significantly higher than the benchmark case 2. The recall values for case 1 is also higher. However in case one, there were 8 models unlike the transfer learning cases or case 1 in which there was a common model for all the users.

Similarly, the variation of Precision with respect to the threshold values is shown in figure 3.5. Similar to the Recall variation, the case 2 exhibited lower values of Precision when compared to the other cases. The transfer learning cases exhibited better values of precision for different values of threshold.

Similar observation was made also in the case of F-score as shown in figure 3.6. Note that F-score is a single score that balances both the concerns of precision and recall in one value. The benchmark case exhibited lower values of F-score when compared to the Transfer Learning cases.

Note that all the recall, Precision and F-score values shown in the graphs are the average values of all the 8 users. As seen in the figures, Transfer Learning can be very efficient in terms of computation and data required for training.

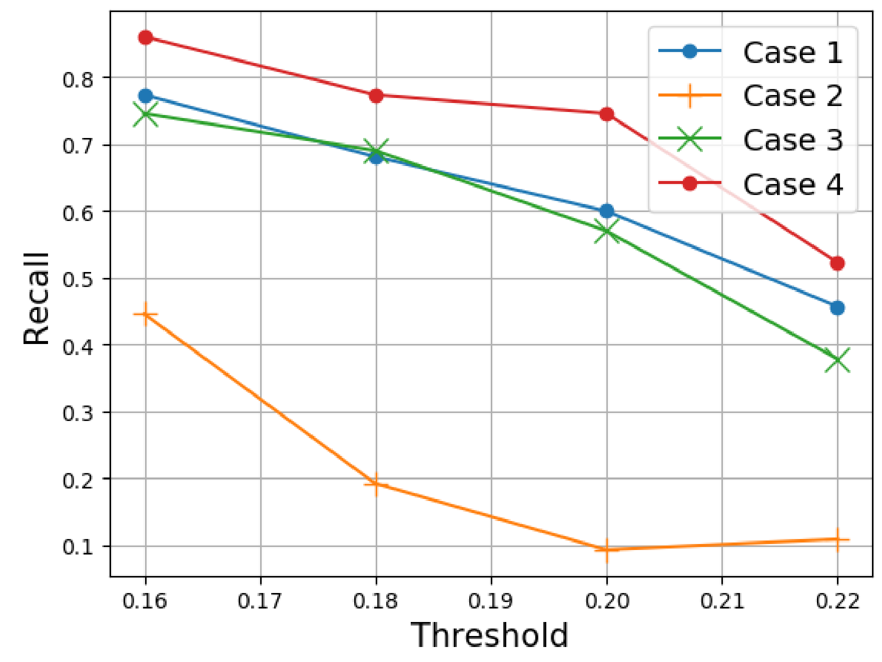


Figure 3.4: Recall vs Threshold

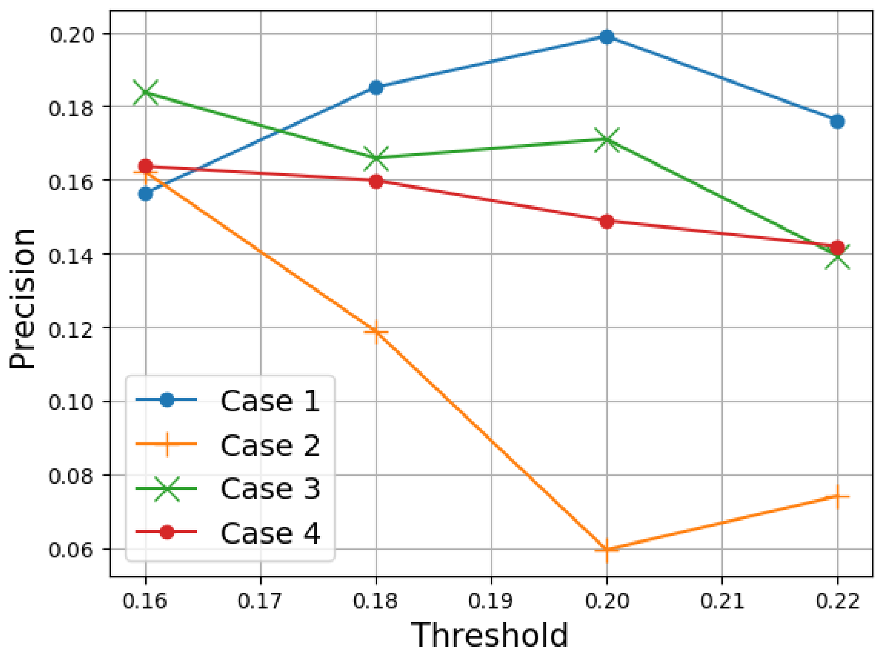


Figure 3.5: Precision vs Threshold

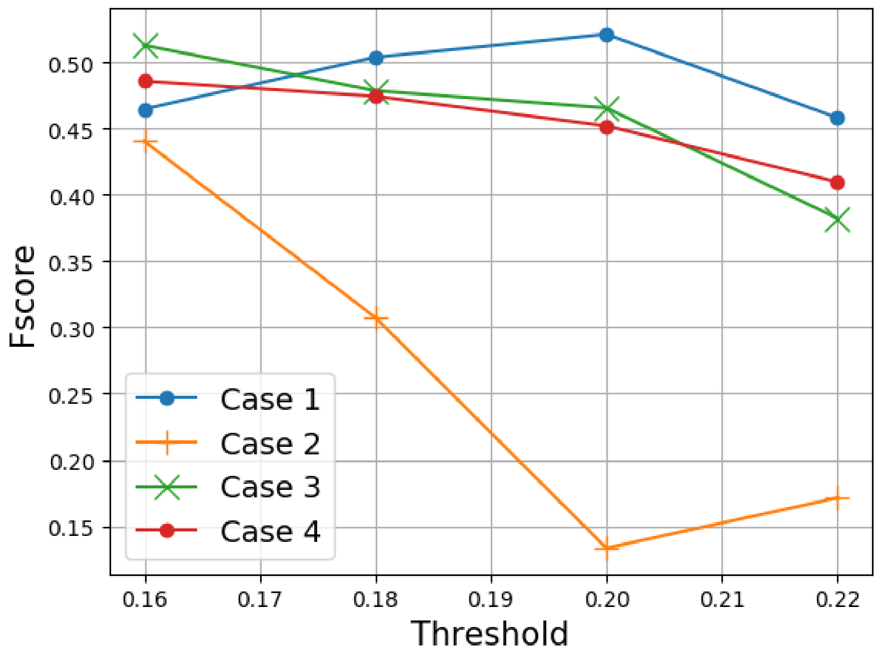


Figure 3.6: F-score vs Threshold

3.4 Achieving Layer Separation by Sequential training

Analysing the road-user interaction in real time can dramatically improve safety in smart-road environments. In the recent times this has become possible as new vehicles are being equipped with plenty of sensors that can provide real time information about the vehicle operations and diagnostics. This data is very helpful in assessing the driving behaviour and detecting anomalies which is an essential part of Intelligent transport systems. Given the fact that vehicle sensors can collect a variety of heterogeneous parameters such as speed, acceleration, fuel consumption, heading, etc, Machine Learning (ML) can play an effective role in analysing the driving behaviour, which could be helpful in detecting anomalies and thus, warning the driver timely. However, it is very demanding and crucial to build models for all the users which would be effective in all kinds of road conditions. In fact, it is obvious that every driver has her own driving style which would vary depending on the road-environment. Therefore, a novel *Vehicle to Infrastructure* (V2I)-based scheme to address the above issue efficiently is discussed in this section. This scheme is based on a training technique that partitions the Neural Network (NN) layers of the model into some layers specific of the user behaviour and others layers specific the road-environment. Therefore, in real time the vehicle user can just obtain the NN layers specific of the road-environment where the vehicle is currently located. More specifically, when a vehicle enters the communication range of a Road Side Unit (RSU), the vehicle will automatically download the specific road-environment layers and plug it with the user-specific NN layers.

3.4.1 V2I Scenario

A V2I scenario is considered in which the communication between the vehicles and the infrastructure is established by the well known IEEE 802.11p protocol. The IEEE 802.11p protocol is defined for *wireless access in vehicular environments* (WAVE) [113–115]. The WAVE systems basically comprises of two fundamental types of elements: the *Road Side Units* (RSU)s that are installed on traffic light posts or other road infrastructure elements and the *On-Board Units* (OBU)s mounted on the vehicles to establish communication with other vehicles and with the road infrastructure [116, 117].

The data for the ML training in vehicular applications is collected from smartphones' embedded sensors and On-Board Diagnostics (OBD) II units [118, 119]. The OBD II is a standard which led to the development of OBD II scanning tools that can interface to any vehicle via a 16 pin port [80]. The OBD II uses two different kinds of codes to request sensor data from the Electronic Control Unit (ECU). They are Parameter Identifiers (PIDs) and Diagnostic Trouble Codes (DTCs). PIDs are used to measure real time parameters such as Engine RPM, Vehicle speed, Fuel pressure, Throttle position and more. Whereas DTCs are used to diagnose malfunctions in the vehicle systems.

The On-Board unit receives the data from the OBD II for processing and ML training.

The IEEE 802.11p protocol aims to provide V2I communication in ranges up to 1 Km and supports data rate from 3 to 27 Mb/s [120]. Therefore, as shown in Figure 3.7, a vehicle communicates with a specific RSU when this is in the communication range of its OBU. Basically, RSUs are set up for broadcasting alert and other types of messages to vehicle users [121]. As we will see later, in our case the RSUs broadcast the scenario layers to the vehicle drivers. We denote the layers of the i-th RSU as S_i. Therefore, in the scenario shown in the Figure 3.7 there are two RSUs, denoted as RSU 1 and RSU 2 serving two areas and the corresponding layers are S_1 and S_2.

Any vehicle driver, say the j-th, has her own specific layers denoted as U_j.

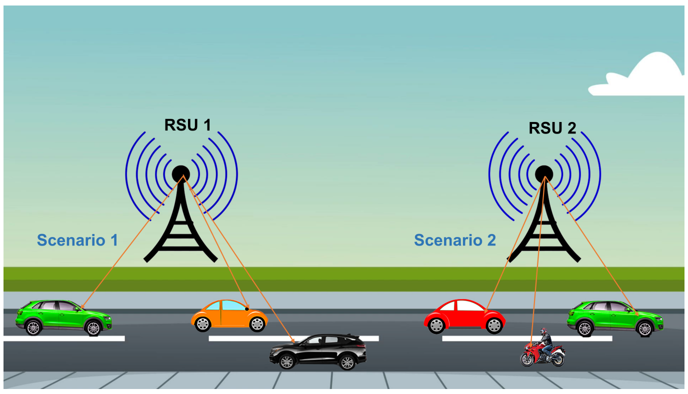


Figure 3.7: V2I scenario

Suppose she enters within the radio range of the i-th RSU, she will receive the layers S_i and thus can construct a neural network specific of the context as the concatenation of the layers S_i and U_j, i.e., $[U_j, S_i]$.

If the application is anomaly detection, the model $[U_j, S_i]$ is used by the vehicle user to detect anomalies. When the same vehicle enters a new area with different road conditions, for instance the area covered by RSU k, the vehicle user receives the layers S_k and plugs them with the user layers U_j to build the model $[U_j, S_k]$.

Training ML models in real time is always crucial as ML training can lead to significant delays. However in our case, there is no model training in real time. The scenario layers are just concatenated with the user layers to build the model. All the training is done in the start up phase and not in real time. We term the training at the start up phase as sequential training which is described in detail in Section **??**. The sequential training enables creation of NN layers that are specific to a scenario.

3.4.2 Sequential Training Algorithm

In the cases in which users interact with the environment, the environment gives input to the user who is expected to react with an appropriate action. Such action and the way in which it is executed depend on the user. For example, in the application domain of our interest, a curve in the road requires the driver to steer. The way in which the steering will be executed depends on the driver (and her current physical/mental conditions). The specific execution of the steering will result in patterns of values measured by the sensors deployed in the environment.

In Figure 3.8, we sketch the architecture for the neural network model that we envision. The collected data are provided as inputs to a set of layers which is specific for the user. The output of these layers will be a representation of

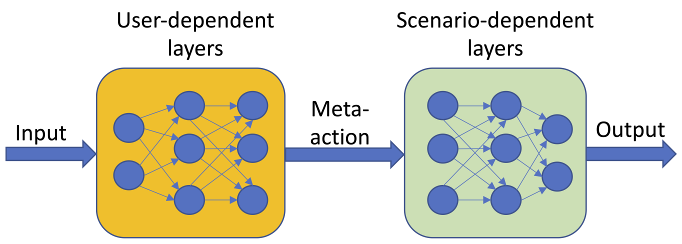


Figure 3.8: Neural network architecture.

the action taken by the user. Indeed, the output contains information going well beyond the mere action taken by the user and therefore, we refer to it as *meta-action*. The meta-action is given as input for the set of layers which are specific of the scenario. Therefore, the final output will be specific of the user in the current particular scenario.

The ultimate objective of the proposed scheme is to enable separate training of the user- and scenario-dependent layers so that it is possible to train:

- the user-dependent layers using the data collected for that user, say the j-th, in any scenario, and
- the scenario-dependent layers using the data collected for any user in that specific scenario, say the i-th.

In this way when user j visits the scenario i the corresponding neural network model will be constructed as the concatenation of the layers specific for user j, which we denoted as U_j, and the layers specific for scenario i, which we denoted as S_i. We explicitly observe that the scheme works even if it is the first time that user j visits the scenario i.

In order to achieve such a result, a procedure is required that automatically builds the interface between the user- and scenario-dependent layers. In other terms, it is necessary to build an appropriate vocabulary for the meta-action recognized by all users- and scenarios-layers. Once such an interface, called *meta-action interface*, has been built adding a new user requires the training of the user-dependent layers in one existing scenario only[1] and, consequently, the construction of the relevant dataset. Once trained, such layers can be used in any scenario.

The *viceversa*, obviously, applies, i.e., when a new scenario is defined the corresponding layers can be trained considering one existing user only.

We propose Sequential Training as the realization of the procedure that builds the interface between the user- and the scenario-specific layers.

[1]and the use of the corresponding scenario-specific layers

Sequential Training is an iterative process which is executed at the start up of the system and assumes that data have been collected for a certain number, say n, of users in a certain number, say m, of scenarios. We denote the dataset collected for user j in scenario i as $D_{j,i}$.

The basic idea of Sequential Training is to build the meta-action interface as follows. The layers specific for user j^*, with $j^* \leq n$, are trained by using the datasets $D_{j^*,i}$ for all $i \leq m$ and the corresponding scenario-specific layers, S_i, which are set as non trainable, we will say *frozen* in the following of the paper. Observe, that when the layers for user j^*, U_{j^*}, are concatenated with the layers for scenario i, which are set as non trainable, we are forcing U_{j^*} to learn the vocabulary utilized by S_i. Viceversa, the layers specific for scenario i^*, with $i^* \leq m$, are trained using the datasets D_{j,i^*} for all $j \leq n$ and the corresponding user-specific layers, U_j, which are *frozen*.

For simple understanding, the example shown in the figure 3.9 is considered. Assume that the sensor data was collected from 5 users in two scenarios: $D_{1,1}$, $D_{2,1}$, ...$D_{5,1}$ from Scenario 1 and $D_{1,2}, D_{2,2}, ...D_{5,2}$ from Scenario 2. As mentioned earlier and as shown in Figure 3.9, the user layers and scenario layers are frozen and trained on the respective data sequentially and in an iterative manner. The goal is to have a separate set of layers for each scenario. That is, to obtain trained scenario 1 layers S_1 and trained scenario 2 layers S_2. The sequential training for two scenarios is comprises of the following operations:

- Freeze the scenario layers in $[U_1\ S_1]$ and train on data $D_{1,1}$
- Freeze the user layers in $[U_1\ S_2]$ and train on data $D_{1,2}$
- Freeze the scenario layers in $[U_1\ S_2]$ and train on data $D_{1,2}$
- Freeze the user layers in $[U_1\ S_1]$ and train on data $D_{1,1}$

The above cycle is done for all the users. Doing this sequential process will result in trained set of layers specific to the scenarios. That is, we will obtain scenario 1 layers $S1$ and scenario 2 layers $S2$ and also the 5 user layers: $U1$, $U2$. $U3$,$U4$ and $U5$. However only the scenario layers are required to be deployed in the RSUs.

3.4.3 Numerical Results

The dataset and the ML algorithm for the experiments are the same mentioned in sections 3.3.2 and 3.3.1.

Algorithm 5 Sequential Training algorithm

```
1: /* Protocol initialization */
2: Load data-set $D_{i,j}$ where $i$=1,2,3...$n$ users and $j$=1,2..$m$ scenarios
3: while True do
4:     if $i \leq n$ then
5:         while $j \leq m$ do
6:             Freeze $S_j$ layers in [U$_i$S$_j$]
7:             Train [U$_i$S$_j$] on data $D_{i,j}$
8:             j=j+1
9:             Freeze $U_i$ layers in [U$_i$S$_j$]
10:            Train [U$_i$S$_j$] on data $D_{i,j}$
11:        end while
12:        while $j > 0$ do
13:            Freeze $S_j$ layers in [U$_i$S$_j$]
14:            Train [U$_i$S$_j$] on data $D_{i,j}$
15:            j=j-1
16:            Freeze $U_i$ layers in [U$_i$S$_j$]
17:            Train [U$_i$S$_j$] on data $D_{i,j}$
18:        end while
19:        $i$=$i$+1
20:    end if
21: end while
```

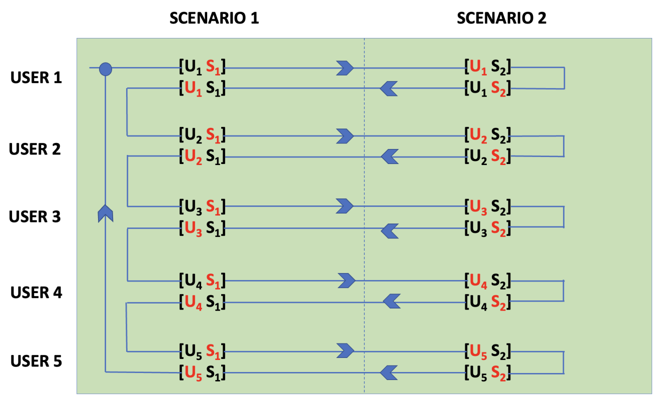


Figure 3.9: Start-up Training

In this section, we the numerical results are reported that are obtained by applying sequential training with the data of six users and testing the model on two users. The performance of the auto-encoder model is evaluated in terms of mean square loss. The mean square loss is evaluated from the data reconstructed by the autoencoder and the input data fed into the autoencoder according to the eq. (3.5), i.e.,

$$MSE(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y - \hat{y}_i)^2 \tag{3.5}$$

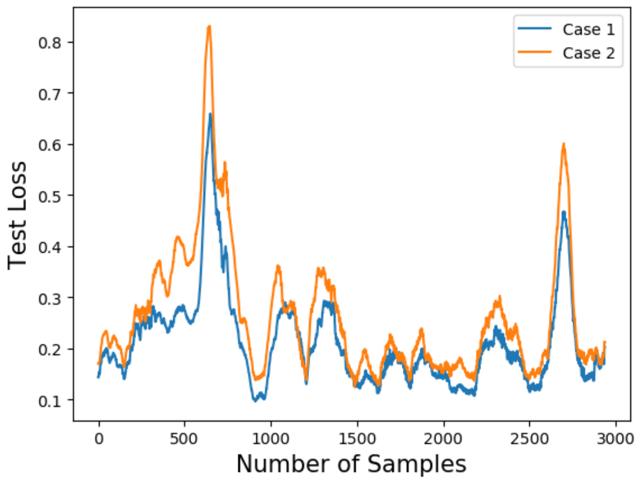
where y is the input data fed into the autoencoder, $\hat{y}$ is the reconstructed output data and N is the length of the input data.

The Sequential training was performed with 6 users in an interactive manner to obtain the scenario layers of scenario 1 $S1$ and scenario 2 $S2$.

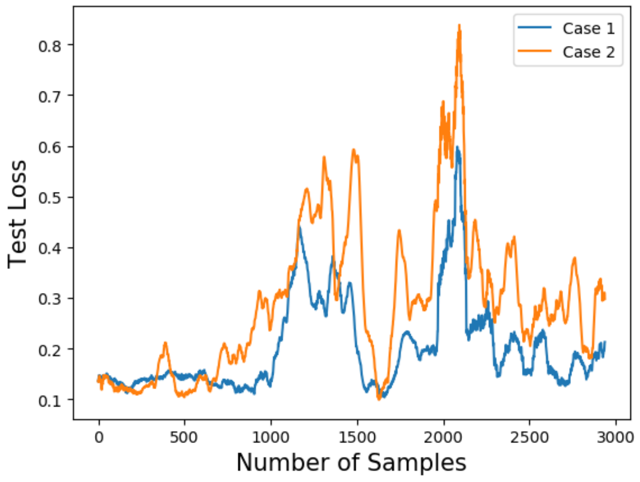
The following 5 cases are considered for the evaluation

- **Case 1**: This case is our proposed approach. Assuming that vehicle user 7 is driving in the scenario 2, user will receive the scenario 2 layers $S2$ from the RSU. The received $S2$ layers will be plugged in with the user layers $U7$ to build the model $[U7, S2]$.
- **Case 2**: Case 2 is the approach in which the vehicle user plugs in the wrong scenario layers instead of the right one. That is, if the vehicle user plugs the scenario 1 layers $S1$ with the user layers when the vehicle is in scenario 2.
- **Case 3**: Case 3 is supposed to give the best results among all the other cases since the model is trained and tested on the same data. Though it might give best results, it is a wrong approach of evaluation a ML model.
- **Case 4**: Case 4 is an approach in which the wrong set of layers is plugged in with the scenario layers. That is, vehicle user 3 in scenario 2 using the model [U1 S2] instead of [U3 S2]
- **Case 5**: Case 5 is similar to case 4 except the fact that the scenario layers are also the wrong ones. To simplify, both the user layers and the scenario layers are not the correct ones. That is, vehicle user 3 in scenario 2 having the layer combination [U1,S1] when it is supposed to be having [U3 S2].

Therefore, Case 1 is the proposed approach and also the ideal one. Firstly, We report the testing loss obtained in both Case 1 and Case 2 in figure 3.10. As seen in the figure, the test loss is lower for the Case 1 when compared to the



(a) User 7



(b) User 8

Figure 3.10: Average Loss vs Number of iterations for different values of Epochs

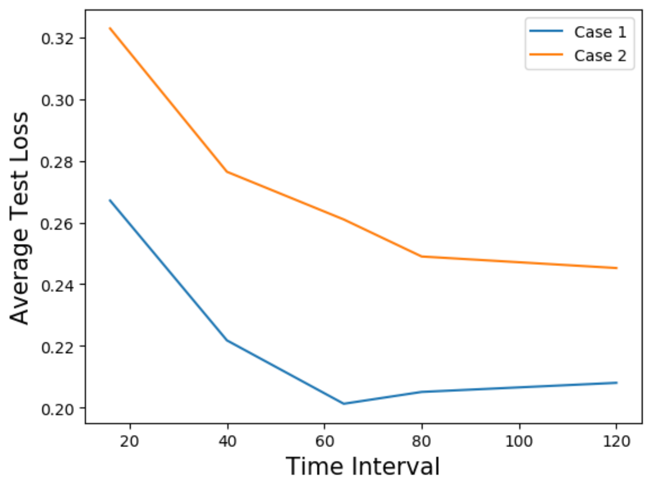


Figure 3.11: Average Loss vs Input-Time Stamp

Case 2. That is, the loss is lower when the correct scenario layers is plugged in with the user layers. Instead, the loss is higher when the wrong scenario layers is plugged in with the user layers. The evaluation was done in both user 7 and 8, that is evaluating the model $[U7, S2]$ on $D7Y$ and evaluating the model $[U8, S2]$ on $D8Y$. We observed similar performance in both the cases.

A similar evaluation was made between the right Case 1 and Case 2 by tuning the size of the input data fed into the auto-encoder. Since the data collected by the bicyclists is time series data, we varied the time interval and measured the average loss . Note that 10 samples correspond to one second. We report this comparison in figure 3.11 and as shown in the figure, the test loss decreases with increase in the time interval. However, the test loss for the right scenario is well below the wrong scenario case.

Finally, all the 5 cases were compared. We report the test loss obtained in all the 5 cases in figure 3.12. The average loss in the following table was reported. As seen in the table, the average loss of Case 3 is the lowest among all and it was expected to be the lowest since the training and testing is done on the same data. However, training and testing on the same data is not an ideal approach of evaluating a ML model. Case 1 which is our proposed approach has the next lowest loss. The average loss of case 1 is lower than case 2 because case 2 involves plugging in wrong scenario layers. Cases 4 and 5 have the highest average loss since they involve the usage of wrong user layers.

Therefore from the simulation results, it cab be inferred that the sequential training approach is able to achieve layer separation with satisfactory results.

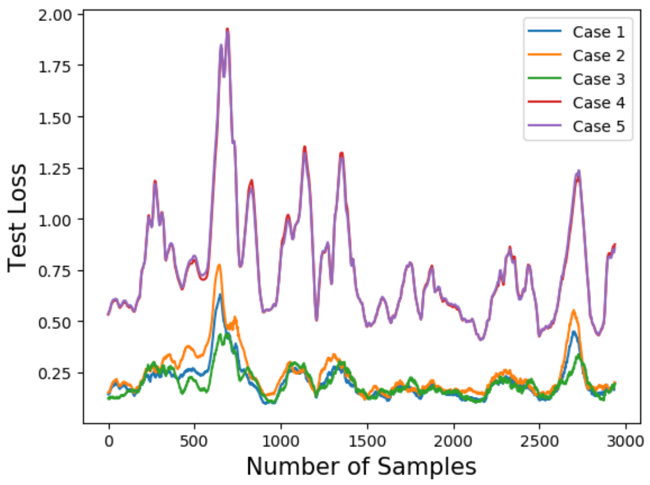


Figure 3.12: Average Loss vs Number of Samples

Cases	Average Test Loss
Case 1	0.20703
Case 2	0.25519
Case 3	0.19732
Case 4	0.764707
Case 5	0.765562

As it is not an ideal approach to use a common ML model for all kind of environment scenarios by every users, the objective was to partition the ML model into separate layers for the users and separate layers for the scenario. In this work, a sequential training approach has been introduced for achieving layer separation in deep neural networks. More specifically, a layer separation approach that can be beneficial in exploiting ML in road-user interaction applications in smart road-environments. This approach was applied on a specific case study in which data was collected from several bicyclists riding along the same path. Numerical results shows that the proposed approach is efficient in achieving layer separation in ML models.

Chapter 4

Internet of Underwater Things

4.1 Introduction to Underwater Test-beds

This chapter discusses the design and development of a hybrid underwater/LoRa testbed. The Internet of Things (IoT) paradigm is nowadays pervasively integrated in our life, while its underwater counterpart, the so called Internet of Underwater Things (IoUT), is still at its infancy and the idea of monitoring and interacting with underwater devices in the ocean is yet to flourish [122]. An expected boost in the underwater research is foreseen and motivated by the fact that more than 90% of the ocean seabed is unexplored. This kind of exploration can have a relevant impact on the sea economy, which approximately contributes as $1.5 trillions annually to the overall world economy. Therefore, it is evident how the success and development of underwater research can play a key role in the evolution of human society. In spite of the architectural similarities exhibited by an underwater communication system as compared to a terrestrial one, numerous criticalities emerge when we move from a "dry" scenario to a "wet" and significantly variable one, where attenuation can be extremely high and radio frequency waves result not the best choice to support communications. In this case, the channel model is highly time varying and unsufficiently modeled as compared to a terrestrial one; high propagation delay, far longer than the one experienced in the terrestrial network, fading, narrow bandwidth and Doppler effect make communications much more complex in these settings. Numerous are the application scenarios where the use of underwater communications can result of paramount importance: just to mention a few, seabed analysis, coral evaluation, marine plant identification, fish recognition, plankton and lobster tracking, fish farming monitoring, tsunami and flood warning systems, as well as submersed cultural heritage preservation [123, 124]. However the possibility to set up a robust communication system which allows to monitor a given sea interest area, relies on the ability to develop an underwater network to successfully deliver the collected information remotely to a sink or collection center which can be located on-shore. On the other hand, devices in the network should be tunable and addressable by means of a two-way communication protocol which allows to contact and actuate remote underwater devices.

Recently, some efforts in the direction of developing underwater networks have emerged. In [125], design, development, and testing of a Smart Buoy system are presented with a specific focus on the support of real-time remote access to underwater devices, with an interest in the use of energy harvesting mechanisms which exploit solar panels to cope with energy constraints. Similar activities have been carried out in the framework of the SEANet project, where high-speed acoustic modems are deployed for supporting short range underwater communications [126]. Many companies also provide commercial products, like monitoring buoys, which are somehow envisioned to be deployed in controlled areas where the collected data can be delivered via radio connections (GPRS, 3G) to a web-based software. This makes, however, their use unfeasible in open ocean areas. Among the providers of floating equipment, Axis Technologies offers products for environmental applications, equipped with air temperature, barometric pressure, or wave height sensors. Another buoy provider is [127], which produces different heterogeneous types of devices, but again interaction with these proprietary devices is not trivial and no flexibility or interoperability among different producers can be provided. Examples of data buoys not developed by product providers but, instead, by universities and research centers are The “Meduza buoy” [128] and the above mentioned "Smart Buoys", [125] developed at Northeastern University in the framework of relevant research projects. Each of these systems focuses on a specific task like, for example, the development of the chassis equipped with solar panels for devices recharging.

In the following a hybrid underwater-terrestrial testbed is described that have been developed and tested in both the premises of University of Catania (aquarium) and in the Catania harbor area, and which implements an Underwater IoT (UIoT). More in detail, the developed system consists of two sub-systems, a pure underwater sub-system, and a terrestrial sub-system. In the underwater sub-system, multiple underwater sensors collect measurements on the marine environment and send this information by means of underwater acoustic modems through an underwater marine link to a receiving edge underwater modem. The terrestrial suub-system consists of the edge underwater modem attached to a buoy

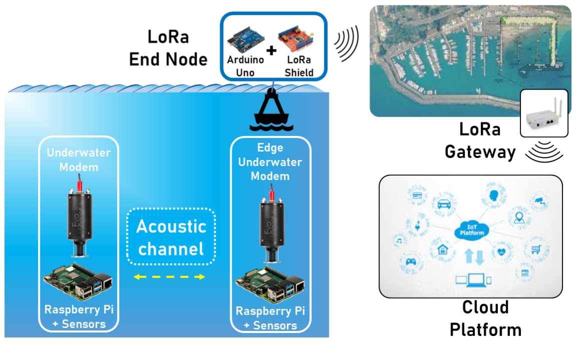


Figure 4.1: The IoUT System

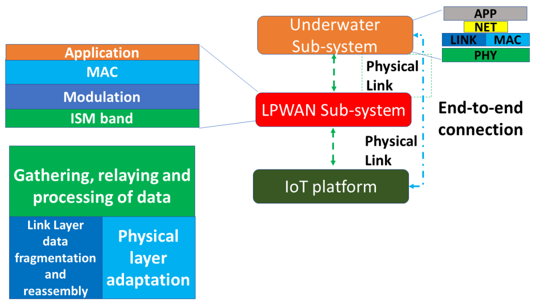


Figure 4.2: Functional View of the System

where a surface LoRa node is connected by way of a LoRaWAN connection [129, 130] to a terrestrial remote LoRa Gateway, which, in turn, remotely interacts with the cloud by means of an Internet connection. In the cloud, this marine data can be saved and analyzed for further processing and/or usage. Data can be retrieved from the cloud by means of an Android app which has been properly designed for the purpose of this project. Also, a Web App has been designed and developed for similar purposes, while also allowing a historical query of data. Note that the IoUT system is highly configurable since the transmission power and other sensing parameters can be remotely tuned (e.g. image and video quality tuning, camera rotation angle, modem transmission power, etc.). This makes the system the first example of a really dynamic and reconfigurable IoUT network.

Numerical results have assessed the effectiveness of using the developed equipment to proficuously send data, images and video with low data rate.

4.2 System Description

In this section, a functional description is provided about the overall system which implements the proof of concept for an Internet of Underwater Things. As shown in Figure 4.1, it consists of two sub-parts: The system has been specifically designed to implement the vision of an Internet of Underwater Things. In this respect, two sub-parts of the system can be identified (see Figure 4.1):

- An *underwater sub-system*, which includes multiple heterogeneous units, such as those performing sensing activities (i.e. the sensors), those executing data transmission (i.e. the underwater acoustic modems), as well as the equipment for power provisioning and control. All this equipment is safely sealed inside a hermetic enclosure. A control board takes care of controlling data collection from the sensor devices; also, it performs collection and sends the collected data by way of the underwater transmission equipment, possibly employing multihop communication among the relay nodes. Data are directed to an edge underwater modem placed under a buoy. This underwater sub-system is fully independent and does not need any cable to connect to the terrestrial section of the system. As regards the sensing activities, different types of data can be collected, thanks to the availability of multiple types of sensors:
  - A Depth/Pressure Sensor, which allows to estimate the depth at which, for example, either a marine wildlife or a shipwreck are located;
  - A camera, which allows to collect visual information, for example, on submarine wildlife environment or on the degree of deterioration of a shipwreck, possibly covered by sediments;
  - A Light Sensor, which estimates the illumination level at a specific depth, so as to get useful information for the support, for example, of recovery campaigns for shipwrecks.
- A *Low Power WAN (LPWAN) sub-system*, which includes two sub-units, i.e. a long range communication node (LoRa End Node) with an embedded control board positioned on a buoy, and a LoRa Gateway which takes care of forwarding the data received from the underwater network section to a remote IoT platform and/or a network server by exploiting terrestrial connections.

Several functionalities have been included in the overall system, as shown in Figure 4.2. Among those, the most relevant ones are:

1. **Gathering, relaying and processing of data**: The control boards manage the available sensors and are able to activate the measurement operations, either periodically or upon reception of a proper command sent remotely. The control boards can also provide a service of data relaying when

needed, in case of multi-hop routing. Moreover, the boards can be requested to process the gathered data (e.g. compressing or scaling the images and videos).

2. **Link Layer data fragmentation and reassembly**: In case data generated by sensors have to be converted and made suitable for transmission over the underwater channel, a data-link fragmentation function, which splits packets into several small frames, is used. For instance, the length of frames may be varied according to the underwater channel state (short data link frames are recommended when the channel is in a bad state and the channel response varies rapidly);

3. **Physical layer adaptation**: Once data are properly fragmented and formatted, they are ready for transmission on the physical channel. To this purpose, signal processing and transduction are needed to transmit/receive the frames as acoustic waves traveling on the underwater channel. Likewise, data format adaptation is required at the gateway node to allow data transmission over the LPWAN sub-system.

4.3 Experimental Setup: Underwater Sub-System

The underwater sub-system consists of three main components:

- **The sensing and actuation components** include the sensors and actuators aimed at monitoring the underwater environment and the operational system conditions. In our setup, the sensing devices are:

  1. A Blue Robotics [**BR**] Bar30 High-Resolution Depth/Pressure Sensor, used to monitor the operating depth of the modems;
  2. A Blue Robotics [**BR**] Low-Light HD USB Camera, used to record pictures and videos of the underwater environment.
  3. A UUGear [**UUGEAR**] Light Sensor Module, used to measure the light intensity in the surrounding environment, e.g. to test if the illumination conditions are suitable to capture images and videos.

  The actuation part, instead, includes a servo motor, to control the orientation of the USB Camera up to a maximum rotation angle of 180°. This feature introduces a high degree of freedom in the acquisition of videos and images under the sea.

- **The control component** handles the sensing and actuation part, either autonomously or in response to remote commands, and is also responsible for the management of the data transmission and forwarding operations. Among the functionalities supported by the control component, there is the capability to reset the devices, to control the transmission power, and to tune the frame length.

  The control part also manages the tuning of several parameters in the sensor devices, such as the camera resolution, the compression ratio of collected pictures, the frame rate of acquired videos, and the light sensor sensitivity.

  The employed hardware consists of a Raspberry Pi board, and a battery pack to power the sensors and the board itself.

- **The transmission component** consists of two Underwater Evologics Modems S2C 18/34 [**evologics**]. These devices represent the core of the underwater network, and are used to convey the sensed data towards the edge underwater modem, and to eventually forward the received information to the cloud platform through the LPWAN sub-system. From an operational point of view, the modems employ acoustic waves for communication across the underwater channel, and are powered by 24 volts supply batteries. Figures 4.3 and 4.4 show pictures of the acoustic modem together with the control equipment inside an underwater hermetic enclosure, taken on the pier and during tests, respectively. The enclosure safely stores the control component, the sensing and actuation components, and the power supply.

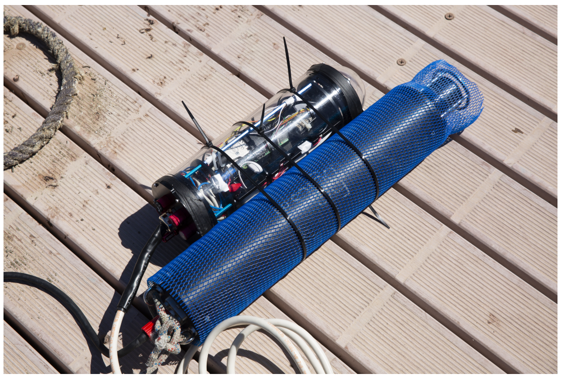


Figure 4.3: The Underwater sub-system components.

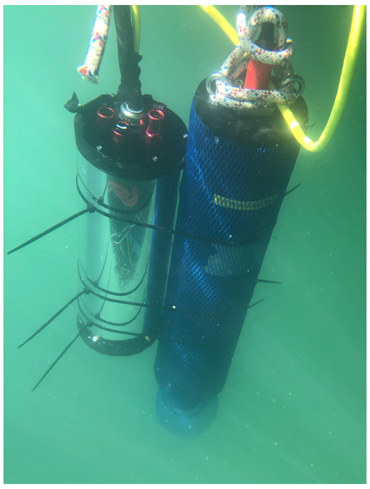


Figure 4.4: The Underwater components during marine tests.

Architecture of the Underwater System

In the following, the protocol architecture of the underwater sub-system is thoroughly.

Physical layer The physical layer of the underwater sub-system is based on a patented S2C (Sweep Spread Carrier) protocol [**kebkal**]. The basic idea of the protocol is using a series of frequency sweeps as a carrier signal to convey information over a communication medium. The transmitted signal is first encoded as a PSK signal, and, then, modulated on the sweep spread carrier. The main advantage of using the S2C protocol consists in the frequency split of signal multi-paths. Hence, it is possible to heavily reduce the multi-path effect, and, consequently, to easily counteract the inter-symbol interference.

Link layer Operation and functionalities of the data link layer rely on a proprietary protocol, namely D-MAC [**kebkal**]. D-MAC is specifically designed to take into account the long propagation delay in the underwater channel. Indeed, in case of large data volumes (the so-called *burst data*), the system employs an efficient mechanism that interleaves confirmation acknowledgments and data packet transmissions. Also, an adaptive transmission mode allows to tune transmission parameters to support the maximum feasible bit rate, depending on the current channel conditions. In case of short messages (denoted as *instant messages*), whose size is limited to 64 bytes, a constant transmission bit rate is employed. Note that these short message can be sent either individually, or together with an ongoing burst data transmission.

The transmission of burst data starts with an initial handshaking procedure to estimate the channel conditions based on the calculation of the round trip time. Accordingly, the protocol automatically improves the transmission performance by choosing a suitable frame size.

Moreover, while in burst mode, data frames are grouped into *clusters* of variable size (tunable by the user) to support the usage of a cumulative acknowledgement mechanism. Frames are eventually reassembled into the original data and finally forwarded to the input/output receiver interface.

Instant messages can instead be employed for different purposes: regular *instant messages* are transmitted immediately and can be used to optionally request packet acknowledgements; *synchronous instant messages* are time-triggered messages; *piggyback messages* are transmitted together with other data, such as instant messages or burst data. No procedure of connection establishment is needed to transmit instant messages.

Network layer The main role of the network layer is to support device addressing functionalities. More specifically, every device has its own address, which can be controlled by the user[**kebkal**] and employed for transmitting both burst data and instant messages. Furthermore, the network layer implements an efficient Backoff mechanism to handle conflicting simultaneous transmission: when the device detects a collision, the node enters the so-called **Backoff State**, and a **Backoff Timeout** is activated.

The network layer supports various transmission modes. Besides unicast, a broadcast modality is available for instant messages, and allows a device to reach all the nodes present in a given coverage area. Moreover, a device can overhear messages intended for other nodes thanks to the so-called **Promiscuous Mode**. Finally, the network layer implements the mechanism of **Extended notifications**, to provide extra information about transmissions and receptions, as well as changes in the device settings. Thanks to the above functionalities, it is possible to configure several aspects related to data forwarding and delivery, for both single-hop or multi-hop transmissions.

Since the underwater network deployed for our experimental work consists of only two nodes, no routing protocol has been used. However, in case of extension of the network to multiple nodes, it is possible to design and employ one or more underwater routing protocols to select the best path for data delivery to the edge underwater modem [**routingUW1**, **routingUW2**, **routingUW3**].

Application layer The application layer has been specifically designed to support the gathering and processing of the heterogeneous sensor data collected in the underwater sub-system. As already anticipated, the different types of data collected include images, videos, temperature, light intensity, and pressure values. Data collection can take place either in a fully-automated fashion, at specific time intervals, or upon receiving an appropriate **data_collection (type, parameters)** command sent by an end user. The user, by means of an Android and/or a web application, is also able to control every functionality of the underwater network. To this purpose, we have implemented several specific commands, such as the **restart (sensor)** command, and the **tuning (sensor, parameter, new_value)** command, to allow access to the sensor parameter settings. These commands trigger the activation of a set of procedures to control the sensing devices. In case of images and video transmission, the images and videos are converted into Base64 strings and then transmitted remotely by using a LoRaWAN communication link. The Base64 strings are split into smaller payload packets with the size of 222 bytes. This choice is motivated by the high packet losses that we experienced for larger payload size. As an example, the pseudo-code reported in Procedure 6 illustrates the procedural steps for requesting the transmission of an image and appropriately encoding and compressing it.

Algorithm 6 Procedure 1: Image Transmission

```
1: procedure Image Transmission
2:     Open serial connection to the modem
3:     $photo \leftarrow Take\_Picture()$
4:     $Scale\_Picture(photo)$
5:     $Apply\_Compression(photo, compression\_factor)$
6:     $photo \leftarrow base64encoding(photo)$
7:     $photolen \leftarrow$ length of $photo$
8:     $startCode \leftarrow$ 0x11
9:     $endCode \leftarrow$ 0x10
10:    $message \leftarrow startCode + photolen + photo + endCode$
11:    Write $message$ on the serial connection to the modem
12: end procedure
```

Concerning the procedure for data sensing and transmission, the control boards can control each individual sensing device through a **Take_measurement(sensor)** command and, accordingly, acquire one or more measurements. Additionally, to cope with the low data rates due to the variable underwater channel conditions, proper compression techniques for large sized data, e.g. pictures and videos, have been implemented. In the specific case of pictures, the function **Scale_Picture(photo)** is first applied to scale the data to an arbitrary resolution, followed by the function **Apply_Compression(photo, compression_factor)**,

Algorithm 7 Procedure 2: Image Reception

```
 1: procedure IMAGE RECEPTION
 2:     Open serial connection to the modem
 3:     Wait for messages on the serial port
 4:     $message \leftarrow serial.readMessage()$ 0x11 not in $message$
 5:     initialize $photo$ to an empty string
 6:     Extract and save $framelen$ from the message
 7:     Wait for messages on the serial port
 8:     $message \leftarrow serial.readMessage()$
 9:     $photo \leftarrow photo + message$ 0x10 not in $message$
10:     $photo \leftarrow base64decoding(photo)$
11:     save $photo$ to local drive
12: end procedure
```

to provide proper data compression.

Once the sensing procedure is completed, the actual communication phase can start. Hence, the control board sends a message to the intended underwater modem through a dedicated serial connection. The message includes:

1. **The start code 0x11**, to help the receiver detect the beginning of the message;
2. **The data length**, a useful information to track the number of remaining bytes to be received;
3. **The encoded data**, which is the actual measured or collected data;
4. **The end code 0x10**, which marks the end of the message.

Conversely, once the transmission is over, the receiver has to reconstruct the original packet from the received frames. As an example, let us analyze the procedural steps executed for image reception, as shown in Procedure 2.

The first step is to open a serial connection with the underwater modem. In the second step, the device remains in a listening state and waits for the start code 0x11. Then, the node can start gathering the frames received until the end code 0x10 is found. The next steps are to decode the received data and save them in the local storage.

4.4 Experimental Setup: LoRa Sub-System

This section illustrates the functionalities implemented in the LoRa sub-system. We first provide some basic information on LoRa technology and, then, the main features and characteristics of the LoRa sub-system are detailed.

LoRaWAN (Long Range Wide Area Network) is a low-power network protocol stack developed by the LoRa Alliance, and built on top of the LoRa physical layer developed by Semtech. The LoRa technology is designed as an energy-efficient, long-range and secure solution for Low-Power Wide-Area Networks. LoRa can drastically increase the network lifetime by transmitting only few data at a low data rate, and by employing a very low transceiver duty cycle, in the order of 1% [129, 130]. It employs a non-licensed frequency range and limits the TX/RX power to 25 mW.

The physical layer of the LoRa technology relies on a spread spectrum modulation technique to enable long range data transmission at low data rates. More specifically, LoRa is a radio modulation technique derived from the Chirp Spread Spectrum (CSS) methodology, and relying on linear frequency-modulated chirp pulses to encode information. The raw information signal is multiplied by the chirp pulses, and generates the actual widespread signal. LoRa supports different modulation Spreading Factors (SFs), ranging from SF7 to SF12. The Spreading Factor is a relevant parameter in LoRa, as it is possible to tune this parameter to impact on the maximum communication range and transmission data-rate. The SF influences the number of chirp pulses per symbol, N_c, according to the relationship

$$N_c = 2^{SF} \tag{4.1}$$

Hence, the largest the SF, the higher the symbol time or, equivalently, the smaller the transmission bit rate. At the same time, a higher number of chirps per symbol increases the so-called *processing gain*, resulting in an improved receiver sensitivity and robustness to noise, and, therefore, in a longer transmission range. Hence, the performance of the system can be properly tuned to extend the coverage range at the expense of the transmission data rate (bigger SFs), or, vice-versa, to increase the throughput at the cost of lowering the transmission distance (smaller SFs). Conveniently, the SF setting, together with the node transmission power, can be adaptively tuned according to the current channel conditions.

LoRaWAN sits on the LoRa physical layer and defines the upper layers to allow bi-directional communication among the LoRa nodes and the LoRa gateways in the communication range. The LoRaWAN architecture includes a back-end and a front-end. The former is embodied by the remote cloud server in the cloud, and stores the data received from the LoRa end-nodes. The front-end, instead, includes both the LoRa end-nodes, and the Gateway(s). The latter relies (rely) on an IP connection to act as a bridge between the network server and the end device. The LoRa data packets exchange between the LoRa end devices and the Gateways is instead based on a wireless transmission in the unlicensed ISM frequency band. The specific operating band depends on the geographical area. For instance, LoRaWAN communications in Europe are based on the 863-870 MHz ISM Band, with 3 main channels centered respectively around 868.10 MHz, 868.30 MHz, and 868.50 MHz. The LoRaWAN technology supports a transmission data-rate between 0.3 and 50 kbps, and also implements an adaptive data rate mechanism to trade energy consumption and data-rate.

In our experimental setup, the LoRa sub-system consists of: i) a LoRa End Node (Figure 4.5) composed of an Arduino Due board with a Dragino LoRa shield on top, connected to the edge underwater modem through a serial connection, and placed on a buoy floating above the water surface; ii) a Dragino LG02 LoRa Gateway (Figure 4.6) which forwards the data received from the LoRa End Node to an MQTT broker.

The interaction of the LoRa subsystem with the cloud platform is realized by means of the MQTT messaging protocol. MQTT is a lightweight publish/subscribe messaging protocol that provides a scalable and reliable way to connect devices over the Internet. In just a few years, it has become the de-facto standard for IoT messaging. The MQTT protocol is based on the coexistence of three entities: the *broker*, the *publishers*, and the *subscribers*. An MQTT broker receives data messages from the publishers, and dispatches those messages to specific MQTT clients, according to a *topic*-based mechanism: when a client is interested in a specific type of messages, it can subscribe to the corresponding topic. Hence, every message sent by a publisher on that same topic will be delivered to the client accordingly. An MQTT message consists of several fields, other than the payload, such as:

- *Client ID*: a unique identifier for each client;
- *Topic*: a UTF-8 string used by the broker to filter and dispatch the received messages;
- *Message*: the actual payload data;
- *Username* and *Password*: the authentication credentials for the MQTT broker. The credentials are sent in plain text, and can be protected by way of the TLS protocol.

In our scenario, the publishers are the underwater modems, while end users act as subscribers to retrieve data.

To allow the users to access the received data, we have developed an Android application and a Web application, as detailed in the following sections. Both the applications can subscribe to the MQTT broker and receive all the data sent from the underwater sub-system. Moreover, the applications offer the possibility to send specific commands in order to either request sensor data or tune some parameters in the underwater monitoring system, such as image and video quality, transmission rate, transmission power levels, or camera rotation angle.

The LoRa Gateway, located on the terrestrial area few hundreds of meters apart, receives the sensor data, images and videos, and, in our setup, forwards this data through the MQTT protocol by employing a 4G cellular network. More in detail, the data are forwarded to an MQTT broker hosted in a virtual machine on a server located in the premises of the underwater laboratory at the University of Catania. On the end users side, the above mentioned Android and Web applications subscribe to the MQTT broker through appropriate topics in order to request and get the data, or send commands in downlink.

We have also developed an SQL database to store the sensor data received by the MQTT broker. Hence, when the applications request an update from the underwater sensors, the most recent data in the database are forwarded to the application by the server. If there is no recent data in the database, a request command will be forwarded to the underwater system, and the data will be collected in real time. This approach reduces the transmission time in the system. The request commands and the broker topics are discussed in detail in the next sections.

Furthermore, for the sake of reliability and data persistence, a *Digital Twin (DT)* of the underwater devices has been implemented in JavaScript.

The DT is hosted together with the MQTT broker and SQL database, and, thanks to the support of both, manages the different data requests coming from the users.

4.5 Android application

As mentioned in the previous section, in order to support the remote control of underwater sensors and actuators, an Android application has been developed to exchange data and commands from/to the underwater system through the MQTT protocol. The application has been implemented in Java through the Android Studio development environment. Each button on the application interface allows the end users to request several types of sensor data, such as pressure, temperature

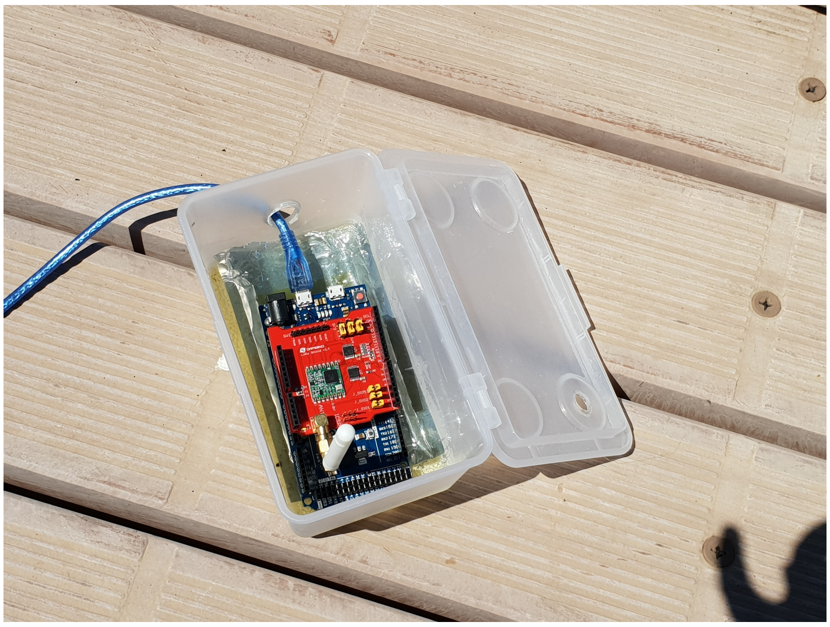


Figure 4.5: LoRa End Node

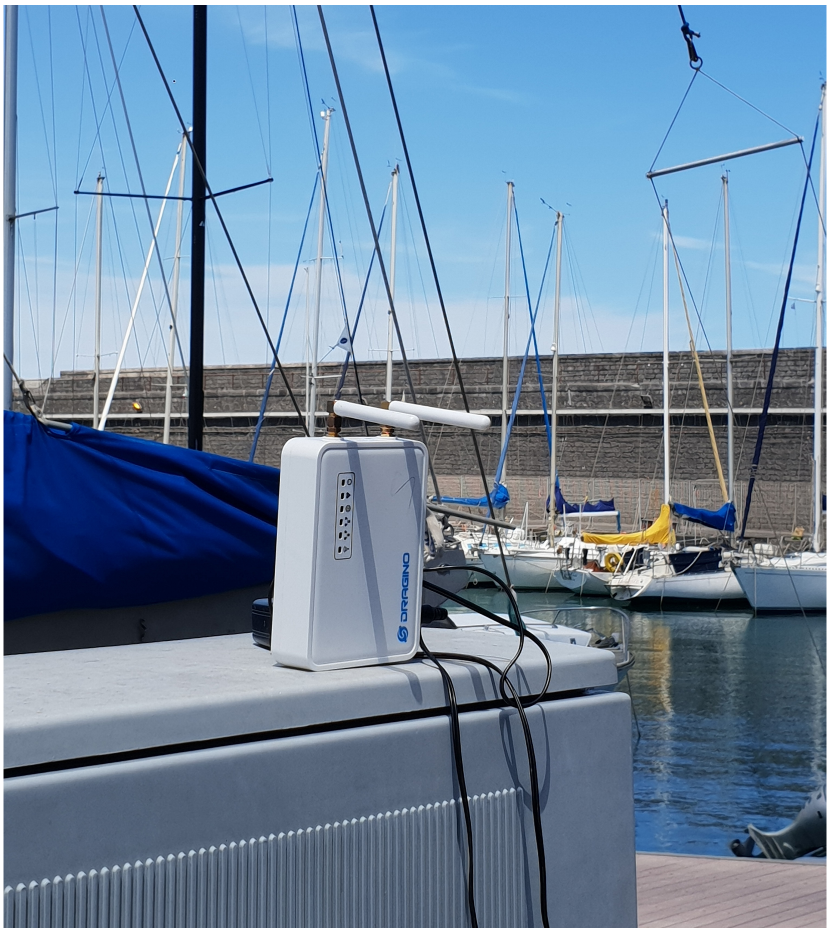


Figure 4.6: LoRa Gateway

and light sensor measurements, together with videos and pictures.

In the initialization phase, the Application interacts with the MQTT broker, and subscribes to a public upstream topic defined as: **dragino-1bc658/ChannelID1/data**. Data associated to this topic are periodically updated by the underwater sensors. In order to distinguish between each specific user connected to the MQTT broker, the application assigns a unique client_id to each user. All requests and responses are managed by the Digital Twin hosted in the Server together with the Database and the MQTT broker. These requests will be sent by the application to a downstream topic of the type: **downlink**, and directly addressed to the Digital Twin.

Data can be sent in two possible ways:

- **Proactive methodology**: Periodically, the data from the underwater system are sent to the appropriate topic, in order to allow the application to update the data stored in the database. Thanks to an **identification tag**, it is possible to distinguish which underwater device has generated what type of data, according to the format **(data type identifier)_(device identifier)(data value)**. Once the data have been received, both the source modem and type of data are shown through the usage of a TextView box in the application.

- **Reactive methodology**: The user explicitly requests data through the Android application interface. Accordingly, the application sends a command to the downstream topic, and the Digital Twin answers with the requested data. The issued command identifies the destination device, and the type of requested data, according to the format **(device identifier)_(data type identifier)**. This methodology relies on the usage of a temporary upstream topic of type **dragino-1bc658/Channel_ID2/data**. After the requested data have been received, the application unsubscribes from the temporary topic. This mechanism is adopted for a simple reason: since the downlink topic is shared among the end users, the requested data are broadcast to all the users associated to that topic. Hence, unsubscribing is necessary to avoid receiving unwanted/unrequested information.

In case of multimedia data requests, such as pictures and videos, the procedure works in a slightly different way. Users are able to request this type of data according to two possible quality levels: low quality, and high quality. Accordingly, the corresponding command identifies i) the destination device ii) the data type, iii) the requested quality level, through the format **(device identifier)_(data**

type identifier),(quality).

As thoroughly described in the previous sections, multimedia data are converted into Base64 strings, split into several smaller data packets, and finally sent in the public uplink topic by the LoRa End Node. The first string reports the special tag **START(modem identifier)**, to identify the source device and to signal the beginning of the process of data reception. The tag is followed by a parameter indicating the total number of data blocks to be received. Hence, the application can continuously track the status of the ongoing procedure through a visual **ProgressBar**. A further delimiter, **DELIM**, precedes the actual data block. Once the delimiter tag **DELIM** has been received, the application starts gathering the actual data packets. The end of this acquisition phase is identified by a further tag of type **END(modem identifier)**. Finally, the application reassembles the received packet into the original Base64 string, decodes the data, and saves it in the local multimedia gallery of the smartphone device.

The Android application also allows to perform actuation. For instance, it is possible to specifically manage the device transmission power level i.e. low, medium and high power levels (buttons L, M and H in the application interface). The corresponding alphanumeric command is in the format **(device identifier)_(data type identifier),(transmission power)**. Moreover, the application offers the possibility to control the rotation of the underwater camera devices, by means of a SeekBar. Thanks to the bar cursor, the user can tune the camera rotation angle at steps of 5°. Figure 4.7 illustrates the Android application interface with all the buttons available to request data from the different underwater modems, as well as an example of camera rotation setting. The alphanumeric command associated to this action identifies the destination device, the type of data, and the requested camera angle, according to the format **(device identifier)_(data type identifier),(angle)**.

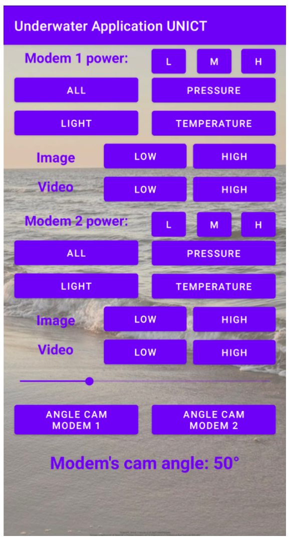


Figure 4.7: Setting camera angle using the Android application.

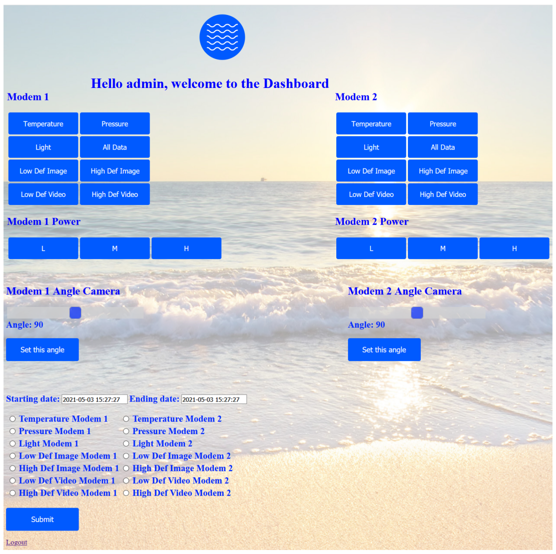
4.6 Web application

As a way to offer a complementary alternative to the Android app, a web application has been developed to extend and integrate the set of functionalities provided by the app.

The web application has been implemented in HTML and CSS languages for the front-end part, and in PHP for the back-end part. Two different databases were used: one for user management and one for data storage. At the startup, the web application displays a proper login page. From here, new users can access a registration page to create a new account by choosing a username and a password. The login process is managed via MySQL queries to the user database, and allows to enter credentials on the login page and access the dashboard (see Figure 4.8). The latter is basically divided into two equal-sized columns, one filled in with data about Modem 1, and the other with data regarding Modem 2.

The web application implements several features, as detailed in the following:

- **Collection of temperature, pressure and light sensor data**: These features allow the user to request the most recent temperature, pressure or light sensor data from the database. Every time a measurement is requested, the web application sends a SQL query to the database with a specific format, according to the type of data requested. When a query is issued, the system searches the database for the most recent data in a time interval of 20 minutes. If no recent data is available, a new data request is sent via the MQTT protocol to the underwater devices. The Digital Twin is then



Hello admin, welcome to the Dashboard

Angle: 90

Angle: 90

Starting date: 2021-05-03 15:27:27 Ending date: 2021-05-03 15:27:27

Figure 4.8: Dashboard of the web application

able to retrieve the new data, insert them in the database, and make it available to the web application.

- **All data request**: This functionality implements a simultaneous request of temperature, pressure and light sensor data for a given underwater device. A time constraint of 20 minutes is still considered to establish the freshness of sensor data. If even one type of data in the database is not recent enough, a re-send request for all the three data types is issued via the MQTT protocol.

- **Images and Videos request**: Similarly to the Android application, the Web Application implements the possibility to request videos and images from the underwater system, either in low or high quality. Using one of the four modem buttons (see Figure 4.8), it is possible to visualize the latest data from the database in chronological order. When the requested data is found in the database, it is first converted from Base64 to jpeg (or mp4, depending on the type of requested data). Then, the user is invited to download a copy of the file. In case the image or video is not available in the database or the available data does not satisfy the 20 minutes time constraint, the web application issues a new data request to the underwater system through the MQTT broker.

- **Transmission Power setting**: The web application includes, for each underwater modem, three buttons to remotely control the transmission power level. The levels are “Low”, “Mid” and “High”, and can be set by publishing the corresponding command on the downlink topic through the MQTT protocol.

- **Camera rotation setting**: This functionality controls the rotation angle of the underwater cameras. This function has been implemented using an HTML slider. and supports angular values in the range from 0° to 180°, in steps of 5°.

- **Time span search**: Through this option, it is possible to choose a start date and an end date and visualize all data available for that specific time interval. Then, it is sufficient to select the type of data to be searched (temperature, pressure, low definition image etc…). Hence, a search is carried out in the database according to the type of data selected and the time interval chosen by the user. If the query finds one or more appropriate results, the retrieved data are immediately printed in a proper table, together with the respective acquisition date and time. In case the request includes images or videos in a given time frame, the user is also able to download the retrieved data to the local storage.

4.7 Numerical analysis

In this section, the results are illustrated that are obtained by testing different data type transmissions through both the underwater sub-system and the LoRa sub-system. Although executing the tests employing the overall system including the two composing sub-systems, to avoid accounting for the impact of data serialization and format adaptation at the LoRa end node, as required by the use of acoustic devices connected to LoRa shields, we have preferred to split the analysis by investigating separately the two sub-systems. In this way the performance achievable in the two isolated sub-systems can be quantified and analysed to have a detailed evidence about the overall potential system behavior. Indeed the currently developed Proof-of-Concept is intended to represent a preliminary step towards the realization of a real Internet of Underwater Things where more sophisticated boards which solve the above mentioned limitations can be utilized, so reducing the required conversion delay.

4.7.1 Underwater sub-system

In this section, the performance obtained by transmitting different types of data across the underwater sub-system is investigated. Our tests have been carried out in two different scenarios, i.e. an Aquarium setting in the underwater laboratory

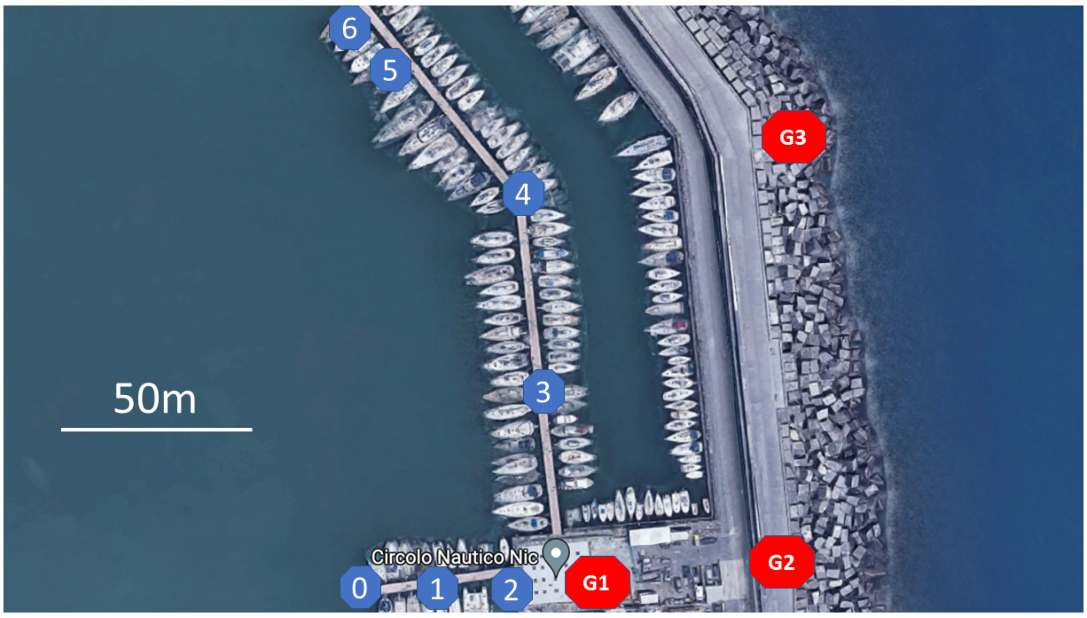


Figure 4.9: Catania Port setting

in the premises of the University of Catania (nodes are located 120 cm apart from each other), and the city port of Catania. Figure 4.9 shows a map of the Catania port setting, where blue dots denote the different placements of the underwater devices during the experiments. In particular, the edge underwater device connected to the LoRa End Node is kept at a fixed position (Point 0), while the other underwater device (or *remote device*) is moved across positions from Point 1 to 6. The red dots denote the different placements of the LoRa Gateway, namely G1, G2 and G3. The LoRa Gateway device has been connected to the Digital Twin and the MQTT broker by means of a mobile 4G connection.

Figure 4.10 depicts the transmission time vs. the image quality indicator in the aquarium setting, and for two different image formats, i.e. WEBP and JPEG. As soon as the image quality increases, the transmission time accordingly increases from 40s to more than 160s in the case of JPEG images, and from 20s to 110s in the case of WEBP images. Note how the supported bit rate in this setting is lower than approximately 10-13 kbps, and strongly depends on both the relative positioning between the underwater modems, and the reflections on the borders of the aquarium. Due to the lower transmission time, the WEBP format is preferable since it can reduce the file size with minimal quality loss due to its aggressive and more optimized compression, as compared to JPEG and PNG.

Figure 4.11 and Figure 4.12 illustrate the same performance metrics in the case of the Port setting, respectively for WEBP images and JPEG images. In

Location	Net bit rate [bps]
Point 1	2481
Point 2	1480
Point 3	1460
Point 4	658
Point 5	1379
Point 6	3097

Table 4.1: Achieved Bit Rate depending on the Location in Catania Port setting

both cases, the experiment has been repeated for several positions of the remote underwater device.

Considering the case of WEBP images Point 1 and Point 6 exhibit a similar behavior in terms of transmission time, in spite of the huge difference in the distancing between the edge underwater modem, and the remote modem (The two devices are 25m apart from each other in the first case, and 175m apart in the second case). Indeed, in both settings, communication can happen in line of sight, with no relevant obstacles in between. In this case, observe how the transmission time is lower, as compared to the aquarium case. This is due to the highly impaired scenario tested in the aquarium, where numerous reflections and serious multi-path is met. On the contrary, Points 2, 3 and 5 exhibit comparable transmission times, and similar to the values experienced in the aquarium setting. Even if the distance from the fixed node is different (50m, 80m and 130m, respectively), the main delay contribution comes from the multi-path reflections and noise. Indeed, Point 2, 3 and 5 are closer to the pier area, where multiple boats are attached and powered, thus causing significant noise and interference. However, the worst performance has been registered in Point 4, located at a distance of 140m from the fixed device, but located in a crowded noisy area. In this case, the delay rises to approximately 250s, depending on the requested image quality. Similar considerations apply to the JPEG case, apart for the delay which, as in the case of the aquarium setting, takes larger values as compared to the case of transmission of a WEBP image. The achieved transmission bit rates for each positioning setting are reported in Table 4.1. An example of a JPEG image transmitted is shown in Figure 4.13.

4.7.2 LoRa Sub-System

In this section, the performance obtained by transmitting different types of data in the LoRa sub-system is investigated.

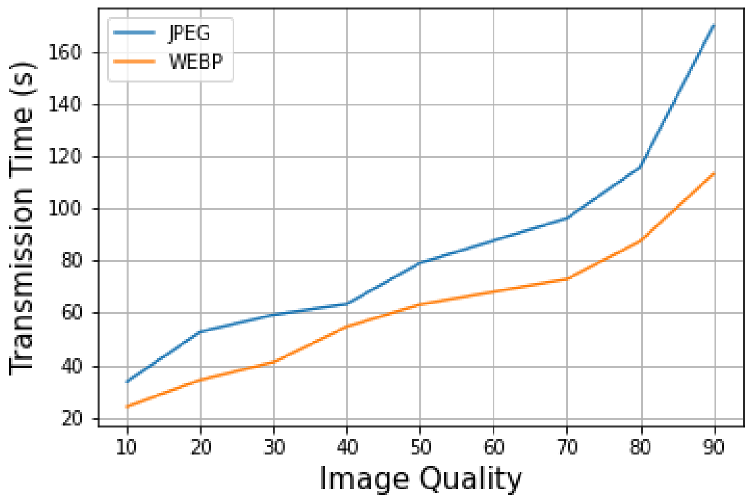


Figure 4.10: Transmission Time (s) Vs. Image Quality-Aquarium setting

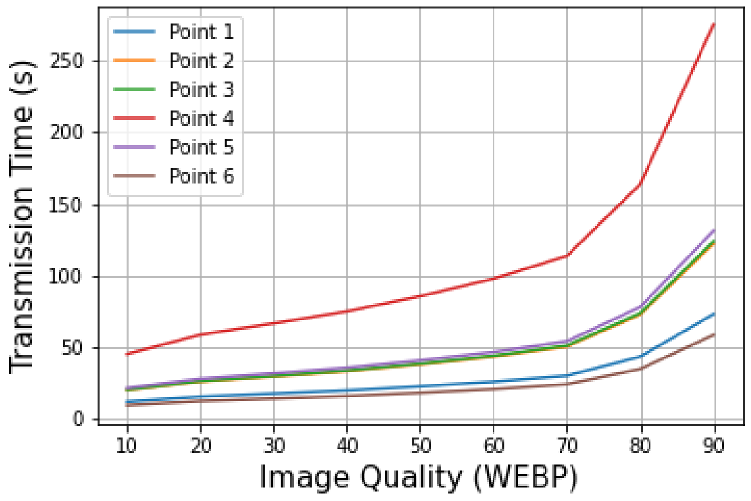


Figure 4.11: Transmission Time (s) Vs. Image Quality (WEBP)- Catania Port setting

Data size (kbits)	Transmission delay (s)
50	8.44
100	15.36
200	29.48

Table 4.2: Transmission Delay using the LoRa/LoRaWAN technology for a SF=7, Bandwidth 250 kHz and a LoRa Gateway located G1.

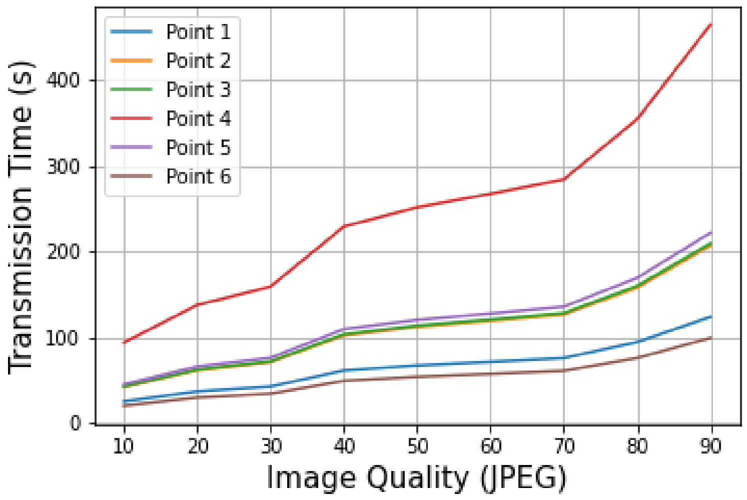


Figure 4.12: Transmission Time (s) Vs. Image Quality (JPEG)- Catania Port setting

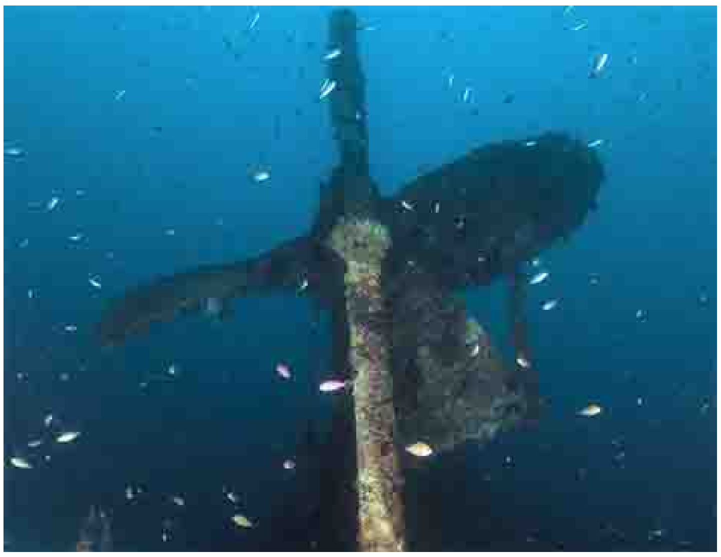


Figure 4.13: Sample image collected during the marine tests.

Gateway Location	PDR (s)
G1	99.71%
G2	99.1%
G3	98.86%

Table 4.3: Packet Delivery Rate as a function of the LoRa Gateway position for SF=11.

Table 4.2 reports the transmission delay as a function of the image size for a fixed spreading factor SF=7, and a bandwidth of 250 kHz. Note that, as expected, an increase in the data size implies a corresponding increase in the transmission time. Also, the measured transmission times are slightly larger than the theoretical expected values. This is due to the delay introduced by a real system implementation.

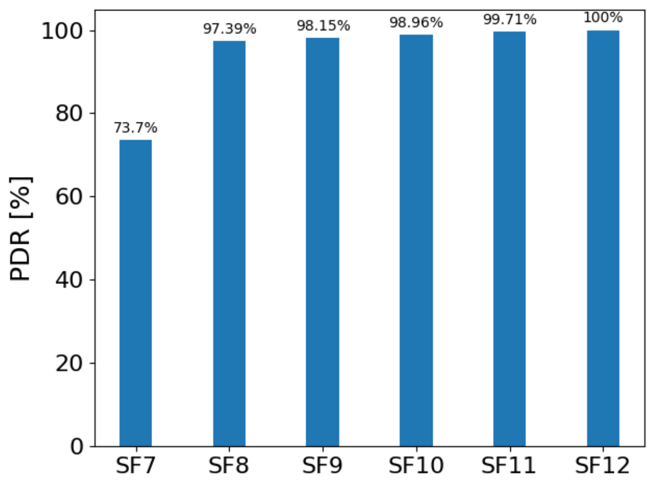


Figure 4.14: Packet Delivery Rate (PDR) as a function of the SF being used.

In Figure 4.14, how different spreading factors influence the Packet Delivery Rate of the LoRa link is analysed when the Gateway node is located in position G1. Note that, upon increasing the SF, a higher PDR is met, at the cost of a lower transmission rate.

To estimate the possible impact of Gateway positioning, we have also measured the Packet Delivery Rate for different positions of the LoRa Gateway, and for SF=11. Table 4.3 illustrates the results of the experiment. Note that, as the Gateway distance from the LoRa End Node increases, the PDR obviously decreases although not exhibiting significant variations, thus proving that the designed and deployed system is robust and reliable.

In this work, an integrated acoustic/LoRa underwater system have been designed and developed that can be used to perform multimedia transmission and actuation over an Internet of Underwater Things. This work represents the first effort in deploying a real system which implements the IoUT vision. Numerous application scenarios for marine monitoring systems have emerged in the last years, including historical sites preservation and wildlife protection. Real field tests assess the reliability and feasibility of the designed system and open up new perspectives in the direction of underwater transmission optimization for large scale monitoring.

4.8 ML in Internet of Underwater Things

In the recent times, more attention has been given to the research on Marine Resources, leading to the development of IoUT. ML algorithms could play a major role in IoUT applications such as earthquake forecasting, underwater navigation and development of underwater monitoring and surveillance systems [131]. Some of the well known underwater monitoring applications are: Coral-reef monitoring, marine Fish monitoring, water quality monitoring and marine plant monitoring [132]. Due to the recent developments in ML and deep neural networks, image recognition systems can achieve significant performances. Especially, Computer vision has been enhanced in the field of image recognition and plays an important role in the above mentioned applications. Computer Vision enables computer systems to automatically derive meaningful information from visual inputs.

As mentioned in the previous sections, many criticalities arise in the case of underwater communication such as high attenuation and high propagation delay. Fading, narrow bandwidth and Doppler effect make communications much more complex in the underwater settings. Therefore transmitting images can be challenging and especially videos can be extremely challenging. Poor channel conditions can result in missing frames in the transmitted video. The application of GANs could address this issue by reconstructing the missing frames.

As discussed in chapter 2, GANs are generative modeling based deep learning algorithms. GANs are exciting and intelligent ML algorithms that have the ability to generate realistic data across various problem domains. In GANs, the generative mode is trained by framing the problem as a supervised learning approach with two sub-models: the *Generator* for generating new data and the *discriminator* for differentiating the real data from the dataset and the fake generated data.

GANs have been used to generate missing frames, especially generating the missing frames in CCTV footage. In [133], a conditional GAN approach is proposed to to learn the spatio-temporal representation of the missing frame in a video. The input to the conditional GAN model will be either the previous frames or the following frames recorded in the same camera or other cameras. However, the representations learned from frames within the camera are given more weightage when compared to the frames from other cameras. This apporach can also be utilized in the transmission of videos in the hybrid underwater/LoRa system for obtaining video data.

Apart from monitoring applications, ML algorithms can also be utilized in developing fault-tolerant IoUT deployment strategies and data fusion strategies. [131].

Chapter 5

Conclusion

The thesis is aimed to analyse the aspects related to the exploitation of ML in infrastructureless networks. In particular, the exploitation of ML was addressed in three areas.

The first one was about the exploitation of ML in the area of WSN. In this context, the application of the well known FL algorithm in WSN and its challenges were discussed. To address those challenges and issues, a novel algorithm called *MGM-4-FL* was introduced by combining FL and Gossiping in WSNs. By jointly considering Gossiping with FL, the energy and communication resources are saved and the funneling effect is eliminated in WSNs. The *MGM-4-FL* was experimented with the auto-encoder ML algorithm and simulation campaigns were conducted. The results obtained indicated that combining FL and Gossiping could be energy efficient and can save the use of communication resources. Another work in the context of WSN was about the use of centrality measures to enhance the collaborative learning of a model by gossiping. A centrality based gossiping protocol *CGL* was introduced and experimented with ML algorithms. Results showed that centrality can influence the gossiping and can help in achieving the convergence at a faster rate, thus being energy efficient. The third work in this direction was about designing and developing a hybrid framework in which nodes have different ML capabilities. A clustering algorithm called *i-WSN* was designed and experimented with different clustering strategies. Simulation campaigns were conducted and experimental results were shown. Finally a SDN approach was proposed in which SDN is applied to WSN with Data mules. The movement of the data mule is forecast by the SDN controller and the forecast positions are considered to generate the flow table entries to be installed in the sensor nodes and schedule their applications. A simple and efficient decision tree algorithm was implemented, which takes the values measured by the sensors as inputs, to forecast the route of the data mule.

The Second area investigated in this thesis was about the exploitation of ML in Vehicular Networks. In this context, two works were carried out. The first one was about monitoring the driving behaviour in real time and warning the driver. A TL approach was introduced to be used with CNN auto-encoders. The TL approach was applied to a specific case study in which various sensor data is collected from different bicyclists riding along the same path. The other work was about achieving layer separation in neural networks. A sequential training algorithm was introduced for partitioning a set of layers for the user and a set of layers for the scenario in a V2I environment. The algorithm was experimented in the same bicycle case study. Results indicated that sequential training is efficient in partitioning the neural network for users and the scenario in a V2I environment.

The third research area dealt with in this thesis was about designing and developing a hybrid underwater/LoRa system that can collect marine data and transmit over real data. Such data can be used through the application of ML algorithms in Internet of Underwater Things applications. The developed system consists of two sub-systems, a pure underwater sub-system, and a terrestrial sub-system. In the underwater sub-system, multiple underwater sensors collect measurements on the underwater environment and transmit this information by means of underwater acoustic modems through an underwater marine link to a receiving edge underwater modem. The terrestrial sub-system consists of the edge underwater modem attached to a buoy where a surface LoRa node is connected by way of a LoRaWAN connection to a terrestrial remote LoRa Gateway, which, in turn, remotely interacts with the cloud by means of an Internet connection. In the cloud, this marine data can be saved and analyzed for further processing and for ML applications.

List of publications

1. J. S. Mertens, G. M. Milotta, P. Nagaradjane and G. Morabito, "SDN-(UAV)ISE: Applying Software Defined Networking to Wireless Sensor Networks with Data Mules," 2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), 2020, pp. 323-328, doi: 10.1109/WoWMoM49955.2020.00061.

2. F. Busacca, L. Galluccio, J. S. Mertens, D. Orto, S. Palazzo, and S. Quattropani. 2020. An experimental testbed of an Internet of Underwater Things. In Proceedings of the 14th International Workshop on Wireless Network Testbeds, Experimental evaluation Characterization (WiNTECH'20). Association for Computing Machinery, New York, NY, USA, 95–102.

3. J. S. Mertens, L. Galluccio and G. Morabito, "Federated learning through model gossiping in wireless sensor networks," 2021 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom), 2021, pp. 1-6, doi: 10.1109/BlackSeaCom52164.2021.9527886.

4. A.A. Brincat, F. Busacca, L. Galluccio, J.S. Mertens, A. Musumeci, S. Palazzo, A. Panebianco, An integrated acoustic/LoRa system for transmission of multimedia sensor data over an Internet of Underwater Things, Computer Communications, Volume 192, 2022, Pages 132-142, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2022.05.032.

5. J.S. Mertens, L. Galluccio, G. Morabito, MGM-4-FL: Combining federated learning and model gossiping in WSNs, Computer Networks, Volume 214, 2022, 109144, ISSN 1389-1286, https://doi.org/10.1016/j.comnet.2022.109144.

6. J. S. Mertens, L. Galluccio and G. Morabito, "Centrality-aware gossiping for distributed learning in wireless sensor networks," 2022 IFIP Networking Conference (IFIP Networking), 2022, pp. 1-6, doi: 10.23919/IFIPNetworking55013.2022.9829795.

7. S. Cafiso, J. S. Mertens, G. Morabito, G. Pappalardo and S. Yaqoob, "Detection of anomalies in cycling behavior with Convolutional Neural Network and Deep Transfer Learning", 34th ICTCT Györ, Hungary 2022.

8. J. S. Mertens, L. Galluccio and G. Morabito, "i-WSN League: Clustered distributed learning in wireless sensor networks". Submitted to IEEE Internet of Things Journal for Publication

9. J. S. Mertens,S. Cafiso, L. Galluccio, G. Morabito,G. Pappalardo, "Cooperative CNN training in wireless networks for applications with user-environment interactions", submitted to IEEE International Conference on Communications, Rome, Italy, 2023.

10. L. Galluccio, J. S. Mertens, and G. Morabito, “Clustered distributed learning exploiting Node centrality and Energy (CINE) in WSNs”, submitted to IEEE International Conference on Communications, Rome, Italy, 2023.

References

[1] Mohammad Abu Alsheikh et al. “Machine Learning in Wireless Sensor Networks: Algorithms, Strategies, and Applications”. In: *IEEE Communications Surveys Tutorials* 16.4 (2014), pp. 1996–2018. DOI: 10.1109/COMST.2014.2320099.

[2] Georgia Koutsandria et al. “Wake-up radio-based data forwarding for green wireless networks”. In: *Computer Communications* 160 (2020), pp. 172–185. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2020.05.046. URL: https://www.sciencedirect.com/science/article/pii/S0140366420300128.

[3] Ilker Demirkol, Cem Ersoy, and Ertan Onur. “Wake-up receivers for wireless sensor networks: benefits and challenges”. In: *IEEE Wireless Communications* 16.4 (2009), pp. 88–96. DOI: 10.1109/MWC.2009.5281260.

[4] V. V. Ghate and V. Vijayakumar. “Machine learning for wireless sensor networks: a survey”. In: *International Journal of Pure and Applied Mathematics* 24 (2018).

[5] *TensorFlow Lite*. https://www.tensorflow.org/lite.

[6] Joost Verbraeken et al. “A Survey on Distributed Machine Learning”. In: *ACM Comput. Surv.* 53.2 (Mar. 2020). ISSN: 0360-0300. DOI: 10.1145/3377454. URL: https://doi.org/10.1145/3377454.

[7] J. M. Kahn, R. H. Katz, and K. S. J. Pister. “Next Century Challenges: Mobile Networking for “Smart Dust””. In: *ACM Mobicom* (1999).

[8] Solmaz Niknam, Harpreet S. Dhillon, and Jeffrey H. Reed. “Federated Learning for Wireless Communications: Motivation, Opportunities, and Challenges”. In: *IEEE Communications Magazine* 58.6 (2020), pp. 46–51. DOI: 10.1109/MCOM.001.1900461.

[9] Tian Li et al. "Federated Learning: Challenges, Methods, and Future Directions". In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60. DOI: `10.1109/MSP.2020.2975749`.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT press, 2016.

[11] L. Bottou. "Large-scale machine learning with stochastic gradient descent." In: *Proc. of COMPSTAT* (2010).

[12] H. B. McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *AISTATS*. 2017.

[13] T. Yang et al. *Applied Federated Learning: Improving Google Keyboard Query Suggestions*. https://arxiv.org/pdf/1812.02903.pdf. 2018.

[14] Dinh Nguyen and et al. "Federated Learning for Smart Healthcare: A Survey". In: *ACM Computing Surveys* 55.3 (2023).

[15] Bruno Camajori-Tedeschini and et al. "Decentralized Federated Learning for Healthcare Networks: A Case Study on Tumor Segmentation". In: *IEEE Access* 10 (2022).

[16] Mohammed Adnan and et al. "Federated learning and differential privacy for medical image analysis". In: *Nature Scientific Reports* 12 (2022).

[17] Keith Bonawitz et al. *Towards Federated Learning at Scale: System Design*. 2019. arXiv: `1902.01046 [cs.LG]`.

[18] Wei Yang Bryan Lim et al. "Federated Learning in Mobile Edge Networks: A Comprehensive Survey". In: *IEEE Communications Surveys Tutorials* 22.3 (2020), pp. 2031–2063. DOI: `10.1109/COMST.2020.2986024`.

[19] Bing Luo et al. "Cost-Effective Federated Learning Design". In: *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications* (2021), pp. 1–10.

[20] *ai.google. Under the hood of the Pixel 2: How AI is supercharging hardware.* https://ai.google/stories/ai-in-hardware/. 2018.

[21] *W.de Brouwer. The federated future is ready for shipping.* https://doc.ai/blog/federated-future-ready-shipping/. March 2019.

[22] David Leroy et al. *Federated Learning for Keyword Spotting*. 2019. arXiv: `1810.05512 [eess.AS]`.

[23] Ji Chu Jiang et al. "Federated Learning in Smart City Sensing: Challenges and Opportunities". In: *Sensors* 20.21 (2020). ISSN: 1424-8220. DOI: 10.3390/s20216230. URL: https://www.mdpi.com/1424-8220/20/21/6230.

[24] Yi Liu et al. "Federated Learning in the Sky: Aerial-Ground Air Quality Sensing Framework With UAV Swarms". In: *IEEE Internet of Things Journal* 8.12 (2021), pp. 9827–9837. DOI: 10.1109/JIOT.2020.3021006.

[25] F. Mao et al. "PPFL: privacy-preserving Federated Learning with Trusted Execution Environments". In: *ACM MobySys 2021*. 2021.

[26] N. Kourtellis, K. Katevas, and D. Perino. "FLaaS: Federated Learning as a Service". In: *1st Workshop on Distributed Machine Learning (DistributedML'20)*. 2020.

[27] Jorge Mena, Mario Gerla, and Vana Kalogeraki. "Mitigate Funnel Effect in Sensor Networks with Multi-interface Relay Nodes". In: *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems* (2012), pp. 216–223.

[28] Gahng-Seop Ahn et al. "Funneling-MAC: A Localized, Sink-Oriented MAC for Boosting Fidelity in Sensor Networks". In: *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*. SenSys '06. Boulder, Colorado, USA: Association for Computing Machinery, 2006, pp. 293–306. ISBN: 1595933433. DOI: 10.1145/1182807.1182837. URL: https://doi.org/10.1145/1182807.1182837.

[29] Alexandros G. Dimakis et al. "Gossip Algorithms for Distributed Signal Processing". In: *Proceedings of the IEEE* 98.11 (2010), pp. 1847–1864. DOI: 10.1109/JPROC.2010.2052531.

[30] D. Shah. "Gossip Algorithms". In: *Foundations and Trends in Networking* 3.1 (2009).

[31] S. Boyd et al. "Randomized gossip algorithms". In: *IEEE Transactions on Information Theory* 52.6 (2006), pp. 2508–2530. DOI: 10.1109/TIT.2006.874516.

[32] S. Kar and J. Moura. "Consensus-based detection in sensor networks: Topology optimization under practical constraints". In: *Int. Workshop Inf. Theory Sensor Netw.* (2007).

[33] V. Saligrama, M. Alanyali, and O. Savas. "Distributed Detection in Sensor Networks With Packet Losses and Finite Capacity Links". In: *IEEE Transactions on Signal Processing* 54.11 (2006), pp. 4118–4132. DOI: 10.1109/TSP.2006.880227.

[34] Michael Blot et al. *Gossip training for deep learning*. 2016. arXiv: 1611.09726 [cs.CV].

[35] Jeff Daily et al. *GossipGraD: Scalable Deep Learning using Gossip Communication based Asynchronous Gradient Descent*. 2018. arXiv: 1803.05880 [cs.DC].

[36] J. S. Mertens, L. Galluccio, and G. Morabito. "Federated learning through model gossiping in wireless sensor networks". In: *2021 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. 2021.

[37] Anusha Lalitha et al. *Peer-to-peer Federated Learning on Graphs*. 2019. arXiv: 1901.11173 [cs.LG].

[38] Koya Sato, Yasuyuki Satoh, and Daisuke Sugimura. "Network-Density-Controlled Decentralized Parallel Stochastic Gradient Descent in Wireless Systems". In: *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)* (2020), pp. 1–6.

[39] Stefano Savazzi, Monica Nicoli, and Vittorio Rampa. "Federated Learning With Cooperating Devices: A Consensus Approach for Massive IoT Networks". In: *IEEE Internet of Things Journal* 7.5 (2020), pp. 4641–4654. DOI: 10.1109/JIOT.2020.2964162.

[40] István Hegedűs, Gábor Danner, and Márk Jelasity. "Gossip Learning as a Decentralized Alternative to Federated Learning". In: *Distributed Applications and Interoperable Systems*. Ed. by José Pereira and Laura Ricci. Cham: Springer International Publishing, 2019, pp. 74–90.

[41] Chenghao Hu, Jingyan Jiang, and Zhi Wang. "Decentralized Federated Learning: A Segmented Gossip Approach". In: *ArXiv* (2019).

[42] Weibo Liu et al. "A survey of deep neural network architectures and their applications". In: *Neurocomputing* 234 (2017), pp. 11–26. ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2016.12.038.

[43] *et al.* J. K. Chow. "Anomaly detection of defects on concrete structures with the convolutional autoencoder". In: *Elsevier Advanced Engineering informatics* 45 (2020).

[44] B. Vandalore et al. “General Weighted Fairness and its Support in Explicit Rate Switch Algorithms”. In: *Computer Communications* 23 (2000).

[45] Antonia Creswell et al. “Generative Adversarial Networks: An Overview”. In: *IEEE Signal Processing Magazine* 35.1 (2018), pp. 53–65. DOI: 10.1109/MSP.2017.2765202.

[46] Jie Gui et al. “A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pp. 1–1. DOI: 10.1109/TKDE.2021.3130191.

[47] Linton C. Freeman. “Centrality in social networks conceptual clarification”. In: *Social Networks* 1.3 (1978), pp. 215–239. ISSN: 0378-8733. DOI: https://doi.org/10.1016/0378-8733(78)90021-7. URL: https://www.sciencedirect.com/science/article/pii/0378873378900217.

[48] Aarti Jain and B.V.R. Reddy. “Node centrality in wireless sensor networks: Importance, applications and advances”. In: *2013 3rd IEEE International Advance Computing Conference (IACC)*. 2013, pp. 127–131. DOI: 10.1109/IAdCC.2013.6514207.

[49] L. Katz. “A New Status Index Derived from Sociometric Analysis”. In: *Psychometrika* (1953), pp. 39–43.

[50] Panpan Zhang, Tiandong Wang, and Jun Yan. *PageRank centrality and algorithms for weighted, directed networks with applications to World Input-Output Tables*. 2021. arXiv: 2104.02764 [physics.soc-ph].

[51] Indranil Gupta, D. Riordan, and Srinivas Sampalli. “Cluster-head election using fuzzy logic for wireless sensor networks”. In: *3rd Annual Communication Networks and Services Research Conference (CNSR’05)*. 2005, pp. 255–260. DOI: 10.1109/CNSR.2005.27.

[52] VMVS Aditya et al. “Closeness Centrality Based Cluster Head Selection Algorithm for Large Scale WSNs”. In: *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*. 2020, pp. 107–111. DOI: 10.1109/CICN49253.2020.9242639.

[53] Eduardo M. R. Oliveira, Heitor S. Ramos, and Antonio A. F. Loureiro. “Centrality-based routing for Wireless Sensor Networks”. In: *2010 IFIP Wireless Days*. 2010, pp. 1–5. DOI: 10.1109/WD.2010.5657731.

[54] Aarti Jain. “Betweenness Centrality Based Connectivity Aware Routing Algorithm for Prolonging Network Lifetime in Wireless Sensor Networks”. In: *Wirel. Netw.* 22.5 (July 2016), pp. 1605–1624. ISSN: 1022-0038. DOI: 10.1007/s11276-015-1054-5. URL: https://doi.org/10.1007/s11276-015-1054-5.

[55] Anne-Marie Kermarrec et al. “Second order centrality: Distributed assessment of nodes criticity in complex networks”. In: *Computer Communications* 34.5 (2011), pp. 619–628.

[56] Klaus Wehmuth and Artur Ziviani. “Daccer: Distributed assessment of the closeness centrality ranking in complex networks”. In: *Computer Networks* 57.13 (2013), pp. 2536–2548.

[57] Keyou You, Roberto Tempo, and Li Qiu. “Distributed algorithms for computation of centrality measures in complex networks”. In: *IEEE Transactions on Automatic Control* 62.5 (2016), pp. 2080–2094.

[58] Ranjan Kumar Behera et al. “Distributed centrality analysis of social network data using MapReduce”. In: *Algorithms* 12.8 (2019), p. 161.

[59] Yanjie Duan, Yisheng L.V., and Fei-Yue Wang. “Travel time prediction with LSTM neural network”. In: *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*. 2016, pp. 1053–1058. DOI: 10.1109/ITSC.2016.7795686.

[60] YiFei Li and Han Cao. “Prediction for Tourism Flow based on LSTM Neural Network”. In: *Procedia Computer Science* 129 (2018). 2017 INTERNATIONAL CONFERENCE ON IDENTIFICATION,INFORMATION AND KNOWLEDGEIN THE INTERNET OF THINGS, pp. 277–283. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.03.076. URL: https://www.sciencedirect.com/science/article/pii/S1877050918303016.

[61] Weicong Kong et al. “Short-Term Residential Load Forecasting Based on LSTM Recurrent Neural Network”. In: *IEEE Transactions on Smart Grid* 10.1 (2019), pp. 841–851. DOI: 10.1109/TSG.2017.2753802.

[62] Martin Rapp, Ramin Khalili, and Jörg Henkel. “Distributed learning on heterogeneous resource-constrained devices”. In: *arXiv:2006.05403* (2020).

[63] Heon Huh and Jeong Yeol Kim. “LoRa-based mesh network for IoT applications”. In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE. 2019, pp. 524–527.

[64] Qi Wang et al. "A comprehensive survey of loss functions in machine learning". In: *Annals of Data Science* (2020), pp. 1–26.

[65] I.F. Akyildiz et al. "Wireless sensor networks: a survey". In: *Computer Networks* 38.4 (2002), pp. 393–422. ISSN: 1389-1286. DOI: `https://doi.org/10.1016/S1389-1286(01)00302-4`. URL: `https://www.sciencedirect.com/science/article/pii/S1389128601003024`.

[66] Jose Polo et al. "Design of a low-cost Wireless Sensor Network with UAV mobile node for agricultural applications". In: *Computers and Electronics in Agriculture* 119 (2015), pp. 19–32. ISSN: 0168-1699. DOI: `https://doi.org/10.1016/j.compag.2015.09.024`. URL: `https://www.sciencedirect.com/science/article/pii/S0168169915002999`.

[67] Milan Erdelj, Michał Król, and Enrico Natalizio. "Wireless Sensor Networks and Multi-UAV systems for natural disaster management". In: *Computer Networks* 124 (2017), pp. 72–86. ISSN: 1389-1286. DOI: `https://doi.org/10.1016/j.comnet.2017.05.021`. URL: `https://www.sciencedirect.com/science/article/pii/S1389128617302220`.

[68] Tales Heimfarth, João Paulo De Araujo, and João Carlos Giacomin. "Unmanned Aerial Vehicle as Data Mule for Connecting Disjoint Segments of Wireless Sensor Network with Unbalanced Traffic". In: *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. 2014, pp. 246–252. DOI: `10.1109/ISORC.2014.51`.

[69] David Palma et al. "Unmanned Aerial Vehicles as Data Mules: An Experimental Assessment". In: *IEEE Access* 5 (2017), pp. 24716–24726. DOI: `10.1109/ACCESS.2017.2769658`.

[70] Emmanuel Tuyishimire et al. "Cooperative data muling from ground sensors to base stations using UAVs". In: *2017 IEEE Symposium on Computers and Communications (ISCC)*. 2017, pp. 35–41. DOI: `10.1109/ISCC.2017.8024501`.

[71] Laura Galluccio et al. "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks". In: *2015 IEEE Conference on Computer Communications (INFOCOM)*. 2015, pp. 513–521. DOI: `10.1109/INFOCOM.2015.7218418`.

[72] Angelos-Christos G. Anadiotis, Giacomo Morabito, and Sergio Palazzo. "An SDN-Assisted Framework for Optimal Deployment of MapReduce Functions in WSNs". In: *IEEE Transactions on Mobile Computing* 15.9 (2016), pp. 2165–2178. DOI: `10.1109/TMC.2015.2496582`.

[73] Muhammad Zohaib Anwar, Zeeshan Kaleem, and Abbas Jamalipour. "Machine Learning Inspired Sound-Based Amateur Drone Detection for Public Safety Applications". In: *IEEE Transactions on Vehicular Technology* 68.3 (2019), pp. 2526–2534. DOI: 10.1109/TVT.2019.2893615.

[74] Lin Shan et al. "Machine Learning-Based Field Data Analysis and Modeling for Drone Communications". In: *IEEE Access* 7 (2019), pp. 79127–79135. DOI: 10.1109/ACCESS.2019.2922544.

[75] Ursula Challita et al. "Machine Learning for Wireless Connectivity and Security of Cellular-Connected UAVs". In: *IEEE Wireless Communications* 26.1 (2019), pp. 28–35. DOI: 10.1109/MWC.2018.1800155.

[76] Olivier Goldschmidt, Dorit S. Hochbaum, and Gang Yu. "A Modified Greedy Heuristic for the Set Covering Problem with Improved Worst Case Bound". In: *Inf. Process. Lett.* 48.6 (Dec. 1993), pp. 305–310. ISSN: 0020-0190. DOI: 10.1016/0020-0190(93)90173-7. URL: https://doi.org/10.1016/0020-0190(93)90173-7.

[77] Fernando C. Gomes et al. "Experimental Analysis of Approximation Algorithms for the Vertex Cover and Set Covering Problems". In: *Computers Operations Research* 33.12 (2006). Part Special Issue: Recent Algorithmic Advances for Arc Routing Problems, pp. 3520–3534. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2005.03.030. URL: https://www.sciencedirect.com/science/article/pii/S0305054805001255.

[78] A. Mei and J. Stefa. "SWIM: A Simple Model to Generate Small Mobile Worlds". In: *IEEE INFOCOM 2009*. 2009, pp. 2106–2113. DOI: 10.1109/INFCOM.2009.5062134.

[79] *WHO Road traffic injuries Key facts*. https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries. Accessed: 17-05-2022.

[80] Reza Malekian et al. "Design and Implementation of a Wireless OBD II Fleet Management System". In: *IEEE Sensors Journal* 17.4 (2017), pp. 1154–1164. DOI: 10.1109/JSEN.2016.2631542.

[81] Emanuele Lattanzi and Valerio Freschi. "Machine Learning Techniques to Identify Unsafe Driving Behavior by Means of In-Vehicle Sensor Data". In: *Expert Systems with Applications* 176 (2021), p. 114818. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2021.114818. URL: https://www.sciencedirect.com/science/article/pii/S0957417421002591.

[82] Yi Niu, Zhenming Li, and Yunxiao Fan. “Analysis of truck drivers’ unsafe driving behaviors using four machine learning methods”. In: *International Journal of Industrial Ergonomics* 86 (2021), p. 103192. ISSN: 0169-8141. DOI: https://doi.org/10.1016/j.ergon.2021.103192. URL: https://www.sciencedirect.com/science/article/pii/S0169814121001104.

[83] Sandeep Nair Narayanan, Sudip Mittal, and Anupam Joshi. “OBD SecureAlert: An Anomaly Detection System for Vehicles”. In: *2016 IEEE International Conference on Smart Computing (SMARTCOMP)*. 2016, pp. 1–6. DOI: 10.1109/SMARTCOMP.2016.7501710.

[84] Selim S. Sarikan and A. Murat Ozbayoglu. “Anomaly Detection in Vehicle Traffic with Image Processing and Machine Learning”. In: *Procedia Computer Science* 140 (2018). Cyber Physical Systems and Deep Learning Chicago, Illinois November 5-7, 2018, pp. 64–69. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.10.293. URL: https://www.sciencedirect.com/science/article/pii/S1877050918319665.

[85] Yuhao Wang and Ivan Wang-Hei Ho. “Joint Deep Neural Network Modelling and Statistical Analysis on Characterizing Driving Behaviors”. In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018, pp. 1–6. DOI: 10.1109/IVS.2018.8500376.

[86] H.E.C. van der Wall et al. “The use of machine learning improves the assessment of drug-induced driving behaviour”. In: *Accident Analysis Prevention* 148 (2020), p. 105822. ISSN: 0001-4575. DOI: https://doi.org/10.1016/j.aap.2020.105822. URL: https://www.sciencedirect.com/science/article/pii/S0001457520316420.

[87] Fatjon Seraj et al. “A Smartphone Based Method to Enhance Road Pavement Anomaly Detection by Analyzing the Driver Behavior”. In: *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*. UbiComp/ISWC’15 Adjunct. Osaka, Japan: Association for Computing Machinery, 2015, pp. 1169–1177. ISBN: 9781450335751. DOI: 10.1145/2800835.2800981. URL: https://doi.org/10.1145/2800835.2800981.

[88] Hiroki Oikawa et al. “Fast Semi-Supervised Anomaly Detection of Drivers’ Behavior using Online Sequential Extreme Learning Machine”. In: *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 2020, pp. 1–8. DOI: 10.1109/ITSC45102.2020.9294659.

[89] Tolga Ergen and Suleyman Serdar Kozat. “Unsupervised Anomaly Detection With LSTM Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 31.8 (2020), pp. 3127–3141. DOI: 10.1109/TNNLS.2019.2935975.

[90] Chong Zhou and Randy C. Paffenroth. “Anomaly Detection with Robust Deep Autoencoders”. In: KDD ’17. Halifax, NS, Canada: Association for Computing Machinery, 2017, pp. 665–674. ISBN: 9781450348874. DOI: 10.1145/3097983.3098052. URL: https://doi.org/10.1145/3097983.3098052.

[91] Shaohan Huang et al. “HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Log”. In: *IEEE Transactions on Network and Service Management* 17.4 (2020), pp. 2064–2076. DOI: 10.1109/TNSM.2020.3034647.

[92] Yongbeom Lee and Seongkeun Park. “A Deep Learning-Based Perception Algorithm Using 3D LiDAR for Autonomous Driving: Simultaneous Segmentation and Detection Network (SSADNet)”. In: *Applied Sciences* 10.13 (2020). ISSN: 2076-3417. DOI: 10.3390/app10134486. URL: https://www.mdpi.com/2076-3417/10/13/4486.

[93] Chanyoung Jung et al. “V2X-Communication-Aided Autonomous Driving: System Design and Experimental Validation”. In: *Sensors* 20.10 (2020). ISSN: 1424-8220. DOI: 10.3390/s20102903. URL: https://www.mdpi.com/1424-8220/20/10/2903.

[94] Yaran Chen et al. “Multi-task learning for dangerous object detection in autonomous driving”. In: *Information Sciences* 432 (2018), pp. 559–571. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2017.08.035. URL: https://www.sciencedirect.com/science/article/pii/S0020025517308848.

[95] Hao Du et al. “A New Vehicular Fog Computing Architecture for Cooperative Sensing of Autonomous Driving”. In: *IEEE Access* 8 (2020), pp. 10997–11006. DOI: 10.1109/ACCESS.2020.2964029.

[96] Changxi You et al. “Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning”. In: *Robotics and Autonomous Systems* 114 (2019), pp. 1–18. ISSN: 0921-8890. DOI: https://doi.org/10.1016/j.robot.2019.01.003. URL: https://www.sciencedirect.com/science/article/pii/S0921889018302021.

[97] Raluca Brehar, Cristian Vancea, and Sergiu Nedevschi. “Pedestrian detection in infrared images using Aggregated Channel Features”. In: *2014 IEEE 10th International Conference on Intelligent Computer Communication and Processing (ICCP)*. 2014, pp. 127–132. DOI: `10.1109/ICCP.2014.6936964`.

[98] Guiyuan Li et al. “Application of Convolutional Neural Network (CNN) AdaBoost Algorithm in Pedestrian Detection”. In: *Sensors and Materials* (2020).

[99] Iftikhar Rasheed, Fei Hu, and Lin Zhang. “Deep reinforcement learning approach for autonomous vehicle systems for maintaining security and safety using LSTM-GAN”. In: *Vehicular Communications* 26 (2020), p. 100266. ISSN: 2214-2096. DOI: `https://doi.org/10.1016/j.vehcom.2020.100266`. URL: `https://www.sciencedirect.com/science/article/pii/S2214209620300371`.

[100] J.S. Mertens, L. Galluccio, and G. Morabito. “Federated learning through model gossiping in wireless sensor networks”. In: *2021 IEEE International Black Sea Conference on Communications and Networking (BlackSeaCom)*. 2021, pp. 1–6. DOI: `10.1109/BlackSeaCom52164.2021.9527886`.

[101] Qinxue Meng et al. “Relational autoencoder for feature extraction”. In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 364–371. DOI: `10.1109/IJCNN.2017.7965877`.

[102] Zhaomin Chen et al. “Autoencoder-based network anomaly detection”. In: *2018 Wireless Telecommunications Symposium (WTS)*. 2018, pp. 1–5. DOI: `10.1109/WTS.2018.8363930`.

[103] Mehmet Saygın Seyfioğlu, Ahmet Murat Özbayoğlu, and Sevgi Zubeyde Gürbüz. “Deep convolutional autoencoder for radar-based classification of similar aided and unaided human activities”. In: *IEEE Transactions on Aerospace and Electronic Systems* 54.4 (2018), pp. 1709–1723. DOI: `10.1109/TAES.2018.2799758`.

[104] Zhengxue Cheng et al. “Deep Convolutional AutoEncoder-based Lossy Image Compression”. In: *2018 Picture Coding Symposium (PCS)*. 2018, pp. 253–257. DOI: `10.1109/PCS.2018.8456308`.

[105] Fuzhen Zhuang et al. “A Comprehensive Survey on Transfer Learning”. In: *Proceedings of the IEEE* 109.1 (2021), pp. 43–76. DOI: `10.1109/JPROC.2020.3004555`.

[106] Fuzhen Zhuang et al. "A Comprehensive Survey on Transfer Learning". In: *Proceedings of the IEEE* 109.1 (2021), pp. 43–76. DOI: 10.1109/JPROC.2020.3004555.

[107] Hoo-Chang Shin et al. "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning". In: *IEEE Transactions on Medical Imaging* 35.5 (2016), pp. 1285–1298. DOI: 10.1109/TMI.2016.2528162.

[108] S. Deepak and P.M. Ameer. "Brain tumor classification using deep CNN features via transfer learning". In: *Computers in Biology and Medicine* 111 (2019), p. 103345. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2019.103345. URL: https://www.sciencedirect.com/science/article/pii/S0010482519302148.

[109] Dongmei Han, Qigang Liu, and Weiguo Fan. "A new image classification method using CNN transfer learning and web data augmentation". In: *Expert Systems with Applications* 95 (2018), pp. 43–56. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2017.11.028. URL: https://www.sciencedirect.com/science/article/pii/S0957417417307844.

[110] Manali Shaha and Meenakshi Pawar. "Transfer Learning for Image Classification". In: *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. 2018, pp. 656–660. DOI: 10.1109/ICECA.2018.8474802.

[111] Okan et al. "Driver Anomaly Detection: A Dataset and Contrastive Learning Approach". In: *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2021, pp. 91–100. DOI: 10.1109/WACV48630.2021.00014.

[112] Yazan Otoum, Yue Wan, and Amiya Nayak. "Transfer Learning-Driven Intrusion Detection for Internet of Vehicles (IoV)". In: *2022 International Wireless Communications and Mobile Computing (IWCMC)*. 2022, pp. 342–347. DOI: 10.1109/IWCMC55113.2022.9825115.

[113] Daniel Jiang and Luca Delgrossi. "IEEE 802.11p: Towards an International Standard for Wireless Access in Vehicular Environments". In: *VTC Spring 2008 - IEEE Vehicular Technology Conference*. 2008, pp. 2036–2040. DOI: 10.1109/VETECS.2008.458.

[114] Stephan Eichler. "Performance Evaluation of the IEEE 802.11p WAVE Communication Standard". In: *2007 IEEE 66th Vehicular Technology Conference*. 2007, pp. 2199–2203. DOI: 10.1109/VETECF.2007.461.

[115] J. Gozalvez, M. Sepulcre, and R. Bauza. “IEEE 802.11p vehicle to infrastructure communications in urban environments”. In: *IEEE Communications Magazine* 50.5 (2012), pp. 176–183. DOI: 10.1109/MCOM.2012.6194400.

[116] H. Hartenstein and L.P. Laberteaux. “A tutorial survey on vehicular ad hoc networks”. In: *IEEE Communications Magazine* 46.6 (2008), pp. 164–171. DOI: 10.1109/MCOM.2008.4539481.

[117] Ribal Atallah, Maurice Khabbaz, and Chadi Assi. “Multihop V2I Communications: A Feasibility Study, Modeling, and Performance Analysis”. In: *IEEE Transactions on Vehicular Technology* 66.3 (2017), pp. 2801–2810. DOI: 10.1109/TVT.2016.2586758.

[118] Kevin Yardy, Abdulaziz Almehmadi, and Khalil El-Khatib. “Detecting Malicious Driving with Machine Learning”. In: *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. 2019, pp. 1–6. DOI: 10.1109/WCNC.2019.8886013.

[119] Abdalla Ebrahim Abdelrahman, Hossam S. Hassanein, and Najah Abu-Ali. “Robust Data-Driven Framework for Driver Behavior Profiling Using Supervised Machine Learning”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.4 (2022), pp. 3336–3350. DOI: 10.1109/TITS.2020.3035700.

[120] Yunpeng Wang et al. “Throughput and Delay Limits of 802.11p and its Influence on Highway Capacity”. In: *Procedia - Social and Behavioral Sciences* 96 (2013). Intelligent and Integrated Sustainable Multimodal Transportation Systems Proceedings from the 13th COTA International Conference of Transportation Professionals (CICTP2013), pp. 2096–2104. ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2013.08.236. URL: https://www.sciencedirect.com/science/article/pii/S1877042813023628.

[121] Gunasekaran Raja et al. “SDN-enabled Traffic Alert System for IoV in Smart Cities”. In: *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2020, pp. 1093–1098. DOI: 10.1109/INFOCOMWKSHPS50562.2020.9162888.

[122] M. Jahanbakht et al. “Internet of Underwater Things and Big Marine Data Analytics – A Comprehensive Survey”. In: *IEEE Communications Surveys and Tutorials* (2021).

[123] M. C. Domingo. “An overview of the internet of underwater things”. In: *Journal of Network and Computer Applications* 35.6 (2012).

[124] I. F. Akyildiz, D. Pompili, and T. Melodia. “Underwater acoustic sensor networks: research challenges”. In: *Ad Hoc Networks* 3.3 (May 2005).

[125] S. Falleni et al. “Design, Development, and Testing of a Smart Buoy for Underwater Testbeds in Shallow Waters”. In: *IEEE Global OCEANS 2020* (2020).

[126] E. Demirors et al. “The SEANet project: Toward a programmable internet of underwater things”. In: *IEEE UComms* (2018).

[127] Fondriest. URL: `https://www.fondriest.com/products/wireless-data/data-buoys.html`.

[128] Northeastern University. *NU MONET Project*. Feb. 2022. URL: `https://www.northeastern.edu/numonet/`.

[129] J. Haxhibeqiri et al. “A Survey of LoRaWAN for IoT: From Technology to Application”. In: *MDPI Sensors* 19.7 (Apr. 2019).

[130] J. Prasanna et al. “A Survey on LoRa Networking: Research Problems,Current Solutions and Open Issues”. In: *IEEE Communications Surveys & Tutorials* 22.1 (First Quarter 2020).

[131] Xiangwang Hou et al. “Machine-Learning-Aided Mission-Critical Internet of Underwater Things”. In: *IEEE Network* 35.4 (2021), pp. 160–166. DOI: `10.1109/MNET.011.2000684`.

[132] Marouane Salhaoui et al. “Autonomous Underwater Monitoring System for Detecting Life on the Seabed by Means of Computer Vision Cloud Services”. In: *Remote Sensing* 12.12 (2020). ISSN: 2072-4292. DOI: `10.3390/rs12121981`. URL: `https://www.mdpi.com/2072-4292/12/12/1981`.

[133] Tahmida Mahmud, Mohammad Billah, and Amit K. Roy-Chowdhury. “MULTI-VIEW FRAME RECONSTRUCTION WITH CONDITIONAL GAN”. In: *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. 2018, pp. 1164–1168. DOI: `10.1109/GlobalSIP.2018.8646380`.