Politecnico di Torino

Politecnico di Torino

Communications and Computer Networks Engineering
A.a. 2022/2023
Sessione di Laurea Ottobre 2023

Decentralized Federated Learning for Object Detection Tasks

A Routing Approach to the Reduction of the Communication Overhead

Relatori:
Prof. Carla Fabiana Chiasserini
Prof. Dr.-Ing. habil. Falko Dressler

Candidati:
Tiago Pina Contini

Abstract

Decentralized Federated Learning (DFL) is a new paradigm that arose to address the problems of having a central server in Federated Learning (FL). FL allows to train Machine Learning (ML) models in a privacy preserving way by sending model updates instead of the raw data to a central server for aggregation, but it presents issues with single point of failure and communication bottleneck at the server. DFL instead eliminates the need for a central server by exchanging the model updates directly between the participating clients. In order to reduce resource consumption, a small number of connections among the clients is preferable. However, the topology of the system plays an important role in the performance of the training task, especially under heterogeneous data distribution, and the elimination of connections can deteriorate it severely. This work presents an analysis of the impact of the topology of a DFL system on the training task, and proposes a routing algorithm for the selection of the minimum amount of connections necessary to reach all nodes while minimizing the impact on performance, considering extremely unbalanced non-Independently and Identically Distributed (i.i.d.) data among the clients. The analysis is based on the distribution of data generated by the partitioning of an Object Detection (OD) dataset, the MS COCO, and through simulations the proposed algorithm was shown to be effective in reducing the number of connections, up to eight times, while keeping a similar performance. A simple validation of the robustness of the system under the specific data distribution used is also provided, by simulating node failures at different points of the training task.

Contents

Chapter 1

Introduction

Society is getting increasingly connected. Digital technologies permeate our lives, and they generate and exchange large amounts of data in order to provide useful services, usually with the aid of Machine Learning (ML) techniques. However, this trend generates societal and technical concerns: First, the amount of data exchanged challenges the capacity of our networks; Second, privacy has become an important issue for governments and citizens.

One of the technologies developed in the last few years to address these challenges is Federated Learning (FL), expected to be a game-changer. It refers to a new approach to ML based on in-device learning, which allows to keep the raw data local, and transmit only the model updates to a central server for aggregation and global model construction. This has several advantages such as increased privacy, less communication overhead, and the possibility of harnessing the available, but not utilized, computing power of many edge devices.

This approach, however, presents some important shortcomings. First of all, it is based on a central server for aggregation, which represents a single point of failure. This may create problems, especially for applications whose topology is highly unstable, such as in swarms of drones. Furthermore, for scenarios with a massive number of devices, it may be difficult for all of them to exchange models with a single server, which becomes a bottleneck for the system.

For these reasons, Decentralized Federated Learning (DFL) architectures have been recently proposed by researchers. It consists in performing the aggregation step in the device itself, after exchanging the model parameters between neighboring devices, eliminating the need for a central server. The main advantage of such an approach is the increased robustness against node failures, as well as the elimination of the need of every node to directly exchange information with a single entity in the system. The connections between the participating clients of such a system form a topology, which has a great impact on the performance of the training task. This

is even more evident for heterogeneous distribution of data among the clients, in which different nodes may have varying levels of contribution to the training of the system.

For both FL and DFL, the reduction of the communication overhead represents an important research direction, as it allows to save resources at devices and networks. This is an important development for future applications, as the trend is for an increasing number of participating devices with limited resources, such as in Internet of Things (IoT) applications, and more complex models to provide more intricate services. Current approaches are based on the reduction of the communication frequency and message size between the participants. In the optic of DFL, an alternative approach is possible to save resources by reducing the number of connections between the participating clients. However, the reduction of the number of connections changes the topology of the system, which can severely deteriorate the performance of the training task. The goal of this thesis is to propose an approach to the reduction of the number of connections while minimizing the impact on performance.

The impact of the topology is highly dependent on the data distribution, which depends on the task and dataset at hand. In this work, the focus is on Object Detection (OD) tasks, as it is an important building block for many applications, and yet with little exploration in the DFL literature. The chosen dataset is the popular MS COCO, partitioned in such a way that generates extremely unbalanced non-Independently and Identically Distributed (i.i.d.) data among the clients, with one client concentrating most of the data. Such a distribution might be found in many situations in which a single entity collects most of the information, for example, with surveillance cameras, when one camera is pointed to an area with high circulation and the others to more isolated areas.

The thesis identifies the main factors of the topology that affect the performance of training an OD model, the YOLOv5, with the specified data distribution by simulating the system with different topologies, and lists guidelines for the improvement of the learning speed and accuracy. Based on these guidelines, the Minimum Dilution Routing Algorithm is proposed as a solution to select the minimum amount of connections necessary to connect all nodes, thus reducing the resource usage, while minimizing the deterioration of the performance. The algorithm is validated in two different scenarios, and it is shown that it can reduce up to eight times the number of connections, while keeping the learning speed of all nodes similar and even improving the final accuracy. An extension to the algorithm is also proposed to address problematic cases and to generalize it to more complex scenarios. A simple validation of the robustness of the DFL system with this data distribution is also provided by simulating node failures at two different points of the training task.

Chapter 2

Fundamentals

In this chapter the main basic concepts needed to understand the work will be presented. They are divided in the following subsections: Machine Learning, Federated Learning, Decentralized Federated Learning, Object Detection, MS COCO, YOLO Model. The algorithms and methods used in this work will also be presented and discussed.

2.1 Machine Learning

ML is a broad term that encompasses many techniques targeted at making a machine perform a given task increasingly better based on experience, or data [1]. There are two types: Supervised Learning, which is based on labelled data to train the algorithm, and Unsupervised learning, which generally searches for relations between unlabelled groups of data. The most common type is Supervised Learning, and its training procedure consists in the computation of an Objective Function that measures the difference with respect to the desired output. In order to reduce this error, the adjustable parameters, named weights, are changed according to the criteria of the optimization algorithm. The optimization algorithms are usually based on gradient computation, which indicates how weights should change in order to reduce the objective function.

2.1.1 Concepts

The main concepts needed in order to understand the ML task in this work are described below:

Dataset: It is the (labeled) data available to perform training and testing operations. The quantity and quality of the data is a critical point to have good performing models. It is usually divided in three parts: Training dataset, used to perform the

changes in the weights in order to decrease the objective function; Validation dataset, used in order to access the progress of performance during the training, and should consist of unseen data samples; Test dataset, for a final evaluaton of the model, consisting also of previously unseen samples.

Epochs: The number of epochs is the number of times that the algorithm will go through the entire training dataset for training.

Batch size: It is the number of samples to be processed before updating the model with the optimization algorithm. When all samples have been processed, taken in groups as big as the batch size, an epoch is completed.

Learning rate: It is a hyperparameter that controls the amount that weights get changed with respect to the loss gradient. Large values tend to lead to a faster convergence, but it also risks overshooting the minimum of the function.

2.1.2 Optimization algorithms

The optimization algorithms used in this work are:

Stochastic Gradient Descent (SGD): It consists in performing the update of the weights using the average gradient over a few data samples, repeated many times. The term "stochastic" comes from the fact that by using just a few samples, the estimate of the average gradient is noisy [1].

Stochastic Gradient Descent with Momentum (SGDM): It is an extension of SGD that implements a momentum, that is an inertia in the gradient that helps the algorithm to escape local minima of the objective function and reach its global minima [2].

An overview of both algorithms can be found in [1]

2.2 Federated Learning

FL is a new approach to ML that consists in collaboratively training a shared ML model using many devices, while keeping their raw data local. This is achieved by sharing only the local model update to a central server, that performs global model aggregation and distribution, synchronizing the participating devices, namely, the clients, as shown in Figure 2.1.

The term was first proposed in 2016 in the work by McMahan et al. [3]. The authors' goal was to describe a system that could leverage sensitive data present at edge devices to train a model. In their own words "We investigate a learning technique that allows users to collectively reap the benefits of shared models trained from this rich data, without the need to centrally store it.".

[1] http://deeplearning.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/

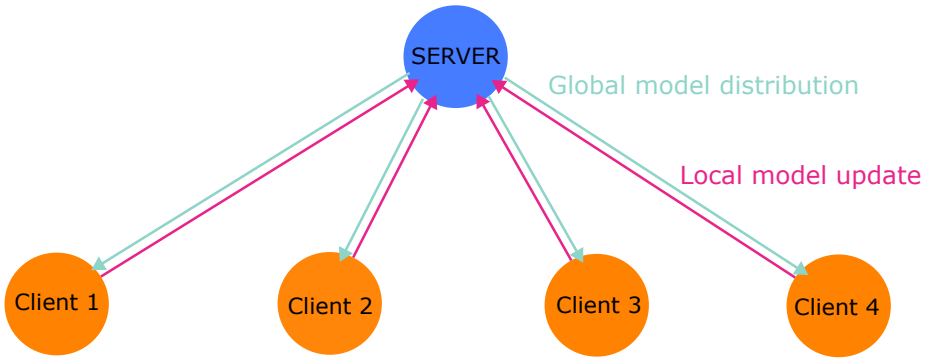


Figure 2.1 – Simplified scheme of a FL system.

The main reason for the recent research focus on this topic is the increasing attention from the public and legislators to data privacy, as demonstrated by more stringent legislation such as the General Data Protection Regulation (GDPR) [4].

2.2.1 Workflow

The process of a FL task can be summarized in six main steps [5], represented in Figure 2.2.

The process begins with client selection (step 1), that is the definition of which clients will participate on that training round according to some suitable criteria. It can be based on characteristics of the local dataset of the clients, or on their resources, such as battery. Then, local learning model updates (step 2) are calculated at each device using their local dataset. The choice of the model and optimization algorithm depends on the task at hand.

Step 3 consists in quantization of the model updates to reduce the number of bits used for its representation and thus improve the communication efficiency. Successively, an encryption algorithm is applied (step 4) to avoid leaking private information about the clients from the model updates, and a suitable channel coding technique is applied (step 5).

After these steps, the local model updates are sent to the aggregation server. The server constructs a global model by feeding the model updates of the clients to a suitable aggregation algorithm (step 6), and then distributes it back to the clients, which will repeat the procedure from the beginning. In this step, blockchain techniques can be applied to achieve a better robustness. Each one of these complete cycles is referred to as a communication round.

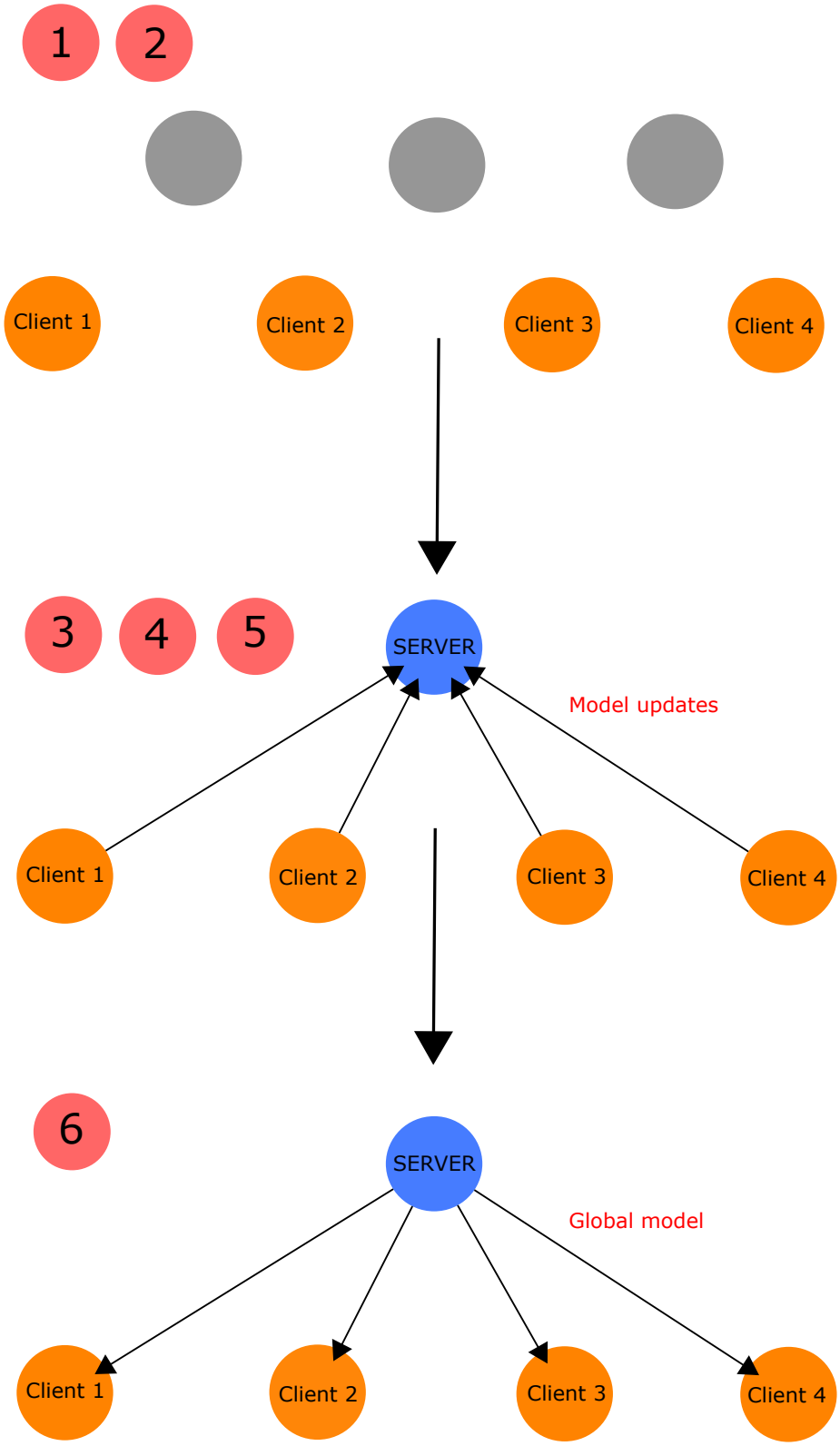


Figure 2.2 – Simplified scheme of the FL workflow of a communication round inspired in the steps described in [5]. The steps corresponding to each image are reported in the numbered circles.

2.2.2 Categories

The sample space describes how the samples are distributed to the different clients, whereas the feature space describes which features from these samples each client exploits. According to how data is distributed over the sample and feature space, 3 classes can be defined [6]:

Horizontal Federated Learning (HFL) consists in scenarios in which the clients have local datasets that share the same feature space, but with different samples.

Vertical Federated Learning (VFL) refers to the cases in which the local datasets share the same sample space, but have different feature spaces.

Federated Transfer Learning (FTL) consists in cases in which the datasets differ both in sample and in feature spaces.

Another categorization is defined in [7], according to the federation scale, defined by the number of clients and their data quantity.

Cross-Device: refers to the case with a large number of clients, each of which with a limited size of data. This is usually the case for IoT applications and may incur in difficulties in the training procedure due to the resources constraints on the devices and the scale of the system.

Cross-silo: on the other hand refers to cases with a small number of clients, each of which with large amounts of data. This is usually the case for applications that employ as clients different organizations or data centers, often with large computational resources.

2.2.3 Advantages

As previously discussed, the main advantage of adopting the FL approach is its increased data privacy due to the fact that it keeps the clients data local, and shares only model updates to the central aggregation server for global model construction. This may have an important impact in applications that deal with intrinsically sensitive data, such as healthcare applications. The problem was described in [8]: "For example, data of different hospitals are isolated and become “data islands”. Since each data island has limitations in size and approximating real distributions, a single hospital may not be able to train a high-quality model that has a good predictive accuracy for a specific task. Ideally, hospitals can benefit more if they can collaboratively train a machine learning model on the union of their data". By employing FL, such applications can be developed while respecting the clients' privacy. Other advantages from such a setting, described in [9], are a better use of the communication resources and power, since the model updates have a much lower dimensionality than the dataset, and thus transmiting it to the central server requires less bandwidth and power. It also has the potential to enhance learning

quality by leveraging the computational resources and diverse datasets of many devices.

2.2.4 Challenges

The FL approach brings many advantages, but it also comes with a new set of challenges that need to be addressed by research, as explained by Li et al. [10]. An important challenge regards the communication cost between clients and aggregation server, which may become an important bottleneck. This may be due to limited resources at the device, such as power, or due to the saturation of the network, especially in cases with a massive number of devices. The size of the messages containing the model updates depends on the model size, reaching the range of gigabytes for more complex models with millions of parameters, and can become a very significant problem when the number of clients is elevated [11]. The problem is confronted from two different directions: Reducing the number of communication rounds and reducing the size of the messages at each round [10].

Another challenge for FL systems refers to heterogeneity. System heterogeneity refers to the variability in devices acting as clients in terms of resources such as CPU, memory, battery and connectivity. This may increase the difficulty in dealing with issues such as fault tolerance and stragglers, which refers to the phenomenon of less capable devices slowing down the entire training procedure. Statistical heterogeneity refers to the fact that devices collect data that varies in terms of number of samples and distribution, and thus break the common assumption of i.i.d. data of many optimization algorithms. This requires the optimization of such algorithms to the FL setting, in order to achieve reasonable performances.

The last mentioned challenge relates to the privacy of clients' data, as the model updates can still leak sensitive information. The solutions explore cryptographic algorithms and their impacts on performances of the FL task.

2.2.5 Aggregation algorithms

The work of McMahan et al. [3] that first defined FL has also proposed the first aggregation algorithm, named Federated Averaging (FedAvg), which consists in the simple averaging of local model updates from the clients at the aggregation server. The algorithm is reported in Algorithm 2.1.

It begins by initializing the weights on all clients. Then, for each communication round, it collects the model updates from the clients and performs a weighted averaging, with weights based on the number of samples of that given client with respect to the total number of samples. Finally, it updates the global model with the results of the averaging, and distributes it back to the clients. The clients produce the model updates by applying the SGD algorithm with a specified batch size B

repeatedly over the specified number of epochs E. If another optimization algorithm were to be used, this is the point of the algorithm that should be edited.

According to Khan et al. [5], FedAvg may present difficulties in converging in heterogeneous environments, that is, in non-i.i.d. scenarios. This is because the local objectives of the clients might be too different, and they may get stuck in their local optima instead of converging to the global one. A solution has been proposed in [12], a variation of FedAvg called FedProx, which adds a proximal term to the local model loss function in order to restrict the local updates more to the global model and thus reduce the impact of heterogeneous data on the training procedure.

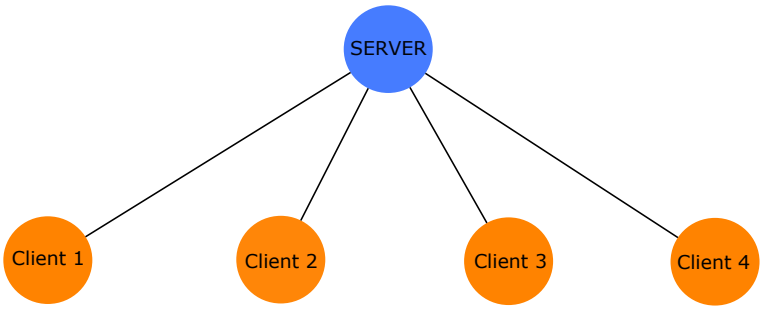
2.3 Decentralized Federated Learning

DFL is a variation of FL that consists in eliminating the central aggregation server, and instead relies on the exchange of model parameters directly between the clients, which perform the aggregation themselves. In Figure 2.3, the two types of FL are represented.

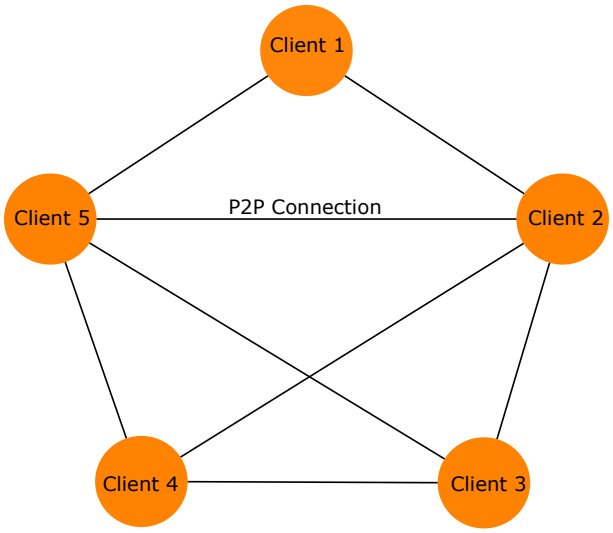
This new approach has been proposed in order to address the problems with Centralized Federated Learning (CFL), which can be summarized in the optic of DFL [13]:

1: ***FedAvg pseudo-algorithm***
2:
3: ***Aggregation Server executes***
4: Weights initialization w_0
5: **for** round t in 0,1,... **do**
6: **for** client k in K **do**
7: $w^k_{t+1} \leftarrow ClientUpdate(k, w_t)$
8: **end for**
9: $m_t \leftarrow \sum_k n_k$
10: $w_{t+1} \leftarrow \sum_k \frac{n_k}{m_t} * w^k_{t+1}$
11: **end for**
12:
13: **Client k executes**
14: ***ClientUpdate(k,w):***
15: **for** epoch e in E **do**
16: **for** batch b in B **do**
17: $w \leftarrow w - \eta \nabla l(w; b)$
18: **end for**
19: **end for**

Algorithm 2.1 – FedAvg pseudo-algorithm inspired by the one reported in [3]. K is the total number of clients; E is the number of local epochs; B is the local batch size; η is the learning rate.



(a) Centralized Federated Learning.



(b) Decentralized Federated Learning.

Figure 2.3 – Comparison between CFL and DFL. Image inspired by images in the work [9].

- Communication overhead and latency when number of participant clients is high;
- Hardware and connectivity heterogeneity, with potential stragglers;
- Central aggregation server represents a bottleneck and a single point of failure;

2.3.1 Advantages

The new approach presents several advantages, such as [13]:

- Fault-tolerance, as the system depends on no single node;
- Self-scaling network, with ease of adding new nodes;

- Zero-cost infrastructure for developers, since it leverages only the clients' computing power;
- Diminished risk of saturating the networks resources, since parameters do not need to be communicated to a central server;

The advantage of the elimination of the single point of failure in the central aggregation server has a particular beneficial impact for applications whose nodes are highly unreliable, such as in Unmanned Aerial Vehicles (UAV) applications, as highlighted in [14].

2.3.2 Network topology

The network topology defines the interconnection between the nodes participating in the DFL task, in terms of which pairs of nodes will exchange parameters at each communication round. It is a vital characteristic to analyze, as it has significant impact on convergence, generalization, overhead, robustness, flexibility and security of the system.

The work [13] listed 3 classes of topologies and their characteristics:

1) **Fully connected networks:** Each node is connected to every other node. It has issues of communication overhead, since each node needs to send parameters to every other node, and scalability, as adding an extra node leads to an increase in the complexity of the entire network, as every other node needs to establish a link with it. However, it is very reliable, as all nodes can still communicate if any node fails.

2) **Partially connected networks:** participating nodes have connections to only some nodes in the network. They have lower communication overhead, as nodes do not need to send their parameters to every other node. It also decouples the scalability of the network from the ability of a node of dealing with new connections. However, it may lead to delays in convergence, since the parameters of a node can take several training rounds to reach every other node in the network, since they need to go through intermediate nodes. Reliability also suffers, as failures in some nodes may lead to a delay or interruption in the communication. This class is generic, and encompasses several different topologies, such as star or ring-shaped ones. The specific topology chosen affects the degree to which the previously mentioned characteristics change.

3) **Node clustering:** Nodes are grouped into clusters based on some similarity criteria.

Figure 2.4 shows an example of topology of each one of the classes discussed.

2.3.3 Synchronization

In [13] a classification regarding the synchronization of nodes is defined:

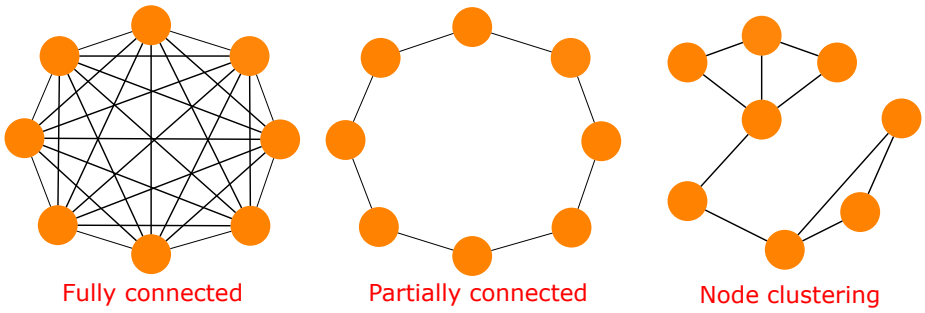


Figure 2.4 – DFL topologies, inspired in topologies presented in the work in [13].

1) **Synchronous:** Each node trains for a determined number of epochs before exchanging the model parameters with the neighbors and aggregating. The exchange of parameters happens only when all nodes in the network have finished training, and thus this may lead to an increased convergence time due to slow nodes.

2) **Asynchronous:** There is no synchronization between the nodes, and they are allowed to send and receive parameters at any given moment. This may lead to faster convergence time, but may cause issues of lower generalization and staleness. It is well suited for cross-device FL since they tend to have heterogeneous computational capability and connectivity.

3) **Semi-synchronous:** The synchronization point is when the slowest node finishes one epoch. In the mean time, the other faster nodes train for as many epochs as they can. When the synchronization point is reached, nodes exchange parameters and aggregate.

2.4 Object Detection

OD is a Computer Vision (CV) task that deals with localizing and classifying objects in a digital image, or as [15] described: "What objects are where?". The location of the object in the image is usually defined by a bounding box. In the same work, the authors described it as being the basis for many other CV tasks such as instance segmentation and object tracking, and it is used widely in many real-world applications such as autonomous driving, robot vision, among others.

2.4.1 Metrics

The formulas and explanations can be found at [2].

Precision: Ratio of correctly detected objects in an image. It indicates how reliable are the model's predictions. It assumes values from 0 to 1. It is measured as

[2]https://www.v7labs.com/blog/mean-average-precision

the rate of True Positive (TP) with respect to the total number predictions, including False Positive (FP).

$$Precision = \frac{TP}{TP + FP} \tag{2.1}$$

Recall: Ratio of correctly detected objects (TP) to the total number of ground truth objects including False Negative (FN), given by $TP + FN$. It indicates the degree to which the model is detecting objects that should have been detected. It assumes values from 0 to 1.

$$Recall = \frac{TP}{TP + FN} \tag{2.2}$$

Intersection over Union (IoU): It is a number between 0 and 1 that describes the overlap between the predicted bounding box and the ground truth box. It can be used as a threshold to classify a prediction as a true positive and is given by:

$$IoU = \frac{AreaOfOverlap}{AreaOfUnion} \tag{2.3}$$

Average Precision (AP): It's a class specific metric that measures the accuracy of the model in predicting both the presence and the location of objects in an image. It was first defined in [16] and it is calculated as the area below the Precision-Recall curve, that is generated by calculating the Precision and Recall of the model's predictions by varying the confidence threshold, which is a metric associated to each prediction and indicates the level of certainty regarding the presence of an object in a particular region of an image. This metric is also defined in relation to an IoU threshold, that defines the minimum overlap between the predicted and ground-truth boxes over which predictions are considered valid, and is the responsible for the characteristic of measuring the location accuracy of the model. For example, AP@0.5 means that the AP will be calculated by considering as true only the predictions for which the predicted box overlaps the ground-truth box by more than 50%. Higher AP values indicate a better performance, and it assumes values between 0 and 1.

mean Average Precision (mAP): It's the average of the AP over all classes, and is used as a final metric of performance. For N classes, it is given by:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{2.4}$$

mAP@.5: It's the average of the AP over all classes calculated at an IoU threshold of 0.5. According to [15] it is the de facto standard metric for object detection.

mAP@.5:.95: It corresponds to the average of the mAP values obtained by varying the IoU threshold between 0.5 and 0.95 with a step size of 0.05. Zou et al.

[15] describes it as encouraging more accurate object localization, and important to some real-world applications that require a higher precision.

2.5 MS COCO

MS COCO [17] stands for Microsoft Common Objects in Context, and it is a popular dataset for computer vision tasks such as OD. The dataset has been created with three goals in mind: Having non-canonical view of objects, which means having objects in non usual positions, partially obscured or in the background; Having contextual information in images, which means having multiple objects of multiple categories in a single image representing a scene; Having precise 2D localization of objects in the image.

2.5.1 Statistics

MS COCO contains 91 object classes, with over 2,500,000 labeled instances in 328,000 images. The goal of creating a dataset with images containing contextual information leads to a higher number of object classes and instances per image when compared to other datasets, such as PASCAL VOC [16], another popular dataset for OD: on average MS COCO contains 3.5 categories and 7.7 instances per image, whereas PASCAL VOC contains only 2 categories and 3 instances per image. Furthermore, only 10% of images on MS COCO contain only one object category, compared to 60% in PASCAL VOC.

2.6 YOLO model

You Only Look Once (YOLO) was proposed by Redmon et al. [18] with a new approach to OD. It is a state-of-the-art object detection system known for having fast performance while maintaining high accuracy. It works the following way: it divides the image into grid cells, and for each of them it independently predicts if it contains the center of an object, the dimensions of the bounding box and the class of the object; each grid can predict multiple bounding boxes, each of which has a confidence metric, that reflects the certainty that the box contains an object, and a conditional class probability, which indicates the probability of the object belonging to a given class. Differently from traditional OD models, that follow a two-stage approach, YOLO unifies all the work into a single neural network, which leads to its excellent performance in terms of speed. Figure 2.5 shows a representation of this procedure.

Since its introduction, several subsequent versions have been proposed to improve its performances.

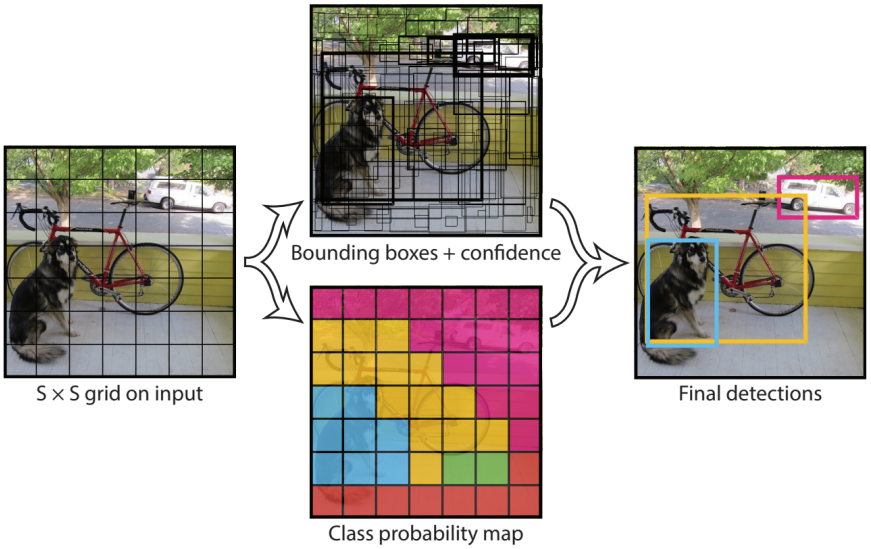


Figure 2.5 – YOLO functioning scheme, taken from [18].

2.6.1 YOLOv5

YOLOv5 [3] has no official paper, but it is based on YOLOv3 [19].

The creators of YOLOv5 made available different models with varying sizes and performance, to allow more flexibility in its implementation. Table 2.1 shows the accuracy of the models at a training checkpoint of 300 epochs, with default settings and declared parameters.

Model	Size (pixels)	$mAP_{val}50-95$	mAP_{val} 50	Params (M)
yolov5n	640	28.0	45.7	1.9
yolov5s	640	37.4	56.8	7.2
yolov5m	640	45.4	64.1	21.2
yolov5l	640	49.0	67.3	46.5
yolov5x	640	50.7	68.9	86.7

Table 2.1 – Table of performances of different model sizes at training checkpoint of 300 epochs with default settings and declared parameters. Values taken from [3].

[3] https://github.com/ultralytics/yolov5

2.7 Related Work

2.7.1 Federated Learning

The surveys [5], [7] and [9] present a detailed introduction to the main concepts of FL and explore in depth its current trends, including the emergence of DFL. The applications are discussed with a special focus on IoT use cases, highlighting the advantages of the FL approach and its variations.

A number of works have been dedicated to exploring the FL performances and how it compares to the centralized learning paradigm. In [3] the authors have coined for the first time the term FL, proposed the FedAvg algorithm and validated its performances. First they have experimented with two different models for a digit classification task on the MNIST dataset considering two cases: i.i.d. distribution of data, in which each client has the same distribution of data samples; non-i.i.d., in which the clients contain only a few classes in their local dataset. The results show that a non-i.i.d. distribution of data is more challenging for the algorithm, as it takes more communication rounds to converge to the target accuracy, but it still converges, providing a confirmation of its robustness to data heterogeneity. The second experiment is performed with a LSTM language model for next character prediction over a dataset created by the authors to reflect a more complex scenario, in which clients' datasets vary in terms of number of samples and are also chronologically separated. The results show the same tendency as the first experiment. The authors have provided an analysis of the impact on performances when varying the fraction of clients participating per round, and the amount of computation performed on each client, by varying the number of epochs and the batch size for each communication round. Another experiment was performed on the CIFAR-10 dataset for image classification task with i.i.d. data over the clients, and a centralized ML benchmark was used for comparison. The results show that for some parameters combinations, the performances of FedAvg are comparable to those of the centralized approach, in terms of convergence speed and final accuracy.

In [20] the issue of data heterogeneity has been further explored. The authors have used a Convolutional Neural Network (CNN) trained on MNSIT, CIFAR-10 and speech commands datasets under i.i.d. and non-i.i.d. data distribution over 10 clients, using FedAvg. For the i.i.d. case, the performances were comparable to centralized learning. Instead for non-i.i.d. the accuracy can drop significantly, up to 55%. They then associate this drop to the weight divergence due to non-i.i.d. data, which quantifies the difference of weights from two different training processes with the same weight initialization, and is measured using Earth Movers Distance (EMD). Lastly, a solution is proposed consisting of creating a small subset of data shared

globally among all clients, which can bring up to a 30% improvement in accuracy from their experiments.

Other works explored FL for OD tasks. The work in [21] analyzes the effects of non-i.i.d. data for OD models using FedAvg. The experiments are done with SSD300 model trained using Pascal VOC 2007 and 2012 [16]. For i.i.d. data, the model achieves a final accuracy comparable with centralized learning. The authors also analyzed the impact of the number of clients, stating that an increase from 4 to 16 clients leads to an increase in the number of rounds needed to converge, in this case of 20%. They also observe that increasing the number of local epochs leads to a faster convergence. For non-i.i.d. data, the accuracy drops by 7%, but for extreme non-i.i.d., it reduces from about 76% to only 52%. This drop in accuracy is attributed to weight divergence between the clients trained with local data, measured with Kullback-Leibler Divergence (KLD), and a possible solution named FedAvg with Abnormal Weight Supression (AWS) is proposed, based on the trimming of the weights that are far from the average of the normal weights. This approach lead to a small improvement in the accuracy at the expense of a larger convergence time for the non-i.i.d. case, and was not effective in the extreme non-i.i.d. case.

In [22] a real-world image dataset is proposed, with realistic elements of a FL application such as non-i.i.d. data and unbalanced datasets in terms of number of samples. The images come from street cameras and can be grouped into geographical regions to provide realistic assumptions of applications. The experiments were performed with Faster R-CNN and YOLOv3. For both models, the results indicate that increasing the level of heterogeneity in the data leads to lower accuracy and convergence time, whereas increasing local computation and using pretrained weights lead to a faster convergence time.

In [23] FedML was proposed. It consists in an open research library and benchmark for FL, and its objective is to provide a standardized environment to facilitate new algorithms development and fair comparison among different approaches. Among its characteristics are: Support of different computing paradigms, such as on-device, distributed and single-machine simulation; Support of diverse FL configurations, different topologies, information exchange schemes and training procedures, including DFL; Standardized FL algorithms implementations, such as FedAvg; Standardized FL benchmarks, with datasets, metrics and baseline results to facilitate comparison. The architecture of the library is modular, with each module responsible for a specific part of the FL system, such as generating the topology or defining the messages exchanged among the nodes. The core of the library is accessible from the outside through easy Application Programming Interface (API)s. On top of this library, other libraries targeted at specific applications have been built.

One such library is FedCV, proposed in [24]. Motivated by the lack of research on FL algorithms for CV tasks, and the resulting poor performance of current combina-

tions of the two fields, the authors created a standardized environment that includes many of the features that limit this research on other frameworks, such as multi-GPU training, in order to allow heavier models to be trained, and a standard way to partition data among clients, facilitating comparison among different approaches. FedCV supports three different tasks: Image Classification, Image Segmentation and OD.For each of them the authors have provided standard datasets and baseline experiments to facilitate comparison.

For OD the authors use the COCO dataset and the pretrained YOLOv5 model with FedAvg. Evaluation was performed using the mAP@.5 metric. In the first experiment, the number of clients was set to 4 and the effect of the learning rate was measured. The authors concluded that a value of 0.01 was the best. For the next experiment, the learning rate was fixated to the best value and different tests were performed with 4 and 8 clients with different model sizes. The conclusions were that the YOLOv5 with FedAvg struggles to reach the performance of the centralized version under non-i.i.d. data distribution, because some of the tricks used in the latter could not be directly transported to the former. Increasing the number of clients lead to a penalty both in convergence speed and final accuracy. Finally, a comparison among the different model sizes was performed to quantify the amount of parameters exchanged and the training time required to achieve the results presented.

2.7.2 Decentralized Federated Learning

The survey [13] analyses the main fundamentals that differentiate DFL from CFL: federation architectures, topologies, communication mechanisms, security approaches. The authors also define Key Performance Indicators and proceed to analyze the existing applications with them. Then, the lessons learned and the current research trends are highlighted.

In [25] Decentralized FedAvg with Momentum (DFedAvgM) is studied as an alternative to CFL to increase the communication efficiency and privacy. The algorithm consists in peforming local optimization at the clients using SGDM, exchanging the quantized model updates through peer-to-peer communication and aggregating locally by averaging the model updates as in FedAvg. An extensive mathematical analysis of the convergence is provided, and then the proposed algorithm is validated by experiments in image classification and language modelling tasks using Deep Neural Network (DNN)s, with a fixed ring topology. The first experiment consists in training two DNNs, a 2NN and a CNN, for MNIST digit recognition using 100 clients and both i.i.d. and non-i.i.d. data partitioning. For i.i.d., the results indicated that increasing local epochs leads to a faster convergence, and DFedAvgM performs almost identically to FedAvg. For non-i.i.d., increasing local epochs lead to no visible improvement, and the achieved accuracy of DFedAvgM was about 10% lower than Fe-

dAvg, since neighboring nodes may not have enough data to cover all classes. In both cases, the proposed algorithm showed that it improves communication efficiency and privacy. The second experiment consists in training a stacked character-level LSTM language model for next character prediction using the SHAKESPEARE dataset with a non-i.i.d. partitioning. The results indicate that a higher communication cost can lead to a faster convergence, and increasing the local epochs deteriorated the performances. The last experiment consists in training a ResNet20 for image classification on CIFAR10 dataset, and the results were analogous to the ones on the first experiment.

The work in [26] also investigates the convergence of DFL using SGD and averaging the model updates exchanged among the clients. The effects of communication and computation frequency are evaluated on 10 clients over CIFAR-10 and MNIST datasets with two different topologies: ring and quasi-ring (ring with few extra connections). The results indicate a slightly quicker convergence for the latter.

In [27] the authors highlight the problems of CFL for single point of failure and scalability issues, and propose two DFL algorithms: Consensus based Federated Averaging (CFA) and Consensus based Federated Averaging with Gradient Exchanges (CFA-GE). CFA consists in performing the optimization of the local model with the local dataset using SGD, exchanging the model updates and combining the updates from the immediate neighbors through a weighted-averaging, whose weights can be optimized according to several criteria. CFA-GE uses both the local model updates and the local gradients of the neighbors in order to improve the convergence and minimize the number of communication rounds, at the expense of a more intensive use of the D2D links and more local computing. The algorithms are first tested in a simplified scenario using a Neural Network (NN) for digit classification on MNIST, with 4 clients connected in a line topology, and their loss over the validation dataset is plotted. Three other algorithms are tested to provide a comparison: Centralized ML, FedAvg and isolated training (no cooperation among the 4 clients). For i.i.d., all algorithms perform worse than the Centralized ML, and better than isolated training, confirming the advantage of cooperation among devices. CFA-GE performed similarly to the FedAvg, whereas CFA presented a difference depending on the node: the nodes connected to only one other node converged more slowly, while nodes connected to two other nodes performed similarly to CFA-GE. For non-i.i.d., the penalty was bigger when the dataset of the central nodes, connected to two other nodes, was the least representative.

The authors then validate the algorithms in a more complex large-scale and dense network of Industrial Internet of Things (IIoT) devices sensing their surroundings for a Human Robot Collaboration task. The dataset is non-i.i.d., and the authors have varied the number of devices from 30 to 80, connected neighbors per device from 2 to 10 and model, as well as provided a Centralized ML and FedAvg version

for comparison. CFA approaches slowly the curves of FedAvg and Centralized ML, and an increase in the network density (conenctions per node), leads to a faster convergence speed. CFA-GE behaves similarly to FedAvg, and with a higher network density, it approximates the Centralized ML curve. The improvement of the increase of network density is more visible for smaller networks, in terms of number of nodes. In the conclusion, the authors discuss the need of evaluating the proposed method over more complex models and tasks.

The work in [28] extends the ideas in [27] by proposing an algorithm based on diffusion techniques and an adaptive combination rule for the gradient information coming from the neighboring nodes, with the objective of increasing the convergence speed, especially for non-i.i.d. scenarios in which the gradients of the different clients tend to point to different directions. The authors validated their ideas on a classification task using the MNIST dataset with 4 and 20 clients, and have provided benchmarks for comparison using centralized ML, isolated training and the consensus algorithms proposed in [27]. The results show that the proposed algorithm converges faster than the other distributed ones for non-i.i.d. cases. It is also possible to observe that in all cases the distributed algorithms performed worse than the centralized ML and that a higher number of clients leads to a slower convergence.

The work in [29] consists in the analysis of the impact of topology on the convergence of a DFL system with a simple averaging implemented as the aggregation algorithm. The experiments were performed on two classification datasets, distributed among clients in a balanced and i.i.d. manner, and the models used were a linear Support Vector Machine trained with SGD, and a Logistic Regressor classifier. The experiments evaluated the impact of different number of clients with 3 different topologies classes: Regular random graph, which consists of nodes with same degree; Small-world graph, in which the distance between two random nodes is proportional to the logarithm of the graph's size; Scale-free graph, in which there are nodes, called hubs, with a much larger degree than the others. The results indicate that clustered networks seems to perform better for DFL, and small-world topologies tended to perform better for small scale networks, whereas Scale-free ones performed better for large scale networks.

The work in [30] begins by presenting that the impact of non-i.i.d. data distribution for sparse topologies, such as rings, is very significant, and it slows by a large amount the convergence of the model. Then, the authors present D-Cliques, a sparse topology that obtains similar convergence speed to a fully connected network even with non-i.i.d. data distribution, by grouping nodes into subsets, called cliques, that have a data distribution in its union representative of the global distribution. The experiments were performed on the MNIST and CIFAR10 datasets.

The work in [31] analyses the employment of DFL for UAV applications. First, the authors discuss the problem of single point of failure of traditional FL for applications

with highly unreliable nodes and links. Then, they propose Serverless FL for UAV Networks (SELF-UN), a novel architecture based on DFL, whose main benefits are: (i) increased robustness due to the lack of a central node coordinating the entire process; (ii) increased flexibility and agility to the network, that can seamlessly adapt to changing topologies and number of participants as required by the environment.

Chapter 3

System design and implementation

The goal of this thesis is to understand the impact of topology on the performance of the training task, when subject to a non-i.i.d. partitioning of the OD MS COCO dataset. Leveraging the insights from this analysis, a routing algorithm for the reduction of the number of connections between clients with minimal degradation of the performance is proposed and validated.

In order to provide such an analysis, the following steps were necessary, which will be further explored in this chapter:

- Implementation of a DFL system for OD tasks.
- Definition of a partitioning method for the MS COCO dataset and analysis of its statistical distribution.
- Definition of an approach to the system evaluation.
- Design of the routing algorithm.

This chapter will begin with a high-level description of the DFL OD system that will be analysed. Then, I will present how I have extended the existing FedML [23] library to implement it. Next, I will present the study of the dataset statistics under a non-i.i.d. partition, and highlight its particularities. Then, I will discuss the approach to the evaluation of the system and the methodology. Finally, I will explain the designed routing algorithm.

3.1 High-level description of the system

A DFL OD system consists in a set of nodes connected to each other to form a given topology. Each node has an instance of the OD model in use, and a local dataset. The nodes train their models with a certain optimization algorithm for

a given number of epochs with their local dataset, and then exchange the model weights with their immediate neighbors, completing a communication round. After exchanging the weights, each node executes an aggregation algorithm with the received weights from its neighbors, and updates its own model with the result. The system is synchronous, and thus a communication round ends only when all clients have updated their models.

The chosen OD model for the system is the YOLOv5 [4] model, because of its good performance in terms of accuracy and speed compared to others and widespread adoption. It also offers different model sizes, allowing to adjust to different applications. The chosen size is the Nano (YOLOv5n), which is the smallest in terms of number of parameters and quickest to train, allowing to save computing resources.

The dataset used for the experiments is the COCO [17], a popular one for OD tasks. Due to its highly contextual nature, partitioning it in a non-i.i.d. way is challenging, and leads to a highly unbalanced distribution of data among the clients, the consequences of which, combined with the topology, is the target of the exploration. Furthermore, it has been used by the creators of YOLOv5 to provide pretrained weights in the model's repository together with their performance, thus providing a reliable initial benchmark to give a reference point to the system being built.

The aggregation algorithm is the simple FedAvg algorithm, adapted to work with the DFL setting. That means that in the averaging, the weight of the model updates is proportional to the amount of samples of the corresponding client in relation to the total number of samples of all nodes connected to the client performing the averaging. The pseudo-algorithm showing the adaptation is reported in Algorithm 3.1.

[4] https://github.com/ultralytics/yolov5

FedAvg for DFL pseudo-algorithm

Client n executes
K = Group of neighboring clients + client n itself
for round t in 0,1,... **do**
  for client k in K **do**
    $w^k_{t+1} \leftarrow OptimizationAlgorithm(k, w_t)$
  end for
  $n_k \leftarrow$ Number of samples of client k
  $m_t \leftarrow \sum_k n_k$
  $w^n_{t+1} \leftarrow \sum_k \frac{n_k}{m_t} * w^k_{t+1}$
end for

Algorithm 3.1 – FedAvg for DFL pseudo-algorithm.

For example, the combination rule for Client 1 reported in Figure 3.1 is given by:

$$w_1^{i+1} = \frac{S_1 w_1^i + S_2 w_2^i + S_3 w_3^i}{S_1 + S_2 + S_3} = \frac{2000 * w_1^i + 3000 * w_2^i + 1500 * w_3^i}{6500} \tag{3.1}$$

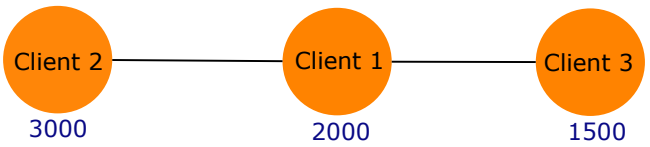


Figure 3.1 – Example of combination rule for a given client in the DFL system.

3.2 Implementation

In order to implement with a standard approach OD models with DFL, FL and Centralized ML, I have decided to use a framework that had most of these features, and extend it in order to implement the desired system.

FedML[23] is a python research library based on PyTorch dedicated to FL algorithms. It includes many of the features of the desired system, such as standard implementations of popular FL algorithms, as well as datasets and benchmarks. Another interesting feature is that it has been designed to support single-machine simulations with Message Passing Interface (MPI), and allows for Graphics Processing Unit (GPU) training.

Of particular interest to the objective is that it has a backbone of the communication and topology definition of a DFL system developed. It also has a specific module, called FedCV [24], focused on CV tasks. This module includes OD tasks, and it provides a functional implementation with FL of the YOLOv5 model.

The goal thus became to combine the DFL backbone with the developed FL OD system in order to build the DFL OD system in an environment that is able to provide fair comparison with the FL and Centralized ML versions.

In this section I will describe in detail how the FL OD system and the DFL backbone work, and then I will describe how I have combined them to create the desired system.

3.2.1 FL OD system

The library has a modular design, that is, it is divided in files or group of files that are responsible for a specific part of the system, such as data handling or communication among nodes. The objective of this design is to make it easy to understand the system and extend it to the researcher's needs. For example, if an algorithm needs

to change only the communication between nodes, it is possible to edit that specific module while reusing all the other ones with perfect compatibility.

In order to explain the functioning of the system, I have defined 5 main modules, represented in the Figure 3.2. The modules are abstract groupings of different files that allow for an explanation based on their functionality, and do not necessarily reflect their organization inside the repository.

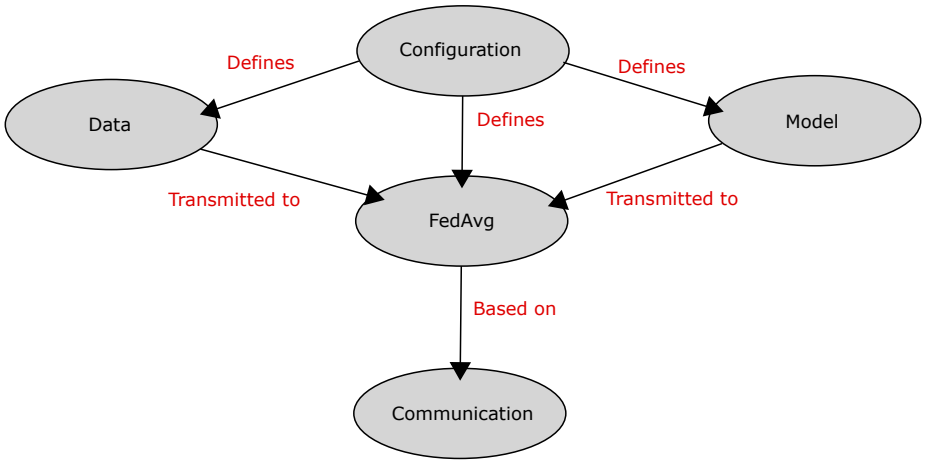


Figure 3.2 – FL OD modules and their relation to each other.

Configuration: This module provides the high-level command of the experiments, allowing for changes in the parameters, data, algorithm, number of clients, data distribution, GPU training and information logging. All the relevant information can be edited in a configurations file (.yaml). In this module there are also the files to launch the MPI processes, load the correct model inside the processes, and provide them with the correct data and configurations.

Data: This module is responsible for downloading the datasets and putting them in the right format to be used with the model. It also provides the partitioning of the dataset between the nodes according to different criteria and takes care of correctly loading them into the nodes.

Model: This module deals directly with the model code. It does that by providing the entire repository of the model inside itself, in this case of YOLOv5, and by creating APIs to access the desired functionalities inside of it, such as training and validation.

FedAvg: It is the core of the system that puts all parts together to generate a FL system. It is composed of: FedAvgAPI, which assigns the roles of server or client to the processes and initializes them; FedAvgClientManager and FedAvgServerManager, which deal with the synchronization and communication between the different nodes according to their roles; FedAVGTrainer and FedAVGAggregator, used by the managers as APIs to call the needed functions of the model; Message define, which specifies the content of the communication between the nodes.

Communication: Low-level module on which FedAvg relies to define the specific implementation of the communication on hardware.

3.2.2 DFL Backbone

The DFL backbone provided in the library is not a complete one, as it doesn't provide any connection with the model, does not perform training or validation and does not implement any aggregation algorithm on the node. It only provides the communication and coordination among the nodes according to the defined topology. The scheme is represented in Figure 3.3.

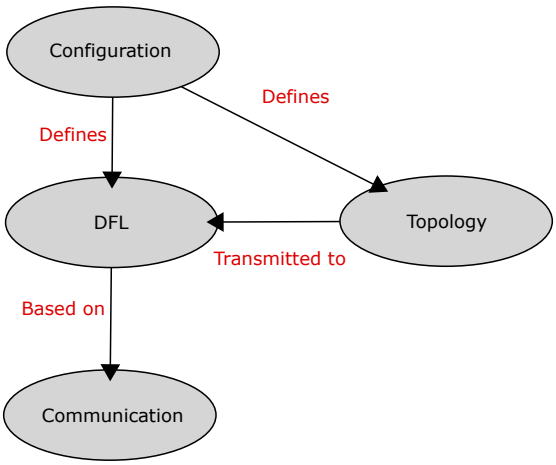


Figure 3.3 – DFL OD modules and their relation to each other.

The different modules are:

DFL: Backbone that provides synchronization and communication. It consists of: AlgorithmAPI, which initializes the topologies and the clients; DecentralizedWorker-Manager, which provides the communication and synchronization; Decentralized-Worker, an API for the model functionalities, not implemented in source code from library; Message define, for the content of the communication between the clients, which also needs to be implemented correctly to work with the model.

Topology: Generates a topology, represented as a matrix, according to the input information such as number of nodes and connections per node.

3.2.3 DFL OD system

The desired system is a combination of the two previously presented, illustrated in Figure 3.4.

In order to build this system the following operations were performed:

- Connect Model and Data modules to the DFL module.

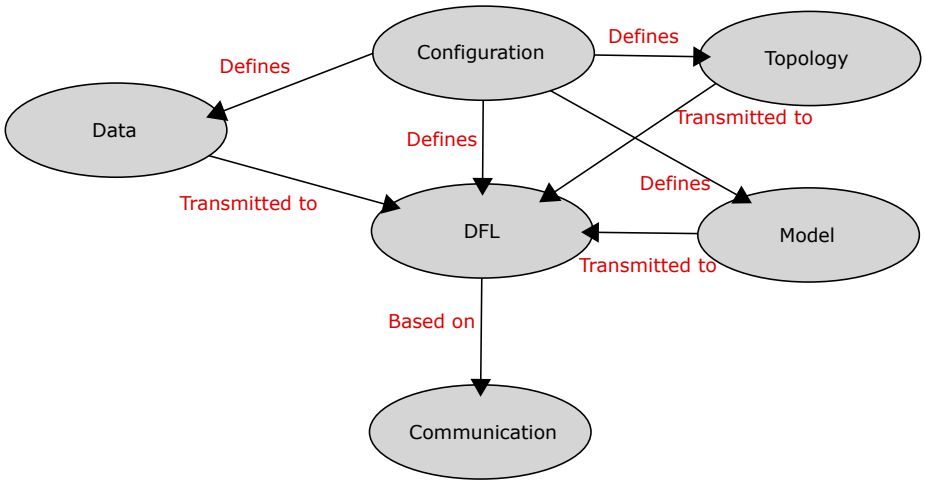


Figure 3.4 – DFL OD modules and their relation to each other.

- Implement model training in the DFL model at the correct synchronization points.
- Set content of messages to communicate model updates between the clients. Set write and reading of the message content to the right format.
- Implement the adapted FedAvg aggregation algorithm to combine the received model updates, and implement model updating with the result of the algorithm.

The resulting system correctly implements OD with a DFL approach, and allows to compare the results with the FL and Centralized ML versions built using the same modules, running on same data and environment.

3.3 Data partitioning

As explained in the chapter 2, MS COCO [17] is a dataset built to have contextual information, which means that each image tends to have more object instances and classes than other datasets, such as PASCAL VOC [16]. A consequence of that is that it becomes much more challenging to isolate the classes in order to provide a non-i.i.d. data distribution among different clients.

I have chosen to adopt the partitioning method for the COCO dataset proposed in the FedCV [24] paper, consisting of the following steps:

- **Step 1:** Calculate the frequency of classes over all images of the dataset, and sort classes by highest frequency;
- **Step 2:** Assign all images containing most frequent class to first client;

- **Step 3:** Remove these images from the dataset;
- **Step 4:** Iterate, assigning the classes to the successive clients;
- **Step 5:** When all clients have been assigned a group, stop.

The pseudo-algorithm in reported in Algorithm 3.2

This partitioning method generates a highly unbalanced non-i.i.d. distribution of data, since the most frequent class of the dataset appears in almost half of the total number of images.

The statistics of the distribution for four clients are reported in Figure 3.5, Figure 3.6 and Figure 3.7.

Dataset partitioning pseudo-algorithm

$D = Dataset$
for client n in $Num_{clients}$ **do**
  for class c in $Num_{classes}$ **do**
    $fs[c] = CalculateFrequency(D, c)$
  end for
  $fs_{sorted} = sort(fs)$
  $D_n = D$ where D contains $fs_{sorted}[0]$
  $D = D - D_n$
  $Client_n \leftarrow D_n$
end for

Algorithm 3.2 – Dataset partitioning pseudo-algorithm.

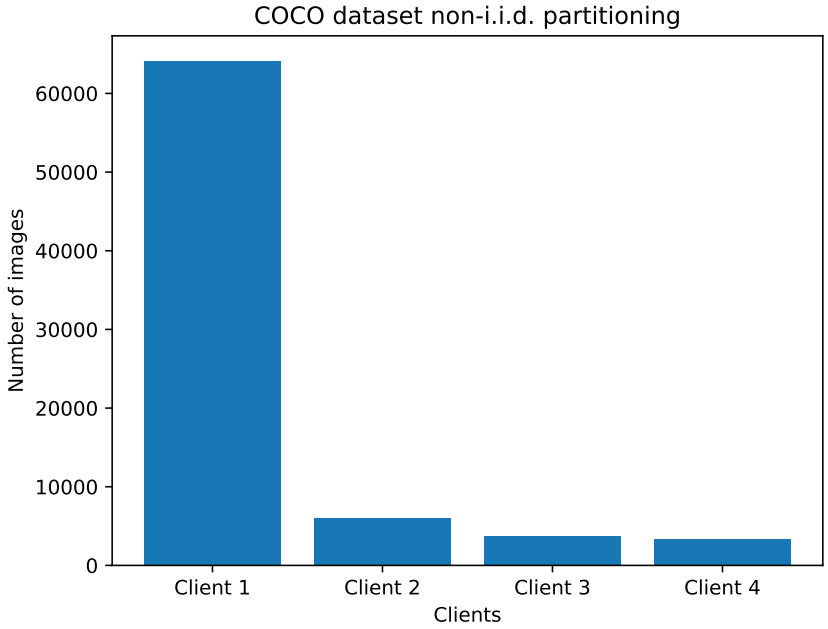


Figure 3.5 – Distribution of number of images in 4 clients using FedCV partitioning method.

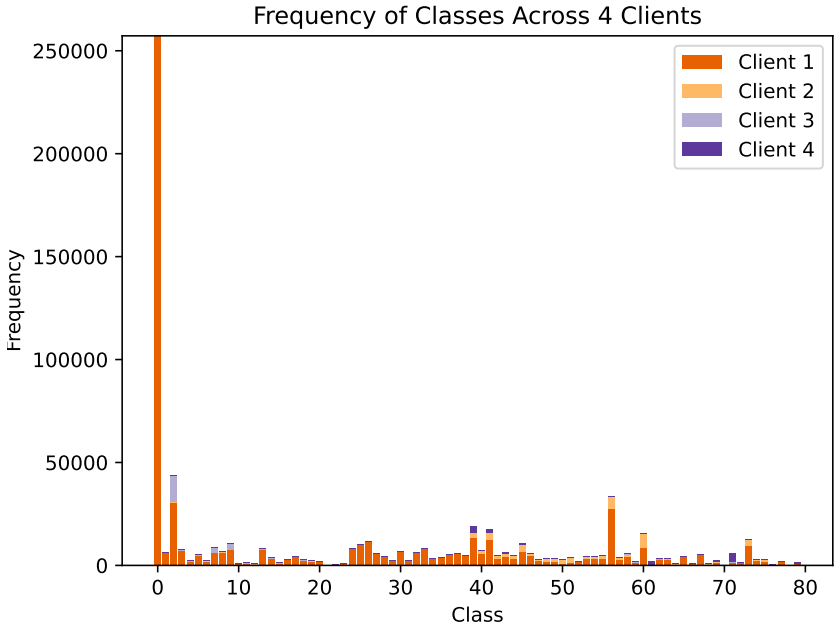


Figure 3.6 – Distribution of classes in 4 clients using FedCV partitioning method.

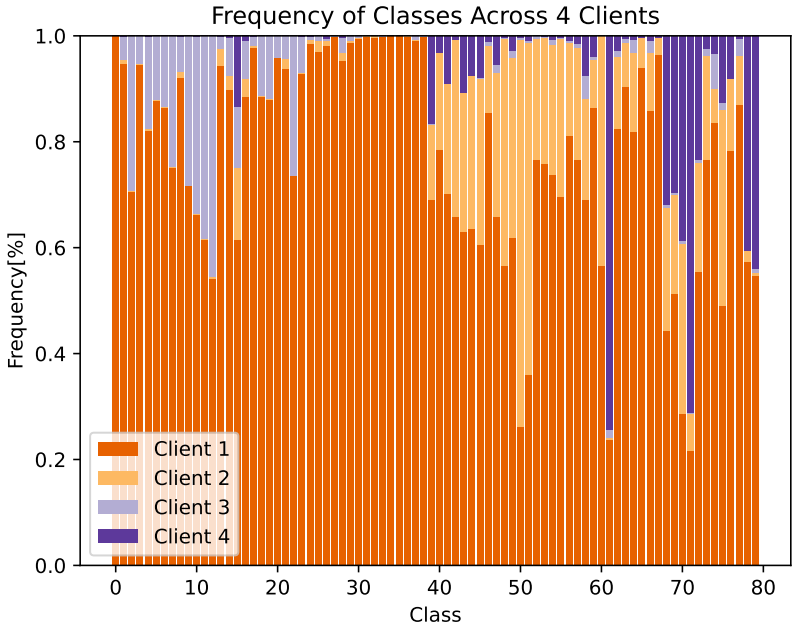


Figure 3.7 – Distribution of classes in percentages in 4 clients using FedCV partitioning method.

From Figure 3.5 and Figure 3.6, it is possible to observe that this partitioning method leads to an extremely unbalanced situation, with Client 1 concentrating most of the images and object instances of the dataset, and will be referred to as the dominant client. From Figure 3.7 it is possible to observe that a non-i.i.d. distribution has been obtained, as the clients concentrate well defined groups of classes. The exception is client 1, that contains a good amount of data of almost every class.

The graphs obtained by executing the algorithm for 8 clients are reported in Figure 3.8, Figure 3.9 and Figure 3.10, for which the same behavior can be observed.

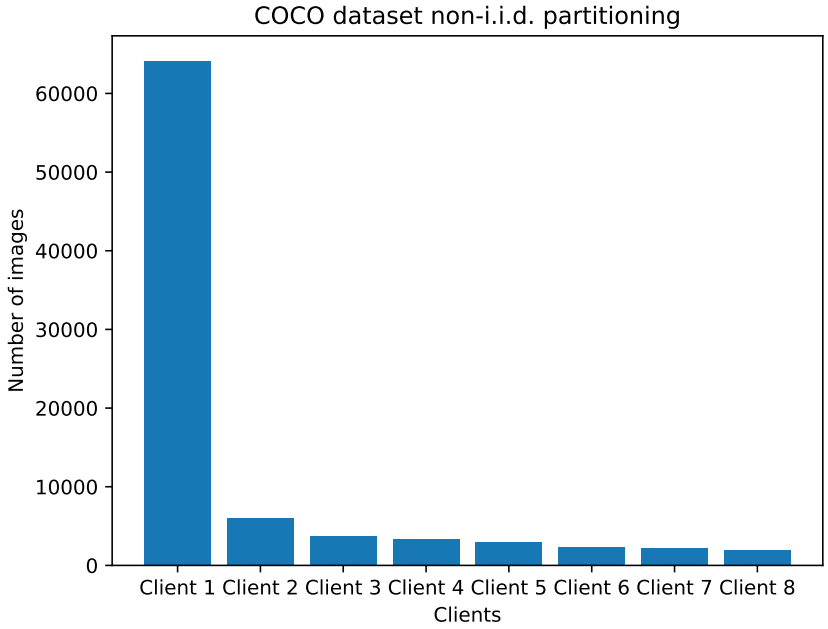


Figure 3.8 – Distribution of number of samples in 8 clients using FedCV partitioning method.

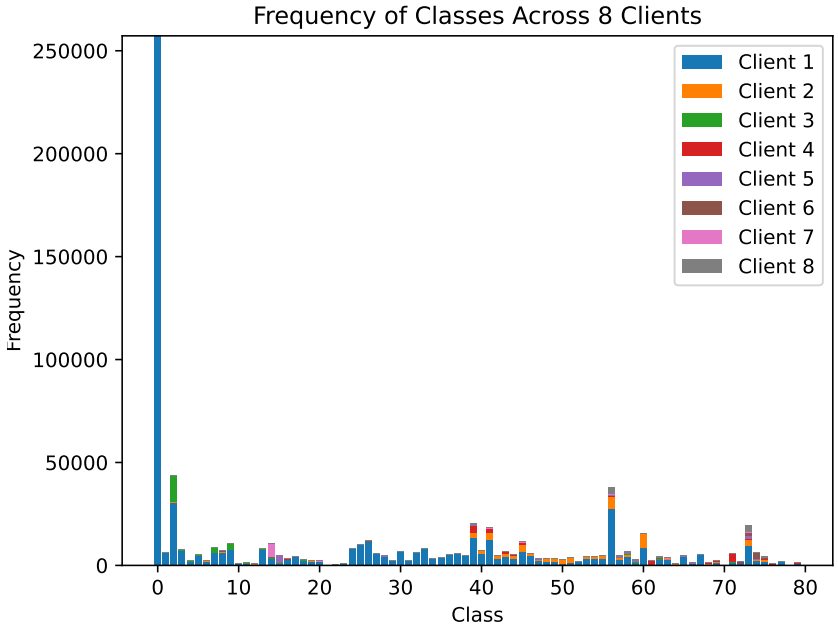


Figure 3.9 – Distribution of classes in 8 clients using FedCV partitioning method.

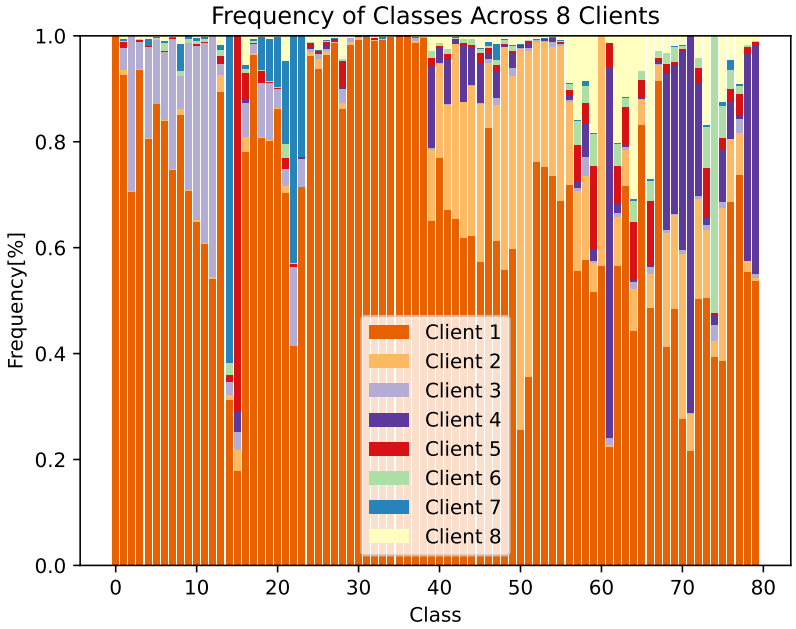


Figure 3.10 – Distribution of classes in percentages in 8 clients using FedCV partitioning method.

This thesis, thus, aims to evaluate how this extremely unbalanced non-i.i.d. data distribution among the clients affects the performances of the system when coupled with different topologies. The findings of this work can be generalized to situations in which a single entity of the system disproportionately concentrates information.

3.4 Approach

The evaluation of the system begins by providing a comparison ground and by reproducing the expected results from the literature. Then, popular and simple topologies are tested with the data distribution in order to understand which factors affect the performances the most, and guidelines for the improvement of performances are derived. Based on these guidelines, a routing algorithm is designed in order to reduce the communication overhead while preserving or improving the performance, and is validated through simulations. Finally, an evaluation of the robustness of the system is provided by testing the impact of failing nodes on the performance.

Five groups of experiments have been defined:

- **Group A:** Baseline experiments.
- **Group B:** Ring topology.

- **Group C:** Line topology.
- **Group D:** Routing experiments.
- **Group E:** Resilience to node failure.

The comparison among the results will be performed using the following metrics:

- **Accuracy:** Measured by mAP@.5, the standard metric for classification and localization accuracy for OD tasks.
- **Learning speed:** Number of communication rounds, in order to compare how quickly the models improve the accuracy compared to one another.
- **Communication overhead:** Measured in number of connections or edges in the topology.

3.4.1 Group A: Baseline experiments

The objective of this group is to provide a comparison ground to the DFL experiments by providing the results for the Centralized ML and FL versions of the system, as well as confirm the correct functioning by reproducing the expected results from the literature. The designed experiments are:

- **Experiment A1:** Centralized ML.
- **Experiment A2:** FL with 4 clients i.i.d..
- **Experiment A3:** FL with 8 clients i.i.d..
- **Experiment A4:** FL with 4 clients non-i.i.d..
- **Experiment A5:** FL with 8 clients non-i.i.d..

3.4.2 Group B: Ring topology

The objective of this group is to provide an initial analysis of the behavior of the DFL system as compared to the Centralized ML and FL versions, and to analyze the effects of the non-i.i.d. distribution for this specific topology.

The ring topology is a popular one in the DFL studies for its simplicity, symmetry and sparse nature, which highlights the effects of DFL in many cases, and is useful to understand the factors affecting the performance with a given data distribution.

The designed experiments are:

- **Experiment B1:** DFL with 4 clients in ring topology i.i.d..
- **Experiment B2:** DFL with 8 clients in ring topology i.i.d..

- **Experiment B3:** DFL with 4 clients in ring topology non-i.i.d..
- **Experiment B4:** DFL with 8 clients in ring topology non-i.i.d..
- **Experiment B5:** DFL with 8 clients in ring topology non-i.i.d. unordered.

3.4.3 Group C: Line topology

The objective of this group is to analyze how a change in the topology of the system affects its performance under the non-i.i.d. data partitioning among the clients.

The line topology is another popular topology for DFL. Since the connection of the nodes does not form a complete cycle, the diffusion of the model of the nodes has only one way to follow. This characteristic gives a higher importance to the order of the nodes in the topology, according to the distribution of their data.

The designed experiments are:

- **Experiment C1:** DFL with 8 clients in line topology non-i.i.d..
- **Experiment C2:** DFL with 8 clients in inverted order line topology non-i.i.d..
- **Experiment C3:** DFL with 8 clients in line topology with bottleneck non-i.i.d..
- **Experiment C4:** DFL with 8 clients in line topology with decreasing bottleneck non-i.i.d..

3.4.4 Group D: Routing experiments

From the results obtained in the preceding groups, the factors affecting performance the most under this data distribution are described and general guidelines to the improvement of the system acting only on the topology are listed. Then, a routing algorithm that allows to select the minimum amount of connections while maintaining a good performance is designed, based on this described guidelines, and validated. Each experiment compares the initial topology with the routed version and with a random selection of connections, to highlight the benefits of the algorithm.

Finally, the proposed extended algorithm, intended to address problematic cases of the basic version, is validated and compared to the former. The designed experiments are:

- **Experiment D1:** DFL with 8 clients routing on a fully-connected topology.
- **Experiment D2:** DFL with 8 clients routing on a meshed topology.
- **Experiment D3:** DFL with 8 clients extended routing on a problematic meshed topology.

3.4.5 Group E: Resilience to node failure

One of the main advantages of the DFL approach is its robustness to node failures. The objective of this group is to analyze the impact of the failure of a node on the performances, under the extremely unbalanced and non-i.i.d. data distribution.

The designed experiments are:

- **Experiment E1:** Failure of dominant node in DFL with 4 clients in ring topology non-i.i.d. at round 50.
- **Experiment E2:** Failure of dominant node in DFL with 4 clients in ring topology non-i.i.d. at round 10.

3.5 Routing Algorithm - Minimum Update Dilution Routing

The reduction of the communication overhead is an important area in FL and DFL research, as it represents an important step for enabling its massive adoption by reducing the demand on limited resources, such as bandwidth and power, especially for cases with a potentially very high number of devices.

One of the possible approaches to reducing the communication overhead for a DFL system is by reducing the amount of connections between nodes. However, removing connections can have an important impact on the performance of the system, as it is essentially changing the topology, which represents a determining factor.

The results of the experiments of Group B and C, which will be discussed in details on the next chapter, allow for the identification of the main factors related to topology affecting the performance of the system under the extremely unbalanced non-i.i.d. data distribution, from which the general guidelines for the improvement of the system were derived, that can be summarized as: Connect as few nodes as possible to the dominant node; Reduce average distances from dominant node; Provide decreasing chain of nodes, in terms of dataset size, beginning from dominant node.

With these insights in mind, this work proposes a routing algorithm for similar situations to reduce the communication overhead by selecting the minimum amount of connections necessary to connect all nodes, while applying the concepts of the guidelines in order to preserve the performance.

The developed algorithm is called Minimum Update Dilution Routing, as it applies the guidelines in order to minimize the dilution of the dominant node's updates as it propagates through the network of nodes. The resulting topology from the application of this algorithm is a Tree, with the minimum amount of nodes connected

to the dominant node, with the shortest branches possible, and with decreasing order beginning from the dominant node, in terms of size of local dataset.

The Algorithm operates in the following way: It begins by selecting the connection of the dominant node to the node with the biggest dataset among the possible ones, and discards all other connections from the dominant node, reducing thus the amount of nodes connected to it (Step 1). Then, all connections of the selected node are chosen, reducing the length of the chains (Step 2). The nodes connected in the previous step are now evaluated in order of largest to smallest dataset. For each node, all of its connections are selected, except for the ones that connect them to nodes of the same level, that is, nodes that have been connected to the previous node at the previous step. The newly connected nodes are nodes of the next level, which eliminate at this step connections to other nodes of the current level that are not the node being currently evaluated (Step 3). When all nodes of the current level have been evaluated, it proceeds to the next level, repeating the step 3 until all nodes of the topology have been evaluated (Step 4), resulting in a Tree topology (Step 5). The application of these steps to a meshed topology are shown in Figure 3.11.

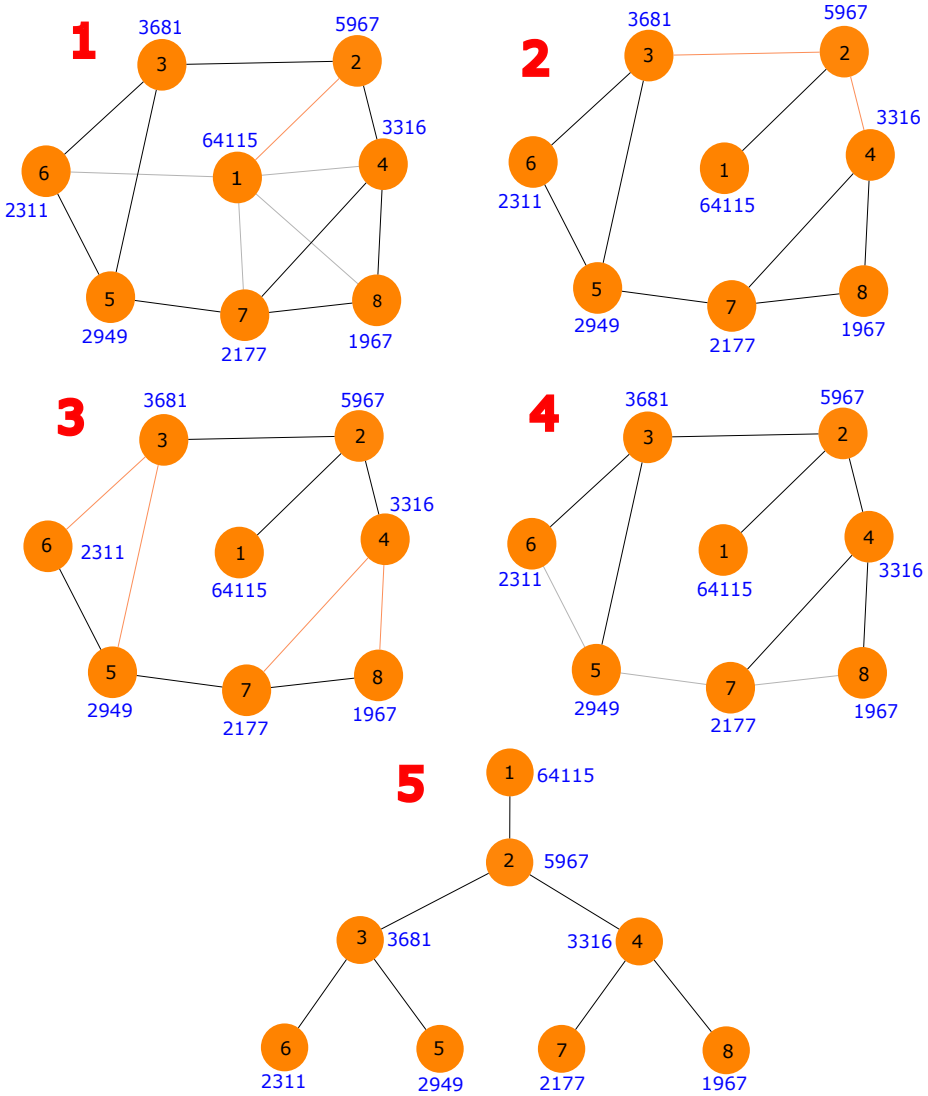


Figure 3.11 – Step-by-step of the application of the Minimum Update Dilution Routing algorithm to a meshed topology. The numbers reported next to each node represents the size of the local dataset in terms of number of images.

The pseudo-algorithm is reported in Algorithm 3.3.

3.5.1 Limitations

This algorithm presents some limitations regarding the types of topology it can be applied to, because it supports only one connection to the dominant node. This may generate problems when the topology requires at least two connections to the dominant node in order to reach all nodes, that is, when the dominant node acts as a bridge between two portions of the topology. Another problem could be the

Minimum Update Dilution Routing pseudo-algorithm

Let N be the vector of nodes ordered by size of local dataset
$N = [n_0, n_1, ...]$
Let E be the connection matrix of the topology, where
$e_{i,j} = e_{j,i} = 1$ if n_i and n_j are connected
Let Q be the vector representing the queue of nodes to be processed by the algorithm

/* The frist step consists in connecting the dominant node, that is n_0, to the node with the most data among the possible ones, and to eliminate the other connections */
for $e_{i,j}$ where $i = 0$ and $e_{i,j} = 1$ **do**
    For smallest j, leave $e_{i,j} = e_{j,i} = 1$, and add n_j to Q
    For any other j, set $e_{i,j} = e_{j,i} = 0$
end for
Add n_0 to Q_{past}

/* The next step consists in analyzing the topology by levels, determined by how many nodes separate a given node from the dominant node. For each level the nodes are analyzed in order of size of dataset, and we select all of its possible current connections to nodes not belonging to the same or previous levels. The nodes connected represent the next level, and are added to Q_{next}. Once a level is fully analyzed, we move to the next level, by assigning Q_{next} to Q*/

while Q not empty **do**
    $n_i \leftarrow$ node at first position in Q

    for j where $e_{i,j} = 1$ **do**
        if n_j in Q **then**
            set $e_{i,j} = e_{j,i} = 0$
        else
            for $e_{j,k}$ where $k \neq i$ and n_k is contained in Q and n_j not contained in Q_{past} **do**
                $e_{j,k} = e_{k,j} = 0$
            end for
            add n_j to Q_{next}
        end if
    end for

    Remove n_i from Q and add to Q_{temp}
    if Q is empty and Q_{next} not empty **then**
        add Q_{temp} to Q_{past}
        empty Q_{temp}
        sort Q_{next} by dataset size
        $Q = Q_{next}$
        Empty Q_{next}
    end if
end while

Algorithm 3.3 – Minimum Update Dilution Routing pseudo-algorithm.

length of the chain of nodes generated by this algorithm, that in some cases could be shortened by simply adding one more connection to the dominant node, resulting thus in a better performance. Considering these limitations, I have developed an extension to this algorithm, called Generalized Minimum Update Dilution Routing, that allows the application of the algorithm to any kind of topology, at the expense of rendering it more complex. First, I'll present the metric used by the choices of the algorithm, and then I'll present the extension itself.

3.5.2 Dilution Metric

The Dilution Metric is defined for each individual node and routing choice, and it represents the weight of the dominant node's updates in that specific node's combination rule for that given routing path. It is calculated as the fraction of the number of samples of the node immediately above it, with respect to the combined number of samples of all nodes connected to it, multiplied by the dilution metric of the node immediately above it. For example, in the topology represented in Figure 3.12, for Client 3 the Dilution metric is calculated as:

$$d_3 = \frac{S_2}{S_2 + S_3 + S_6 + S_5} * d_2 = \frac{5967}{14908} * 0.83 = 0.33 \tag{3.2}$$

Where:

$$d_2 = \frac{S_1}{S_1 + S_2 + S_3 + S_4} = \frac{64115}{77079} = 0.83 \tag{3.3}$$

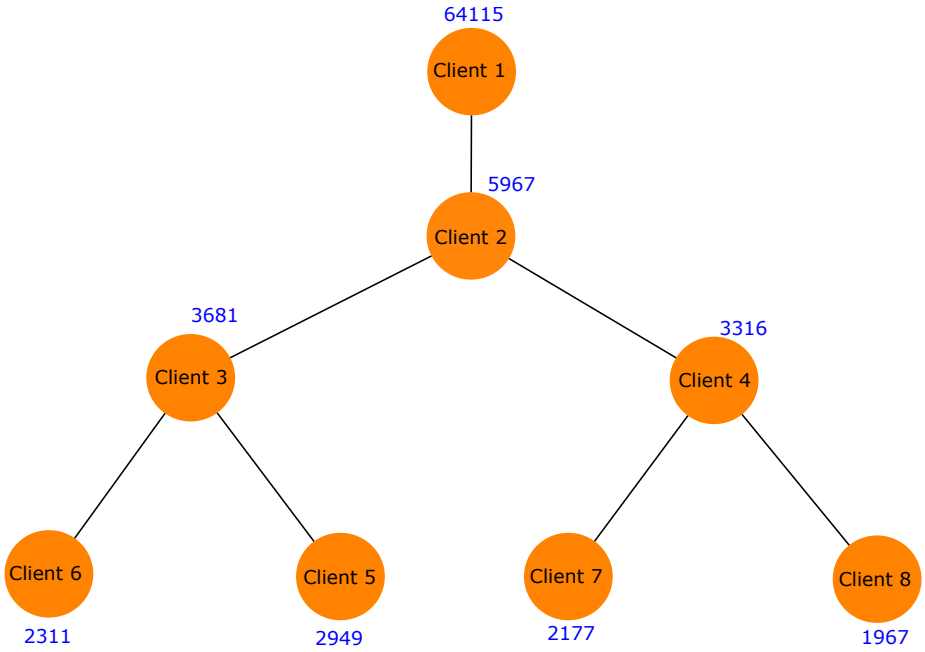


Figure 3.12 – Example topology for the calculation of the dilution metric.

This metric is a number that varies between 0 and 1, and it comprises two important characteristics of the path in one single number: The amount of nodes in the path connecting it to the dominant node, how decreasing/increasing is the path. In the end, it aims to reflect, as the name suggests, the dilution of the dominant node's updates, and allows to make a decision based on a single number about the best path to reach a node in the topology.

In order to detect unconnected nodes, the default value for this metric is set to zero, that will only change when the value for that routing is calculated. If the routing does not reach a given node, that node's metric will be kept to zero, allowing for its detection and subsequent solution by triggering the extension to generate a routing that can reach all nodes.

3.5.3 Generalized Minimum Update Dilution Routing

The idea of the algorithm consists in generating alternative routings departing from the dominant client, if the dilution metric of any client falls below a certain threshold, and selecting for each client the best path leading to it.

The steps of the algorithm will be presented by following an example. Let's consider the topology represented in Figure 3.13.

The algorithm begins by executing the basic version, and calculating the Dilution metric for every client. A threshold for the dilution metric is defined, and if any node falls below that value, the extended algorithm is executed. If every node is above the

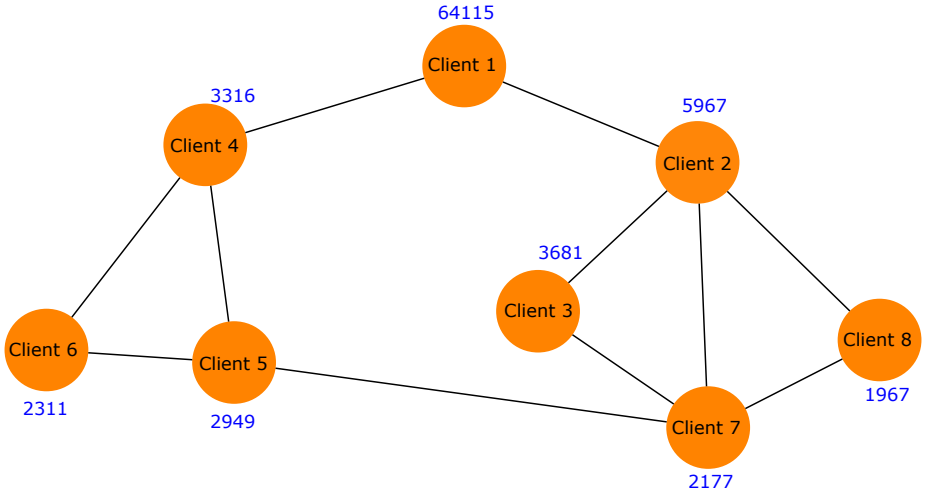


Figure 3.13 – Example topology for Generalized Minimum Update Dilution Routing explanation. The numbers next to the nodes represent the size of the local dataset, in terms of number of images.

threshold, the algorithm ends there. The threshold is necessary for allowing to select for the minimum amount of connections necessary to the dominant node, adding new ones only if a bad performance is predicted for a given node. It also allows to detect unconnected nodes, since they will always have their Dilution metric below the threshold. The threshold is a parameter of the algorithm and its optimal value is dependent on the specifics of the application. The choice should be based on a preliminary study, such as simulations of the expected scenario. For the situation in hand, I have empirically chosen a threshold of t = 0.1 for the Dilution metric, since from the previous experiments, the nodes that fell below that value behaved very badly. By executing the basic algorithm on this topology, the routed topology in Figure 3.14 is obtained, with the dilution metrics in red.

It is possible to see that the clients 4 and 6 fall below the chosen threshold of t = 0.1, due to the long chain of nodes connecting it to the dominant node. Thus, it will trigger the extended algorithm. The extended algorithm consists in calculating an alternative routing departing from a different edge from client 1, if there are any. We select the edge connecting client 1 to the next client with most data among the possible ones, different from the one of the first routing, and apply the basic algorithm beginning from this client. Then we calculate the dilution metrics of this new routing, and compare to the previous one. For each client, we assign the routing that renders the best metric. For example, in our topology we would now route beginning from client 4. The two possible routings and the metrics are shown in Figure 3.15, with red representing the first routing and green the second one. The color of a node represents which routing it has been assigned to.

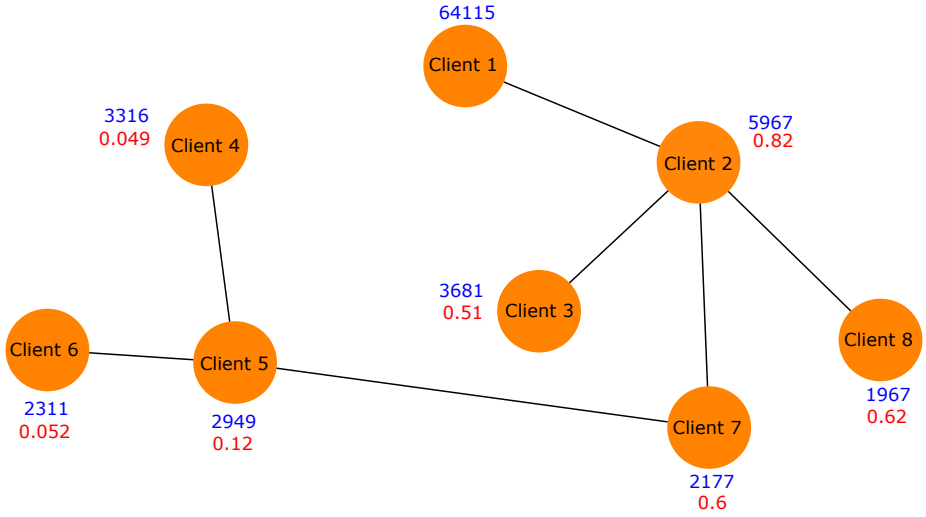


Figure 3.14 – Example topology routed by basic algorithm. For each node, two numbers are reported: The one on top is the size of the local dataset, and the one on the bottom is the dilution metric generated in that routing.

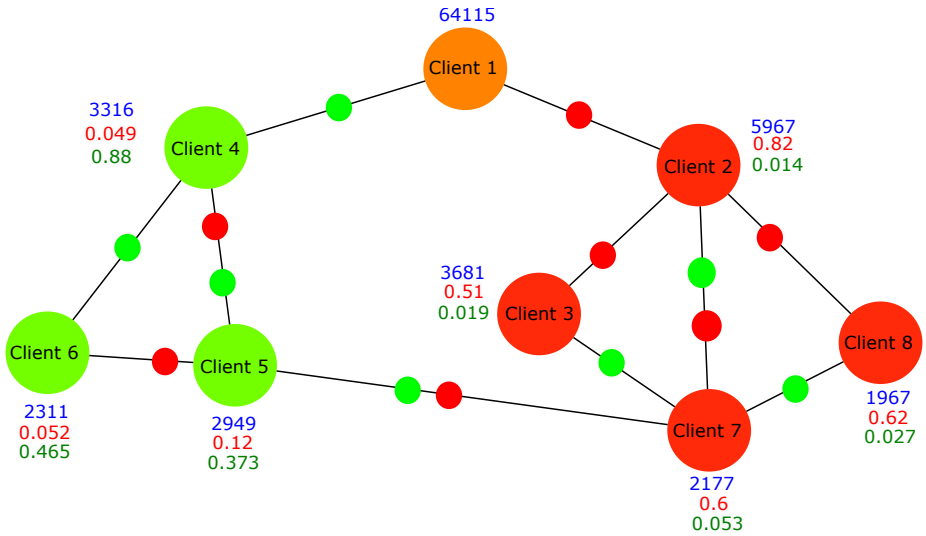


Figure 3.15 – Example topology possible routings according to Generalized Minimum Update Dilution Routing. Next to each node three numbers are reported: On top, the size of the local dataset; In the middle, the dilution metric generated by the first routing, represented in red (dark); At the bottom, the dilution metric generated by the second routing, represented in green (light). The colored circles on the edges indicate whether it has been selected by the routing of that color. The color of the client indicates which routing generated the highest dilution metric.

Now, for each client we perform the edge selection. First we take the routing tree generated by each routing and combine them to generate the global tree. For

each level, we select a node of a tree only if it is painted in that color, that is if that node has been assigned to that routing, as shown in Figure 3.16.

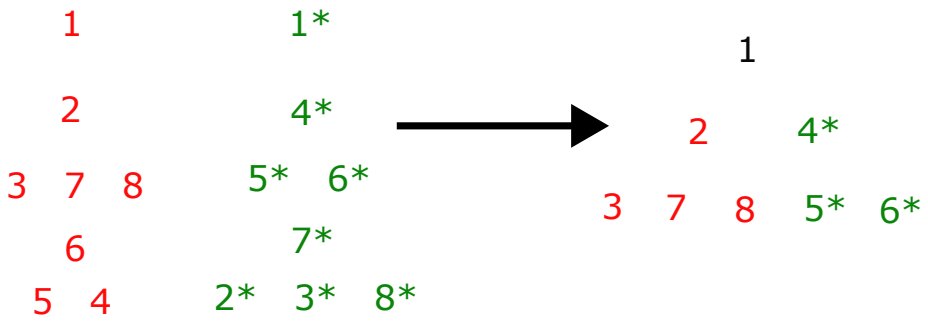


Figure 3.16 – Global routing tree construction.

To select the edges, we evaluate the global tree from top to bottom. For each node, we check the routing tree of its corresponding assigned routing, and select the edge connecting it to the node immediately above it in this tree, to which it possesses a connection.

The final result of the routing is reported in Figure 3.17, where it is possible to see that the clients that previously fell below the threshold are now connected to the dominant node by a much shorter chain of nodes, which will be demonstrated later in the work to render a better performance.

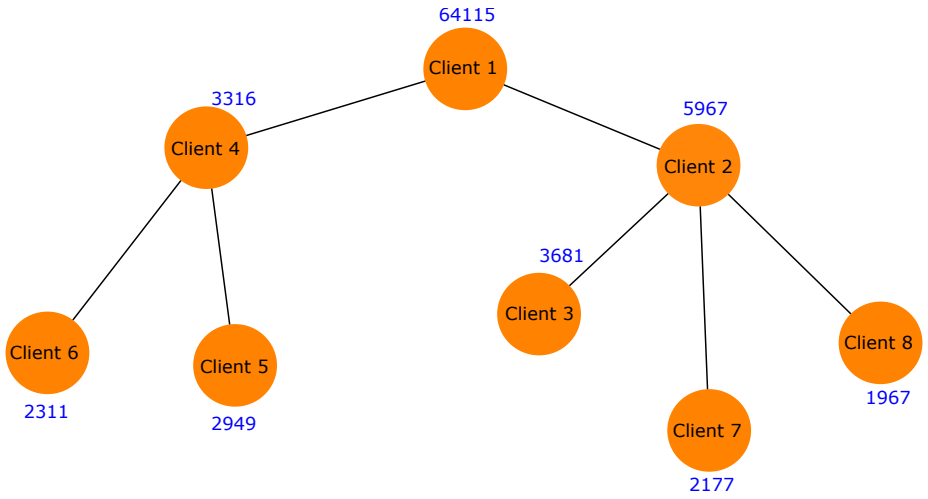


Figure 3.17 – Routed topology resulting from the application of the Generalized Minimum Update Dilution Routing to the example topology.

The pseudo-algorithm describing these steps is reported in Algorithm 3.4 and the continuation in Algorithm 3.5.

Generalized Minimum Update Dilution Routing pseudo-algorithm

Let N be the vector of nodes ordered by size of local dataset
$N = [n_0, n_1, ...]$
Let E be the connection matrix of the topology, where
$e_{i,j} = e_{j,i} = 1$ if n_i and n_j are connected
Let N_{dom} be the vector of n_i belonging to N, for which $e_{i,0} = 1$, that is, they are directly connected to the dominant node
$Num_{routings} = 0$

//We cycle for every node connected to the dominant node, to generate the possible routings
for $n_{initial}$ in N_{dom} **do**
  //MinUpdateDilution is the function that calculates the basic algorithm routing, beginning from the $n_{initial}$ node. It returns the connection matrix of the routing and the routing Tree.
  E_r, T_r = MinUpdateDilution($n_{initial}$,N,E)

  //DilutionMetric is the function that calculates the Dilution metric for all nodes of that given routing r, with default value = 0.
  D_r = DilutionMetric(E_r)

  $Num_{routings}++$

  //If no node is below the threshold, the algorithm can end here, else continue generating alternative routings
  for d in D_r **do**
    if all $d \geq$ threshold **then**
      Break initial for cycle
    end if
  end for
end for

Routing assignments
//Now for each node we decide the best routing based on is dilution metric
for n in N **do**
  Select routing r for which $D_r(n)$ is maximum
  append n to R_r
end for

Global tree construction
//Now we build the global tree beginning from the routing trees
$c = 0$ //Counter representing the level of the tree
while $nodes\_added \neq num\_nodes$ **do**
  for i in num_routings **do**
    for j in $T_i(c)$ **do**
      if j E R_i **then**
        Add j to Global_tree(C)
        nodes_added++
      end if
    end for
  end for
  c++
end while
Continues...

Algorithm 3.4 – Extended Routing pseudo-algorithm

3.6 Methodology

The experiments are performed with single-machine simulations, and due to the OD model complexity and dataset size, they may take a considerably long time, roughly one day for each group of 100 epochs with my available computing resources.

The ideal methodology to follow would be to perform grid searches to find the optimal parameters, and to repeat each experiment multiple times in order to provide statistical validity to the results. Due to the limited computing power and time available, some choices had to be made to minimize the amount of experiments performed.

The experiments of groups A,B and C run for 300 epochs, in order to be comparable to each other and to the benchmark, and have been performed only once, due to its very prolonged time and number of experiments. These experiments are only used to understand the impacts of the topology with general behaviors, and thus are not as critical.

The experiments of group D run for 200 epochs, to reduce the total time, and are run three times each, in order to provide statistical validity to them. These results test the developed algorithm, and thus represent the core of the thesis. For this reason they have been chosen as the ones to be repeated.

Finally, the experiments of group E run for 300 epochs and are repeated three times each, since they are smaller in scale and thus take less time.

Generalized Minimum Update Dilution Routing pseudo-algorithm
Edge selection
//Now we select the edges of each node based on its assigned routing and the global tree
for d in c **do**
  for i in Global_tree(d) **do**
    // First find the assigned routing and corresponding tree
    find r index of R_r to which i belongs
    find i in T_r
    Select the edge that connects it to a node of the previous level in T_r
  end for
end for
Eliminate all non selected edges

Algorithm 3.5 – Extended Routing pseudo-algorithm continuation

Chapter 4

Evaluation

4.1 Group A: Baseline experiments

The objective of this group of experiments is to provide a comparison with the Centralized ML and FL versions for the DFL experiments. It also aims to verify the correct functioning of the system by reproducing expected results from the scientific literature.

4.1.1 Experiment A1: Centralized ML Benchmark

The first step for an evaluation of the system is to understand how it would behave in a Centralized ML setting, that is, a single node concentrating all the dataset, and training a model on it. The result will serve as a benchmark to the other approaches, as well as a confirmation of the correct functioning of the system created.

This experiment was performed training the YOLOv5n model over the entire MS COCO dataset for 300 epochs using the parameters contained in the file whose values are reported in Appendix A, the `hyp.scratch-low.yaml`. This is the same setup declared in the YOLOv5 repository for the pretrained weights checkpoint, whose performance will be used as an initial reference to the system.

In order to reduce the usage of GPU memory for the experiments, the image size has been reduced from the default [640,640] to [480,480]. The table summarizing the used parameters are reported in Table 4.1.

Image size	[480, 480]
Learning Rate	0.001
Batch size	32

Table 4.1 – Parameters used for Centralized ML.

The result of the experiment is reported in Figure 4.1. The final performance achieved is mAP@.5 = 0.4436. The reported performance in the YOLOv5 repository for the YOLOv5n model with this same setup is mAP@.5 = 0.4570. Considering that the used image size is smaller, and that this reduction during the tests lead to a decrease in the mAP@.5 of approximately 0.015, it is safe to conclude that the system behaves as expected and achieves the declared performance.

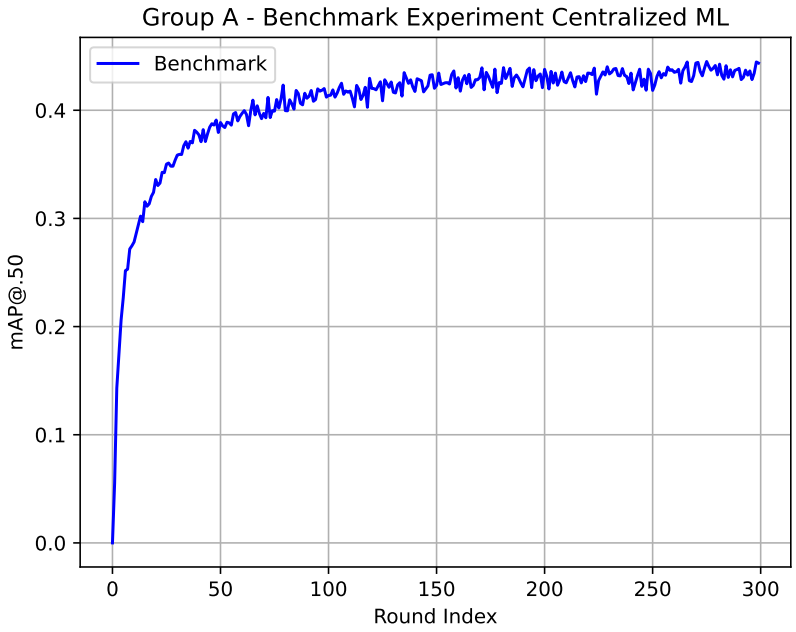


Figure 4.1 – Result of Centralized ML run on the COCO dataset for 300 epochs.

4.1.2 Experiment A2: FL with 4 clients i.i.d.

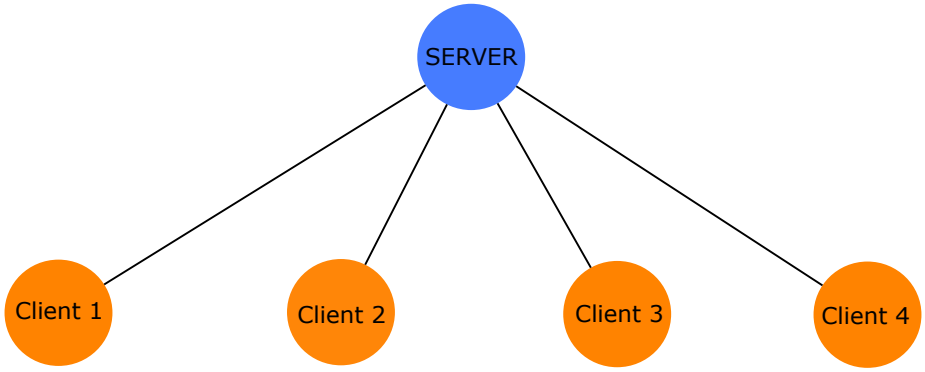


Figure 4.2 – Scheme of FL with 4 clients.

In this experiment a FL version of the system is tested, with an i.i.d. partition of the dataset among the clients, as reported in Figure 4.2. The parameters used for each of the models are the same ones as for the Centralized ML benchmark, except for the batch size, which has been reduced to 8 in order to maintain the global batch size for a fair comparison. Furthermore, an extra parameter needs to be defined, which is the number of local epochs before aggregating the global model, which is set to 1. A summary is reported in Table 4.2.

Image size	[480, 480]
Learning Rate	0.001
Batch size	8
Local epochs	1

Table 4.2 – Parameters used for FL with 4 clients.

The results of the experiment are represented in Figure 4.3, in which it is possible to observe that the curve of FL with four clients for i.i.d. data distribution achieves approximately the same final accuracy as the Centralized ML benchmark. The difference is that it converges a bit slower than the benchmark, because the gradients of the clients may point to slightly different directions in each round, rendering it harder to converge as fast as the centralized version.

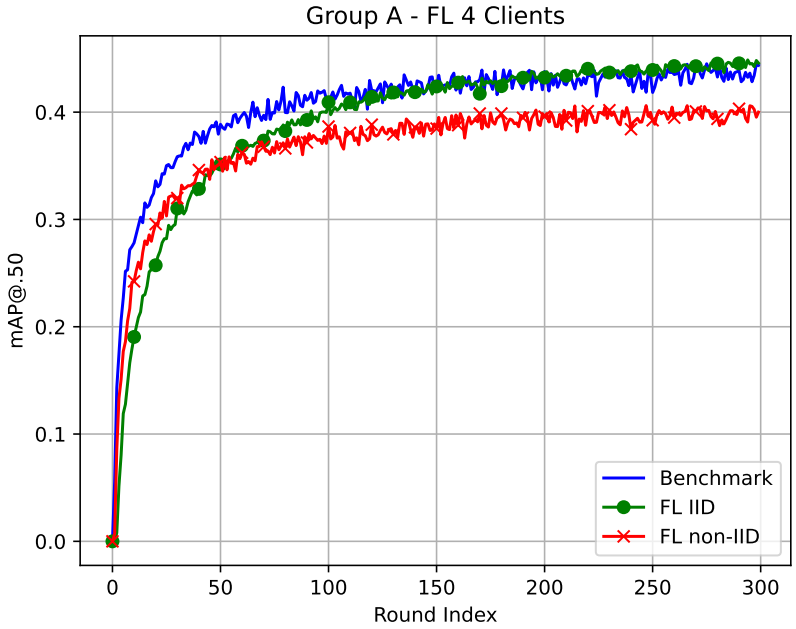


Figure 4.3 – Result of FL with 4 clients for i.i.d. and non-i.i.d. data distribution among the clients.

4.1.3 Experiment A3: FL with 8 clients i.i.d.

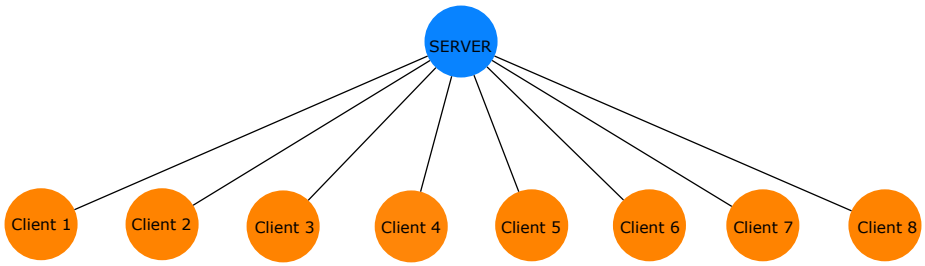


Figure 4.4 – Scheme of FL with 8 clients.

In this experiment, a larger version of the system with 8 clients is tested, as showed in Figure 4.4. The parameters are the same as in the four clients case, with the exception of the batch size, which is reduced to 4 in order to maintain the same global batch size. The summary of the parameters is reported in Table 4.3.

The results are reported in Figure 4.5. An increase in the number of clients leads to each client having a smaller local dataset, resulting in more "noisy" model updates. This, coupled with the fact that there are more updates being fed to the combination rule and pushing towards different directions, leads to a slower convergence. In fact, the final accuracy obtained in the 300 epochs is lower than in the 4 clients case.

Image size	[480, 480]
Learning Rate	0.001
Batch size	4
Local epochs	1

Table 4.3 – Parameters used for FL with 8 clients

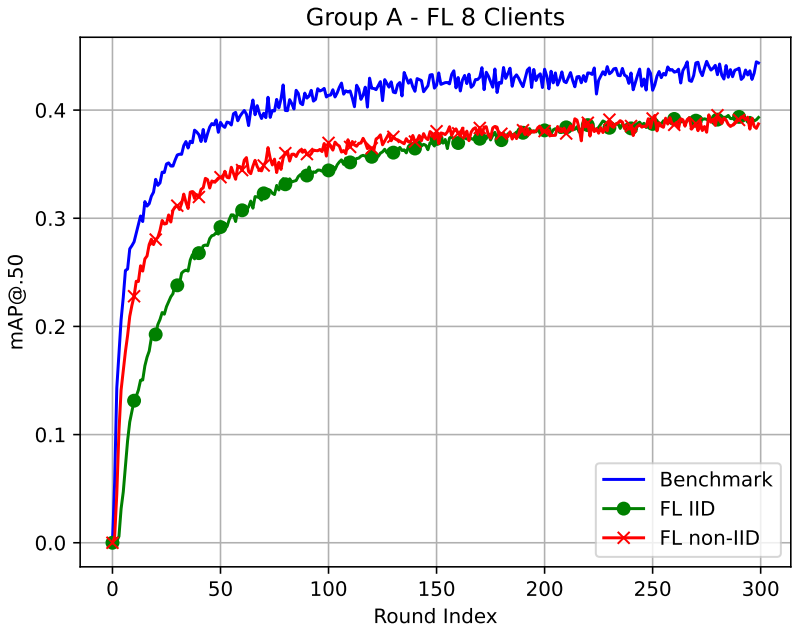


Figure 4.5 – Result of FL with 8 clients for i.i.d. and non-i.i.d. data distribution among the clients.

4.1.4 Experiment A4: FL with 4 clients non-i.i.d.

The results of this experiment are reported in Figure 4.3 and the scheme in Figure 4.6. It is possible to observe that the non-i.i.d. curve starts converging faster, but it slows down quickly, leading to a lower final accuracy by the end of the 300 epochs. This behavior can be explained by the fact that one single node concentrates most of the data, and it weights much more than the other ones in the combination rule at the server, thus giving a more defined direction with its own gradient, leading to a faster convergence in the beginning. But a certain point is reached where the dominant node has exploited most of the useful information contained in its dataset, and since the update of the other nodes count very little in the combination rule, the information contained in their dataset, necessary for the improvement of the accuracy, is exploited very slowly.

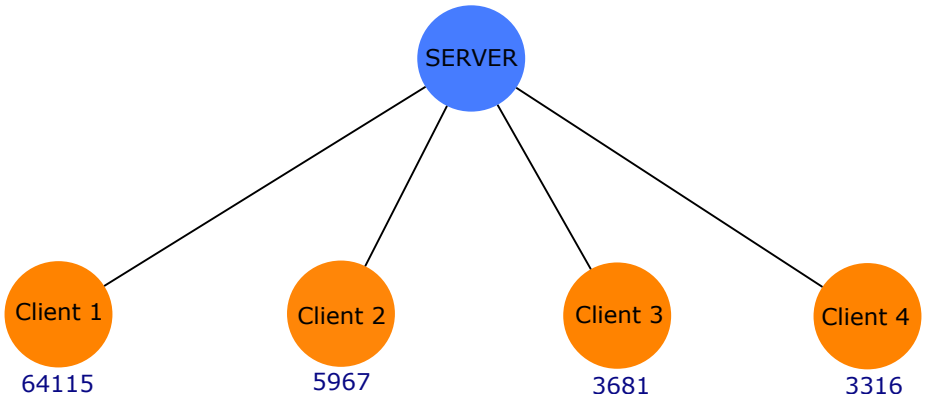


Figure 4.6 – Scheme of FL with 4 clients for non-i.i.d. distribution. The number under each node represents the number of images of its local dataset.

4.1.5 Experiment A5: FL with 8 clients non-i.i.d.

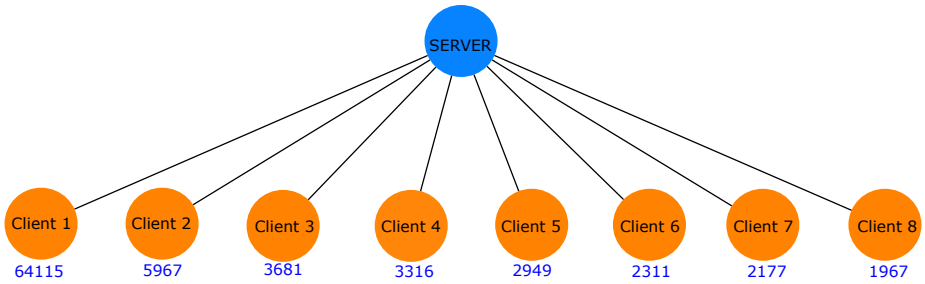


Figure 4.7 – Scheme of FL with 8 clients for non-i.i.d. distribution. The number of images is reported under each node in blue.

The results are reported in Figure 4.5 and the scheme in Figure 4.7. Similarly to the four clients case, the non-i.i.d. curve converged faster in the beginning, but then slowed down to be overtaken by the i.i.d. curve in accuracy. In this case, the i.i.d. curve converges much slower too, and so by the end of the 300 epochs, the two curves have similar accuracies.

4.2 Group B: Ring topology

This is the first group of experiments dealing with the DFL system. The objective is to observe how the topology of the system changes the performance for different data distributions. The Ring topology has been chosen as the first one because it is a popular one in the studies, and it tends to highlight the effects of sparse connections.

4.2.1 Experiment B1: DFL with 4 clients in ring topology i.i.d.

The parameters of this experiment, as for every other experiment with four clients, are the same reported in Table 4.2. The topology scheme is reported in Figure 4.8 and the results in Figure 4.9.

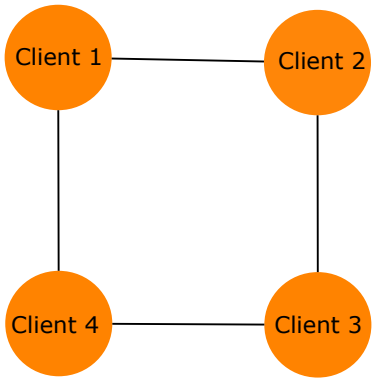


Figure 4.8 – Scheme of DFL with 4 clients for i.i.d. distribution.

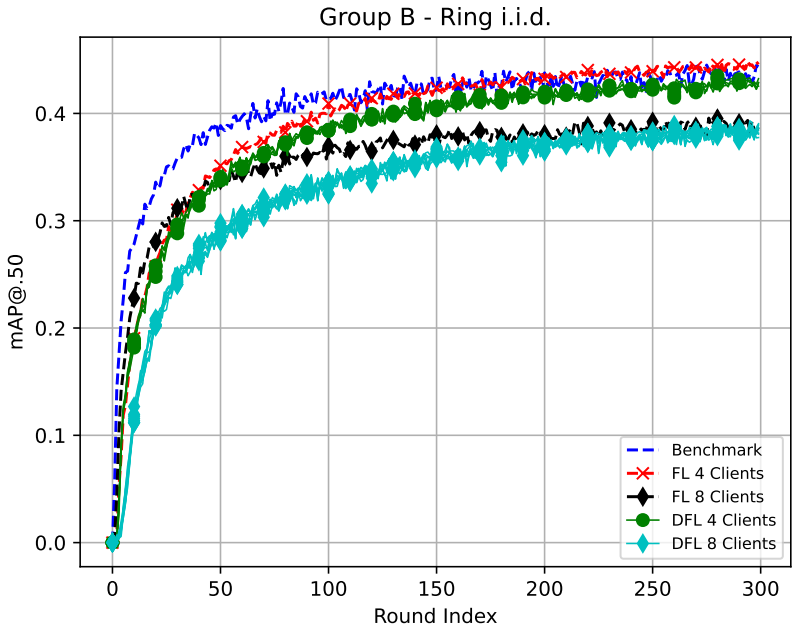


Figure 4.9 – Results of training for 300 epochs of DFL system for i.i.d. data distribution with 4 and 8 clients, compared to the FL and Centralized ML versions.

As expected from the related work, with i.i.d. data distribution the ring topology behaves similarly to the FL version. It is possible to observe that it has slightly more

difficulty converging, due to the fact that the updates need to propagate through the network of nodes.

4.2.2 Experiment B2: DFL with 8 clients in ring topology i.i.d.

The parameters of this experiment, as for every other experiment with eight clients, are the same reported in Table 4.3. The topology scheme is reported in Figure 4.10 and the results in Figure 4.9.

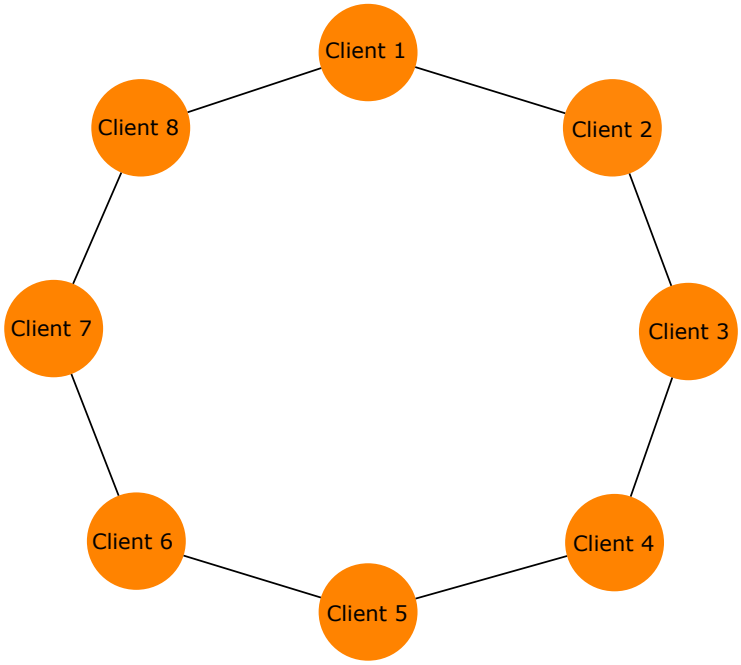


Figure 4.10 – Scheme of DFL with 8 clients for i.i.d. distribution.

In this case again, we have that the ring topology behaved similarly to the FL version. We can observe that the difference between them increases for a higher number of clients. This is because the topology becomes more sparse, thus increasing the impact of the update propagation.

4.2.3 Experiment B3: DFL with 4 clients in ring topology non-i.i.d.

In this experiment we introduce the non-i.i.d. distribution of the dataset among the clients, in order to observe how a topology may interact with it. The scheme of this experiment is reported in Figure 4.11 and the results in Figure 4.12.

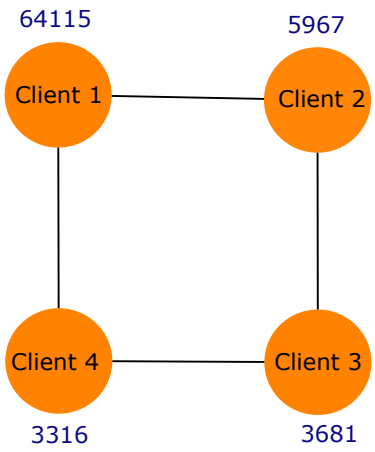


Figure 4.11 – Scheme of DFL with 4 clients for non-i.i.d. distribution. The numbers in blue next to the nodes represents the number of images in the local dataset.

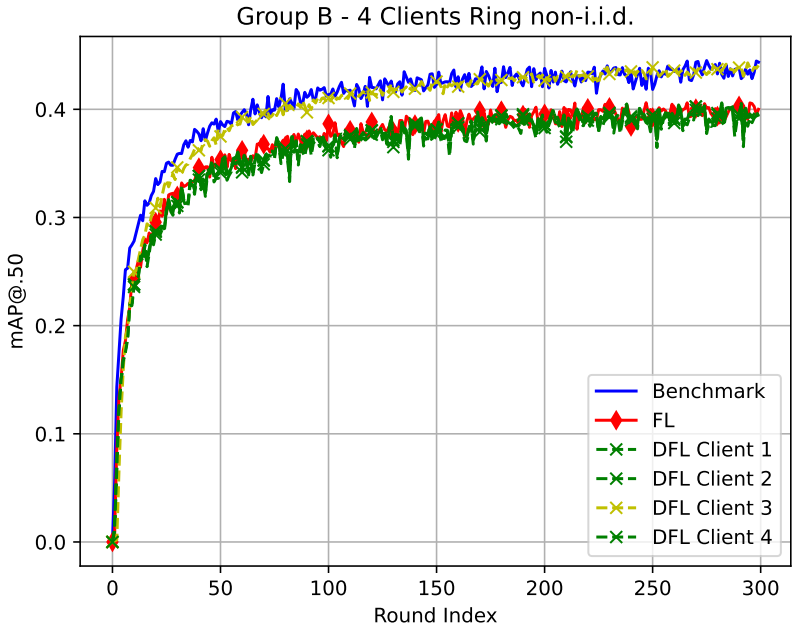


Figure 4.12 – Results of training for 300 epochs of DFL system for non-i.i.d. data distribution with 4 clients for Ring topology, compared to FL and Centralized ML versions.

From the results, it is possible to observe a behavior of the nodes divided in two groups: nodes directly connected to Client 1, the dominant node; nodes not directly connected to the dominant node.

The first group is formed by three nodes, which behave almost identically. This is due to the fact that Client 1 contains a much larger dataset in terms of number of images, and thus the weight of its updates in the combination rule of every client connected to it is much larger than the other updates. Consequently, the model of all of these nodes is determined almost exclusively by Client 1's updates, calculated on its dataset, leading them to behave very similarly. The curve of this group is very similar to the curve of the FL version, since the combination rules are very similar. In fact, to obtain the exact same combination rule as the FL on all nodes, it would be sufficient to add two extra connections, rendering the topology fully connected.

The second group is formed by only one client in this case, Client 3, and it achieves a higher accuracy than its peers in the analyzed time frame. This client is connected only to clients with a comparable local dataset size, leading to a much more balanced combination rule. This allows the client to take into account the updates coming from other clients, with different datasets, which contain new information since the distribution is non-i.i.d.. For these reasons, the client is able to converge to a higher accuracy. The other group instead cannot exploit the information from other datasets, because it is almost completely determined by the updates coming from the dominant one, which eventually reaches the limit of the dominant dataset. It is worth noticing that the curve of the second group performs better than the FL version, and approaches the performances of the Centralized ML version.

4.2.4 Experiment B4: DFL with 8 clients in ring topology non-i.i.d.

Now we observe the effects of an increase in the number of clients under the same conditions. The scheme is reported in Figure 4.13 and the results in Figure 4.14.

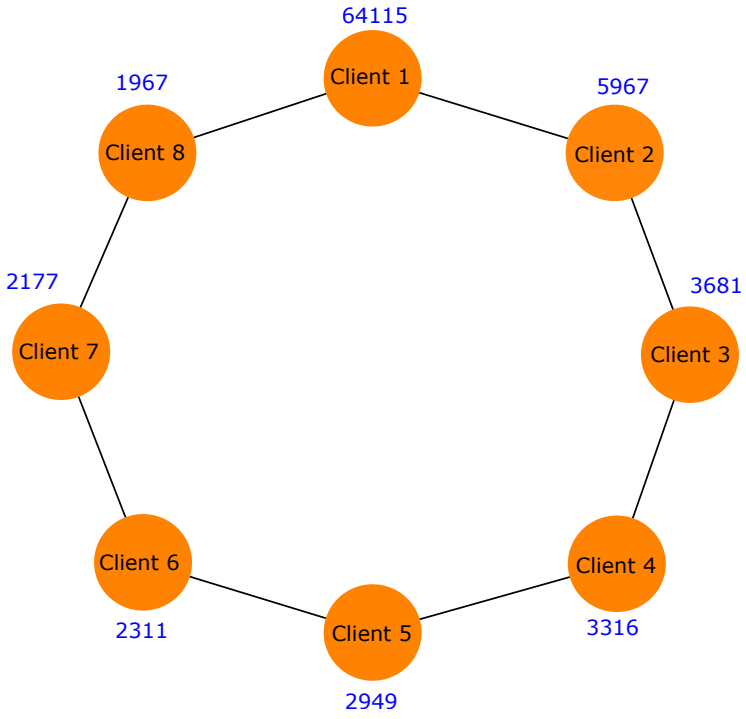


Figure 4.13 – Scheme of DFL with 8 clients for non-i.i.d. distribution. The numbers in blue below the nodes represents the number of images in the local dataset.

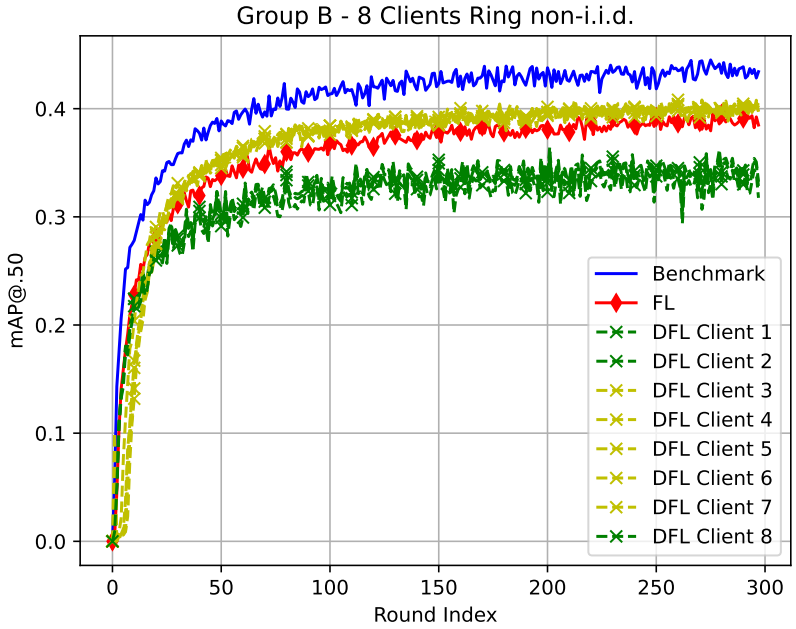


Figure 4.14 – Results of training for 300 epochs of DFL system for non-i.i.d. data distribution with 8 clients for Ring topology, compared to FL and Centralized ML versions.

Here again the behavior is divided in two groups, for the same reasons stated in the previous experiment. The difference is that an increase in the number of clients leads to a slower improvement and a lower final accuracy for both groups. It is also possible to observe that an increase in the number of clients lead to a bigger drop in accuracy for the DFL Ring system than for the FL system. This is due to the fact that the ring topology becomes more sparse, and the updates of clients with the most information takes longer to propagate, thus highlighting the effects of the topology on increasing the difficulty of convergence.

4.2.5 Experiment B5: DFL with 8 clients in ring topology non-i.i.d. unordered nodes

The aim of this experiment is to understand how big of an impact a change in the order of nodes has on the performances of the system for the Ring topology. The scheme is reported in Figure 4.15 and the results in Figure 4.16.

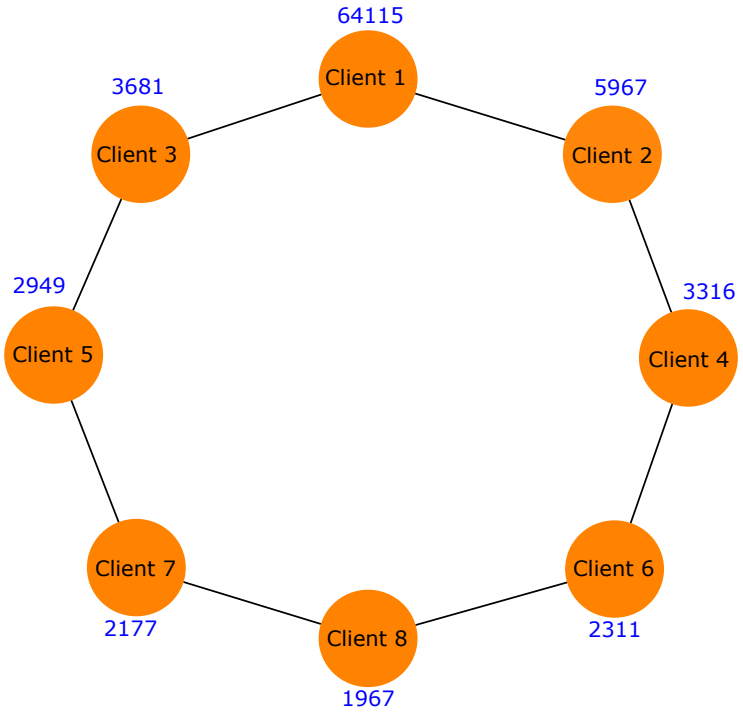


Figure 4.15 – Scheme of DFL with 8 clients for non-i.i.d. distribution. The numbers in blue below the nodes represents the number of images in the local dataset. The clients are in a different order than in the other experiments.

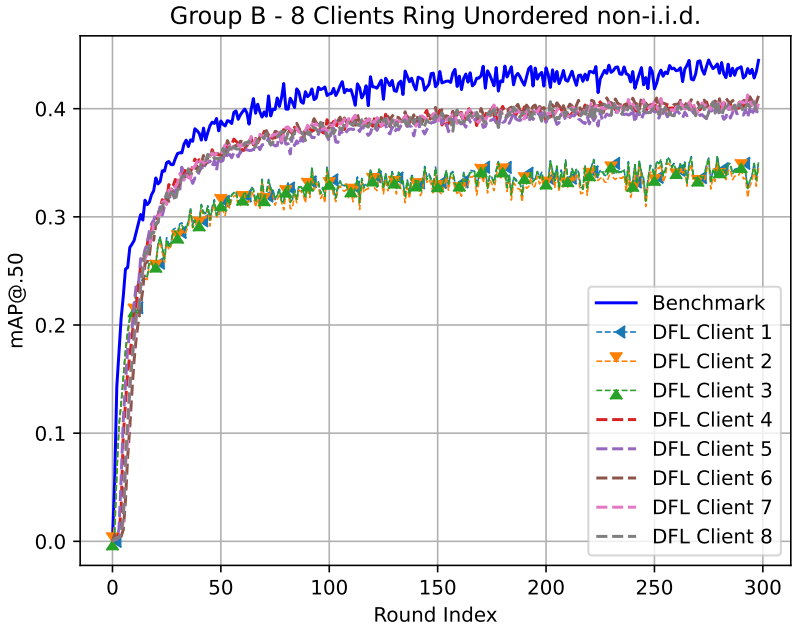


Figure 4.16 – Results of training for 300 epochs of DFL system for non-i.i.d. data distribution with 8 clients for Ring topology with a different order of nodes, compared to the Centralized ML version.

A change in the order of nodes did not significantly affect the performances in this case.

4.2.6 Conclusions Group B

With i.i.d. data, the impacts of the topology were very subtle. They made themselves more apparent with non-i.i.d. data distribution, in which the behavior of the clients was divided in two different groups, defined by the presence of a direct connection with the dominant node. The nodes not directly connected to the dominant client could perform even better than the FL version. Thus, a direct connection with the dominant node in the topology might be a limiting factor for a node's performance.

It was also possible to observe that the topology scale tends to affect the learning speed, as it renders slower the process of knowledge diffusion in the network.

4.3 Group C: Line topology

In this group, the performances of the system are analyzed under a different structure, the Line topology. The aim is to observe how a different topology may change the system behavior, as well as to analyze the particularities of this topology. Differ-

ently from the Ring, the Line has only one way for the propagation of the model updates, and this characteristic may highlight some of the effects of topology on the performance.

4.3.1 Experiment C1: DFL with 8 clients in Line topology non-i.i.d.

As a first experiment, we take the ring topology of the previous experiments and remove the connection between the first and last client, thus creating a chain of decreasing nodes by size of dataset. The scheme is reported in Figure 4.17 and the results in Figure 4.18.

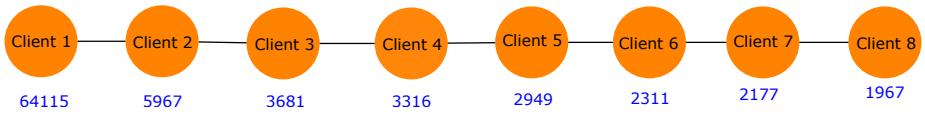


Figure 4.17 – Scheme of DFL with 8 clients for Line topology with non-i.i.d. distribution. The numbers in blue below the nodes represents the number of images in the local dataset.

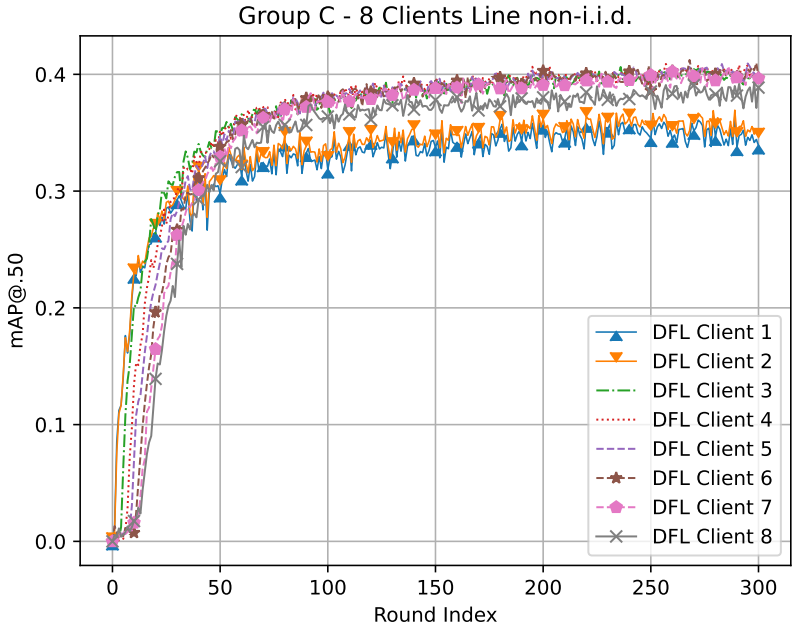


Figure 4.18 – Results of training for 300 epochs of DFL system for non-i.i.d. data distribution with 8 clients for Line topology

From the results it is possible to observe some differences with respect to the Ring topology. First, by disconnecting the Client 8 from the dominant node, it joins

the second group of nodes and obtains the higher final accuracy. Second, the effect of the length of the chain of nodes is apparent in the learning speed of the nodes: The closer a node is to the dominant node, the quicker it reaches the higher accuracy. This is due to the fact that the updates of Client 1 are critical for the improvement in accuracies, as it concentrates most of the data. The distance of a node influences not only the round at which it will receive the for the first time information coming from the dominant client due to the propagation, but also the level of dilution that it will have suffered, since it has to go through many combination rules until it reaches the furthest nodes.

4.3.2 Experiment C2: DFL with 8 clients in Inverted Line topology non-i.i.d.

This experiment aims to evaluate the impact of a change in the order of nodes for this topology. The Inverted Line topology scheme is represented in Figure 4.19 and the results in Figure 4.20.

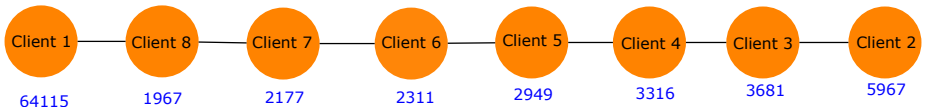


Figure 4.19 – Scheme of DFL with 8 clients for Inverted Line topology with non-i.i.d. distribution. The numbers in blue below the nodes represents the number of images in the local dataset.

From the results, it is possible to observe a big difference in performance deriving solely from the order of the nodes. The learning speed of most nodes has been severely reduced.

Client 1, the dominant client, depends almost exclusevely on its own dataset, and thus it behaves as in the other experiments. Client 8 is directly connected to it, and thus it behaves identically, since it gets dominated by the former's updates in the combination rule. Client 7 receives the updates of Client 1 through Client 8, which reduces its weight in the combination rule. Thus, Client 7 begins improving its accuracy after some rounds, when the updates received are enough to guide it to the right direction. In the end of the 300 epochs, it achieves higher performance than the previous two nodes, because its more equal combination rule allows it to exploit the non-i.i.d. information of the other datasets in the topology.

Clients 2,3 and 4 are the next ones with the best performance, even though they are located at the opposite edge from the dominant node. They manage to obtain these performance because they have big enough datasets to start improving on their own, without waiting for Client 1's updates. They also manage to achieve the higher accuracy by exploiting the datasets of other nodes in the topology.

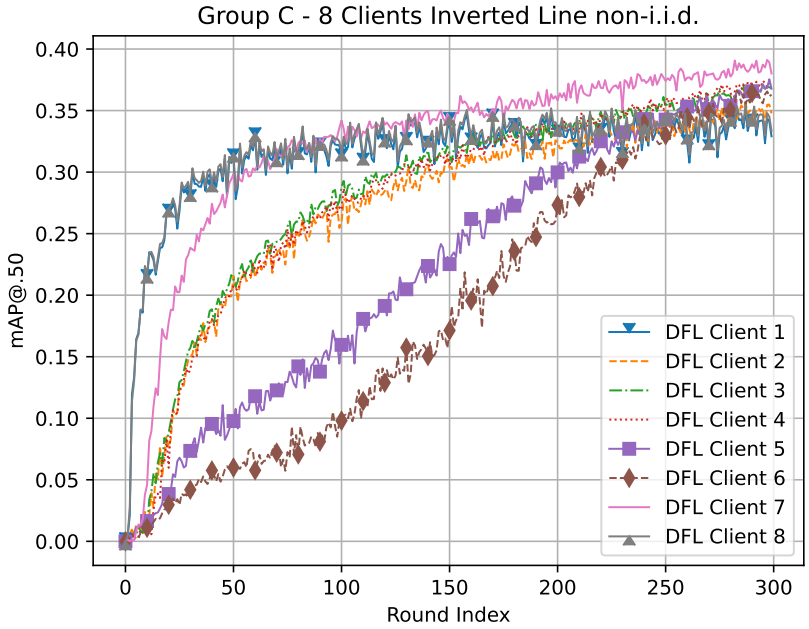


Figure 4.20 – Results of training for 300 epochs of DFL system for non-i.i.d. data distribution with 8 clients for Inverted Line topology.

Finally there are the nodes at the center of the chain, 5 and 6, which obtained the worst performance of all. This behavior is mainly due to two reasons: Their datasets are too small for them to improve on their own, and thus they are dependent on updates coming from other nodes; They are separated from the two nodes with the biggest datasets, 1 and 2, by two other nodes, and thus the updates that get to them are "diluted" by the other nodes' combination rules. It is also worth noticing that they have a gap between their curves, with node 5 performing better than node 6, even though the latter is closer to the dominant node of the topology. This can be attributed, among other possible reasons, to the order of the nodes connecting them to the closest main source of information: Client 6 is connected to Client 1 by an increasing chain in terms of numbers of samples (1-8-7-6), which leads to a higher dilution of the updates; Client 5 instead is connected to Client 2 by a decreasing chain of nodes (2-3-4-5), which mitigates the effects of the dilution of the node's updates. The effects of the dilution of the dominant node's updates are shown in Figure 4.21, measured using the Dilution metric defined previously, where is possible to observe that an increasing order leads to much lower values along the chain.

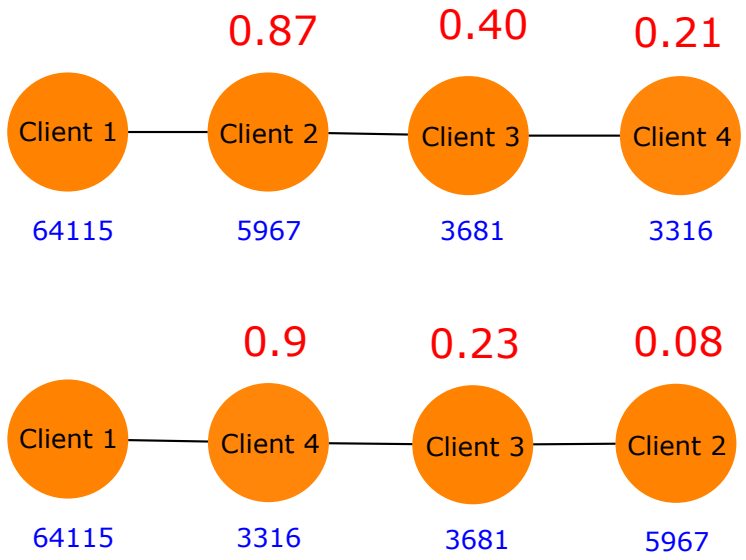


Figure 4.21 – Example of how the order of the nodes affects the dilution of the dominant node updates, measured using the Dilution Metric, written in red above the respective nodes. Below each node the size of the local dataset is reported.

4.3.3 Experiment C3: DFL with 8 clients in Bottleneck Line topology non-i.i.d.

For this topology, the dominant node has been placed in the middle, reducing the average distance from it to every other node in the topology. The scheme is reported in Figure 4.22 and the results in Figure 4.23.

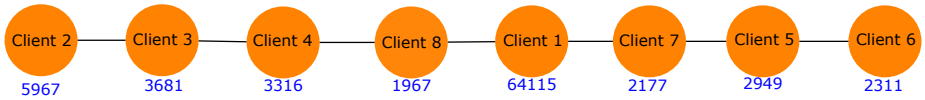


Figure 4.22 – Scheme of DFL with 8 clients for Bottleneck Line topology with non-i.i.d. distribution. The numbers in blue below the nodes represents the number of images in the local dataset.

From the results it is possible to observe that the reduction of the average distance between the dominant node and every other node accelerated the improvement. It is also possible to observe that the two branches coming out of Client 1 performed differently: The branch to the right started improving before the branch on the left. This can be attributed to the difference in the number of samples of the nodes in the chain, with the branch on the left having a bigger difference between each node and being increasing, which amplifies the effects of the dilution of the dominant node's updates.

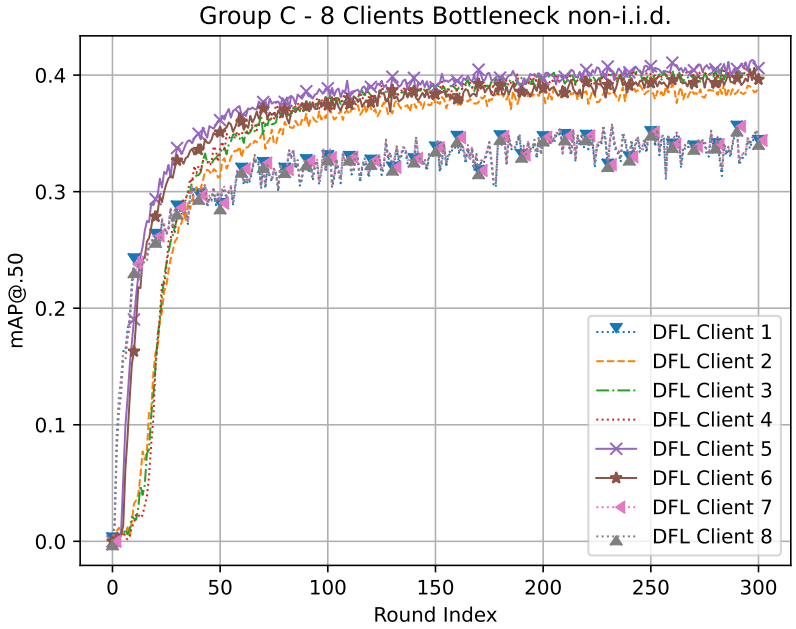


Figure 4.23 – Results of training for 300 epochs of DFL system for non-i.i.d. data distribution with 8 clients for Bottleneck Line topology.

4.3.4 Experiment C4: DFL with 8 clients in Bottleneck Decreasing Line topology non-i.i.d.

The idea behind this experiment is to apply both the reduction of the average distance to the dominant node and decreasing chains, and to verify if an improvement in the performances actually follows. This topology, as shown in Figure 4.24, places Client 1 in the middle as the Bottleneck one, but provides decreasing chain of nodes for both branches. The results are reported in Figure 4.25.

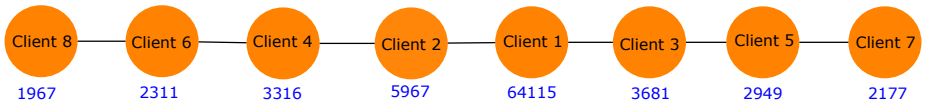


Figure 4.24 – Scheme of DFL with 8 clients for Bottleneck Decreasing Line topology with non-i.i.d. distribution. The numbers in blue below the nodes represents the number of images in the local dataset.

Indeed, by applying both of the concepts derived from the previous experiments, the best performance has been obtained, in terms of speed of learning of the nodes.

To draw a comparison among the different topologies, we take as reference the dominant node's curve, together with the nodes directly connected to it, since they are almost independent from the topology configuration. Then, the round index of

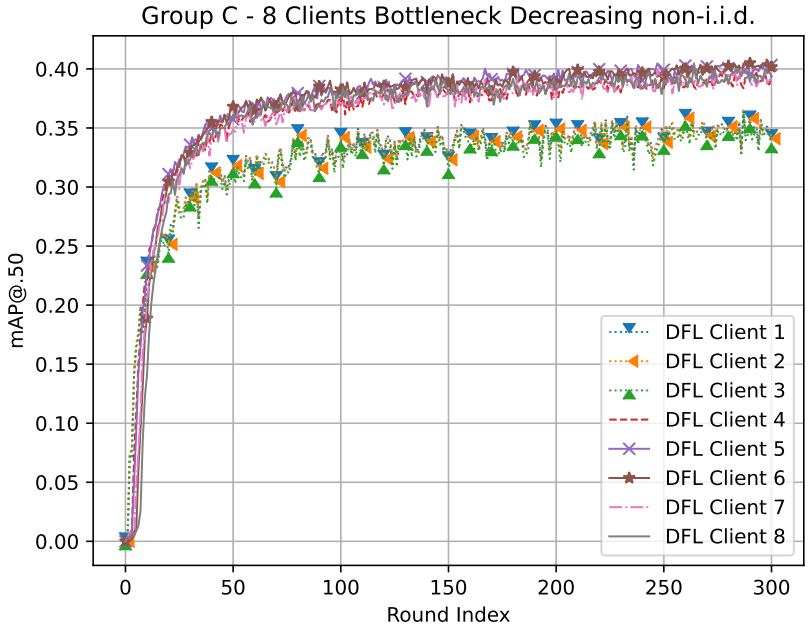


Figure 4.25 – Results of training for 300 epochs of DFL system for non-i.i.d. data distribution with 8 clients for Bottleneck Decreasing Line topology.

the first and last node curves to cross this reference are used to compare the learning speed in the different topologies. The table comparing these indexes is reported in Table 4.4.

Topology	Round first crossing	% from best	Round last crossing	% from best
Bottleneck Decreasing	8	-	15	-
Bottleneck	13	162,5%	33	220%
Line	15	185,5%	37	246,6%
Line Inverted	47	587,5%	239	1593,3%

Table 4.4 – Table comparing the learning speed of the different line topologies, by taking the round index of the first and last crossings of the dominant node's curve.

From the results it is possible to observe a difference in the last crossing between the best and worst topology of 15 times. This leads us to conclude that the order of the nodes is a critical choice in the topology, and that it is possible to improve performance by a big amount by simply reducing average distances from the dominant node and by providing decreasing chain of nodes starting from it.

4.3.5 Conclusions Group C

From the experiments so far, it is possible to identify the factors in a given topology affecting a node's performance the most, under the extremely unbalanced non-i.i.d. data distribution:

- Presence of direct connections to dominant node.
- Number of nodes between a given node and the dominant one.
- Order of the nodes in the chain connecting a given node to the dominant one, in terms of increasing or decreasing number of samples.
- Size and representativeness of a node's own dataset.

These factors boil down to the unbalance of the combination rule, the dependency of a node on other nodes' updates, and how diluted the dominant node's updates get when they reach these dependent nodes.

From these factors, it is possible to derive general guidelines on how to maximize the performances of a system acting only on the topology:

- Connect as few nodes as possible to the dominant node, as it will restrict their performance in terms of final accuracy.
- Reduce the average distance between each node of the topology and the dominant node.
- For each chain of nodes, try to make it as decreasing as possible, in order to limit the dilution of the dominant node updates as it propagates.

4.4 Group D: Routing

In order to reduce the communication overhead of a DFL system, it is possible to eliminate connections between the nodes, which can have a big impact on performance. As shown in the previous experiments, the choice of how to organize the topology is critical for the performance of the system, and the guidelines for improving them have been defined. The proposed routing algorithm selects the minimum amount of connections necessary to reach all nodes in the topology, thus reducing to the minimum the communication overhead, while applying the guidelines for increased performance, ensuring that the system behaves as similarly as possible, or even better, than the version with no routing. The following experiments aim to validate the usefulness of this algorithm in different scenarios. The graphs of the repetitions of the experiments with the 95% confidence interval is reported in Appendix B.

The algorithm is described in section 3.5.

4.4.1 Experiment D1: DFL with 8 clients routing on a fully-connected topology

First, the algorithm is tested with a fully-connected topology, which means that it is possible to select any connection between two nodes of the topology. Thus, it is possible to obtain the theoretical best possible topology, in terms of performance, with the designed algorithm.

The fully-connected topology scheme is represented in Figure 4.26. It is possible to point out that the fully connected topology obtains exactly the same results as the FL version, since in every node in this case there will be the same combination rule as there was in the server for the FL.

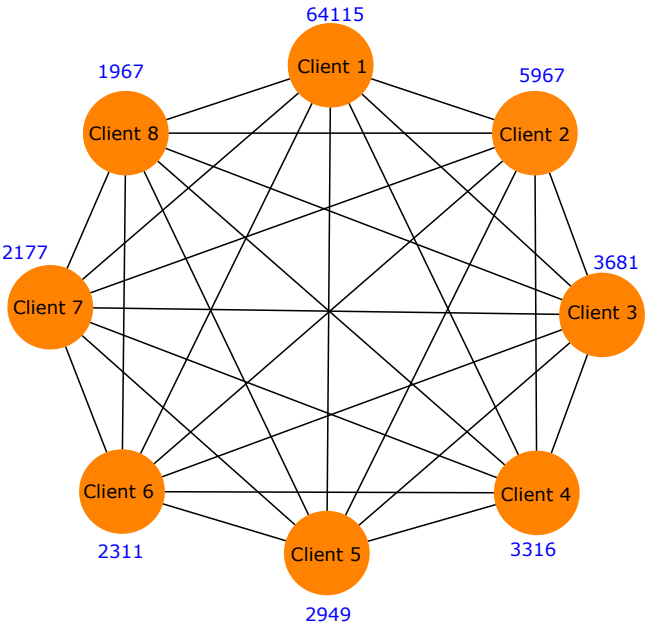


Figure 4.26 – Scheme of DFL with 8 clients for a fully-connected topology with non-i.i.d. distribution. The numbers in blue next to each node represents the number of images in the local dataset.

Routing it according to the algorithm, the topology reported in Figure 4.27 is obtained. It is possible to observe that it implements every one of the derived guidelines: There is only one node connected to the dominant one; The average distance from the dominant node is minimized, with the furthest away node being only one node away; Every chain of nodes departing from the dominant node is decreasing. The results are reported in Figure 4.28.

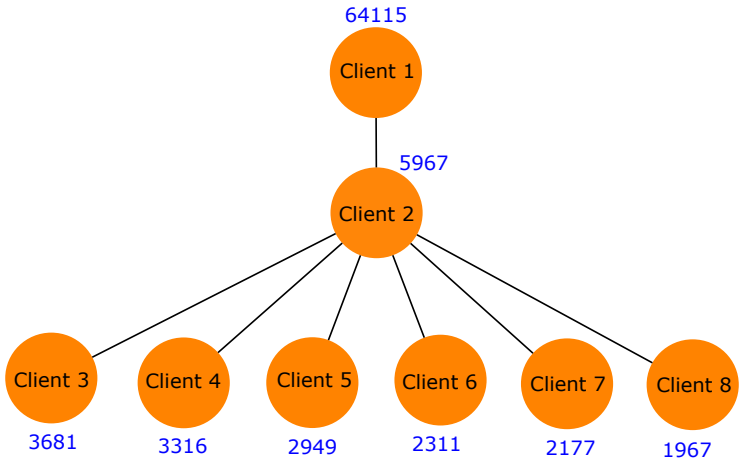


Figure 4.27 – Scheme of DFL with 8 clients for the routed fully-connected topology with non-i.i.d. distribution. The numbers in blue next to each node represents the number of images in the local dataset.

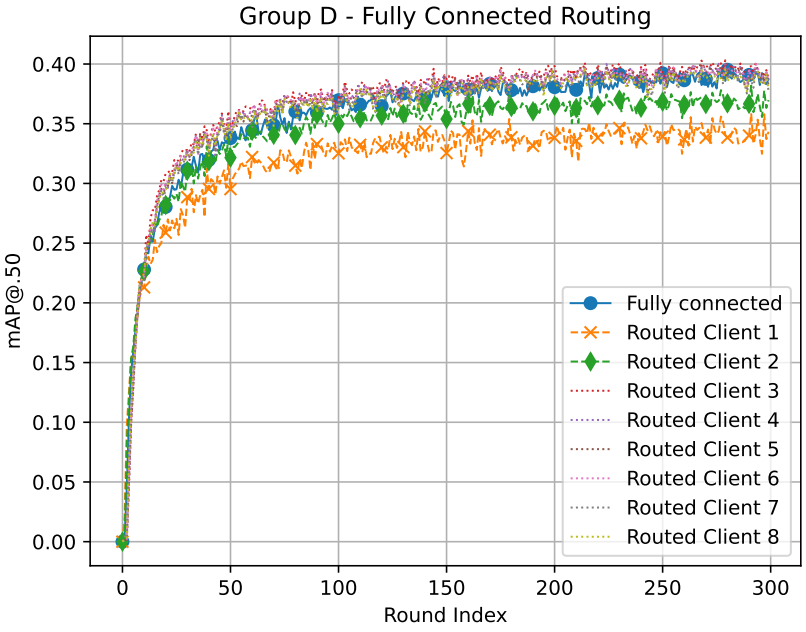


Figure 4.28 – Results of training for 300 epochs of DFL system for non-i.i.d. data distribution with 8 clients for the fully-connected topology and the routed version.

The routing allowed to reduce the number of connections from 56 to 7, a massive 8 times reduction in the communication overhead. The routed version obtained two nodes, namely clients 1 and 2, with a lower final accuracy than the fully-connected

one. This is because these nodes are limited by the dominant dataset. All other nodes obtained approximately the same final accuracy as the fully-connected, and improved slightly faster.

In order to grasp the importance of the algorithm in obtaining such performances, it is possible to compare it to the inverted line topology of the experiment C.2, which can be obtained from the fully connected topology by routing without following the guidelines. The graph comparing the approaches is reported in Figure 4.29.

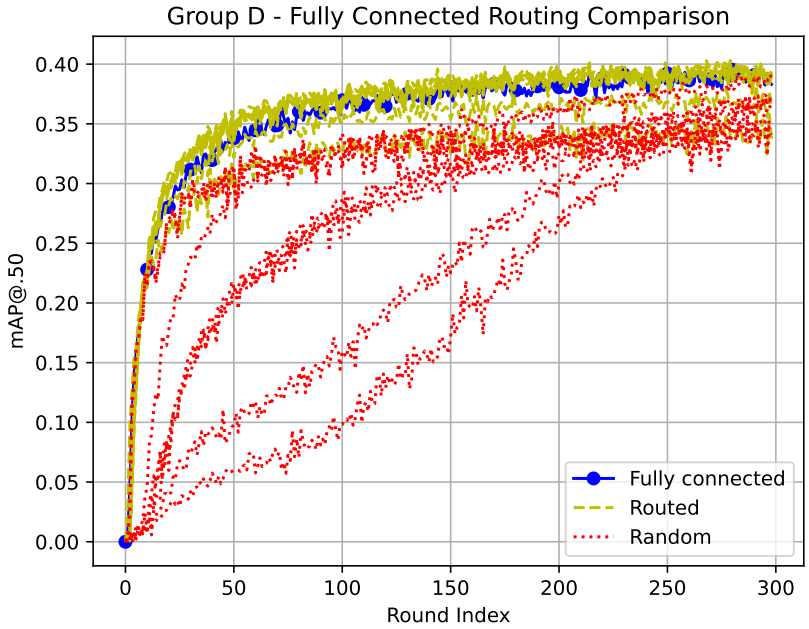


Figure 4.29 – Comparison between fully-connected, routed and random topologies.

From the graph, it is apparent that the routing algorithm is useful in preserving the performance while eliminating most connections.

4.4.2 Experiment D2: DFL with 8 clients routing on a meshed topology

In order to further validate the usefulness of the algorithm, we now apply it to a meshed topology. The difference is that now not every connection between the nodes is possible, and the algorithm has to choose between the existing ones. The meshed topology is reported in Figure 4.30, and the results of its training in Figure 4.31.

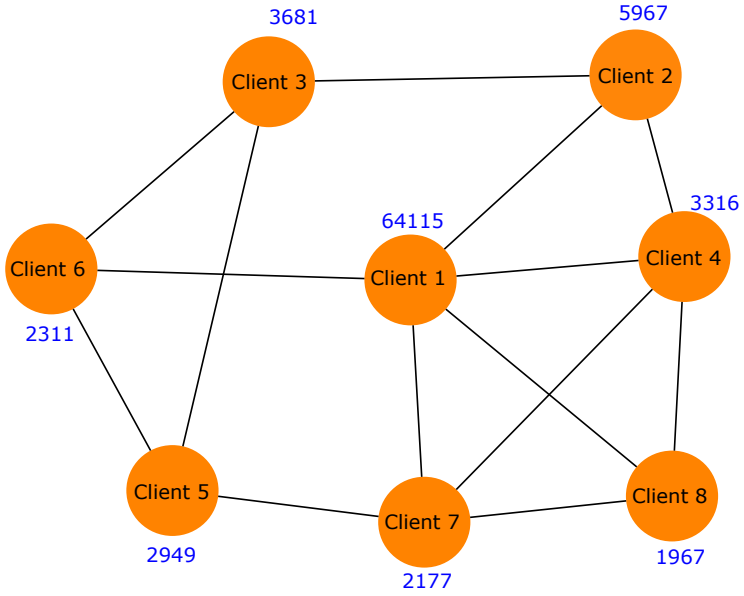


Figure 4.30 – Scheme of DFL with 8 clients for the meshed topology with non-i.i.d. distribution. The numbers in blue below the nodes represents the number of images in the local dataset.

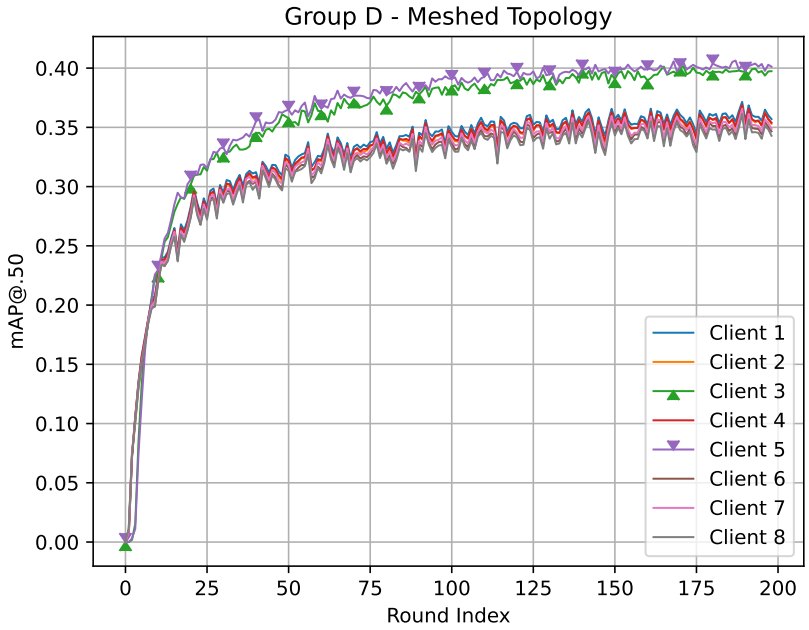


Figure 4.31 – Results of training for 200 epochs of DFL system for non-i.i.d. data distribution with 8 clients for the meshed topology.

As expected, the clients directly connected to the dominant client had their accurracy restricted, whereas the ones not directly connected achieved a higher accuracy.

By applying the routing algorithm as shown in Figure 3.11, the topology reported in Figure 4.32 is obtained, and its results are reported in Figure 4.33.

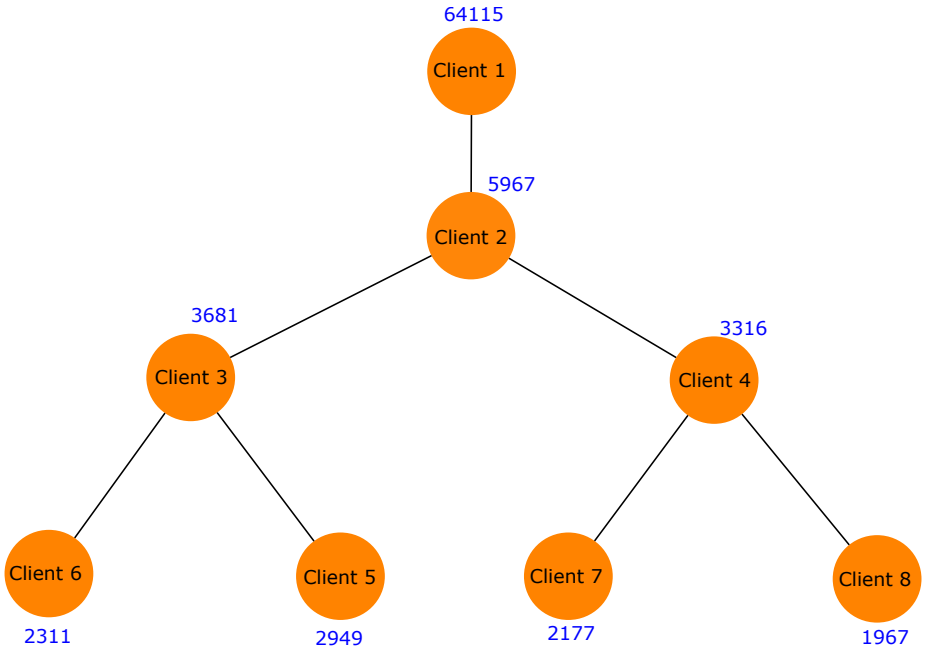


Figure 4.32 – Scheme of DFL with 8 clients for the routed meshed topology with non-i.i.d. distribution. The numbers in blue below the nodes represents the number of images in the local dataset.

The routing reduces the number of connections from 14 to only 7, thus reducing the communication overhead in half. The effects on the learning speed are minimal, with all nodes reaching the accuracy of the dominant node by round 13. Furthermore, a higher number of nodes achieve the higher accuracy since they are not directly connected to the dominant node.

If we compare these results to the ones reported in Figure 4.35, with a topology obtained with a random selection of nodes shown in Figure 4.34, it is possible to conclude that the algorithm was very useful in order to maintain a good performance. The random routing lead to very big reduction on performance, with some nodes not being able to come even close to the dominant node's performance during the analyzed time frame.

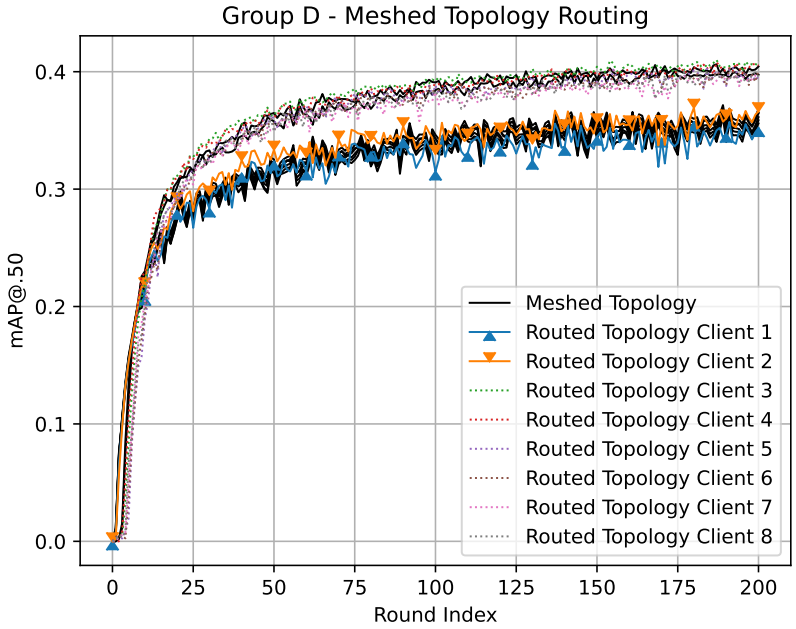


Figure 4.33 – Results of training for 200 epochs of DFL system for non-i.i.d. data distribution with 8 clients for the routed meshed topology, compared to the non-routed version.

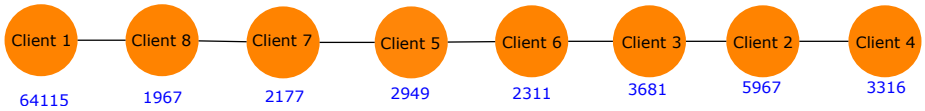


Figure 4.34 – Scheme of DFL with 8 clients for the randomly routed meshed topology with non-i.i.d. distribution. The numbers in blue below the nodes represents the number of images in the local dataset.

4.4.3 Experiment D3: DFL with 8 clients extended routing on a meshed topology

As previously explained, the basic algorithm presented some problems for some types of topologies. In order to address these cases, an extension has been proposed, which exploits the Dilution metric and a threshold to select between different routes leading to a certain node.

The advantage for topologies in which the dominant client acts as a bridge between the two portions of the network is immediate, as it would be impossible to connect all nodes without it.

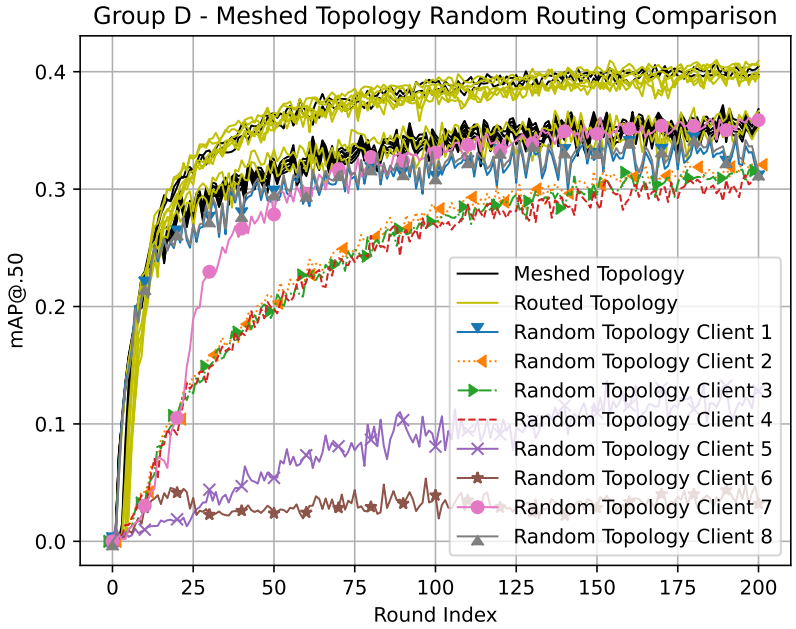


Figure 4.35 – Comparison between meshed, routed and random topologies.

This experiment, instead, aims to confirm the advantage in the second case, that is, when a much shorter path to a given node could be created by simply connecting one more client to the dominant node.

The used topology for this experiment is the one reported in Figure 3.13. We compare the results of the routing using the basic algorithm, which renders the topology in Figure 3.14, with the results of the routing using the extension, which renders the topology in Figure 3.17. The process of the application of the extension to this topology is explained in details in the same section.

The graphs are reported in Figure 4.36, Figure 4.37 and Figure 4.38, and the repetitions in Appendix B.

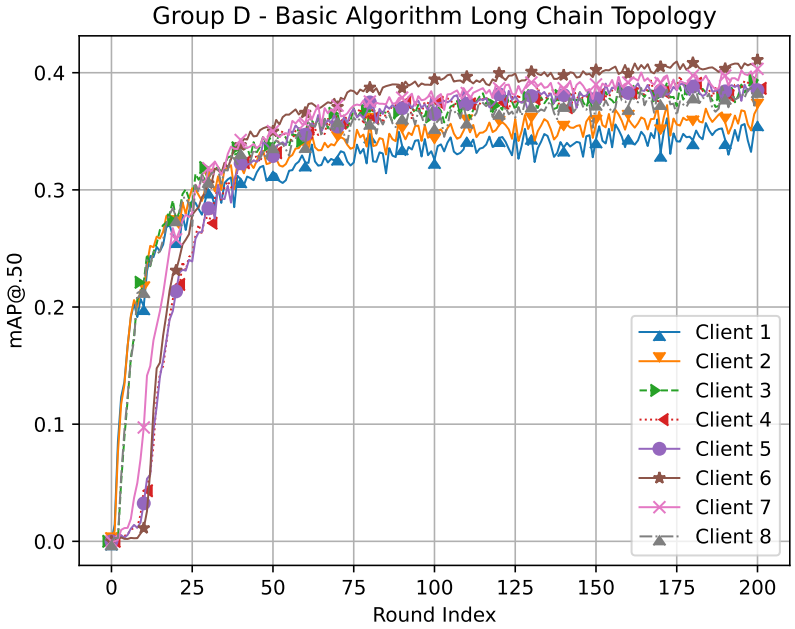


Figure 4.36 – Result of experiment with DFL 8 clients non-i.i.d. routed with the basic algorithm.

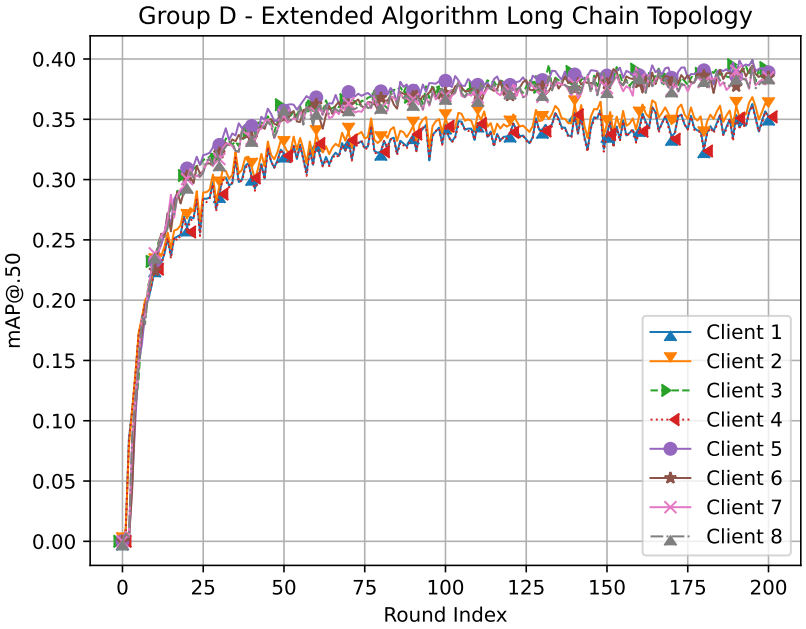


Figure 4.37 – Result of experiment with DFL 8 clients non-i.i.d. routed with the extended algorithm.

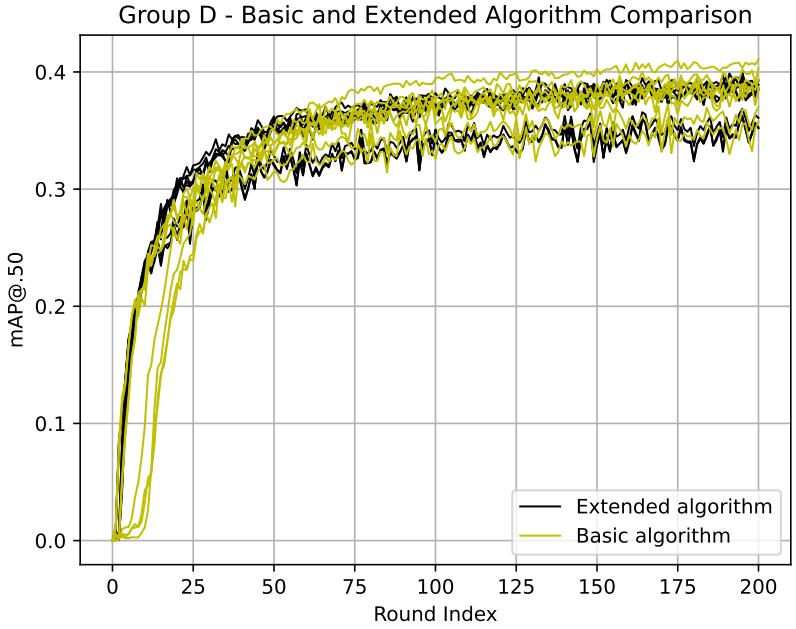


Figure 4.38 – Comparison of results of basic and extended routing algorithms.

From the results it is possible to observe an improvement in the learning speed of the nodes, specially for those whose Dilution metric fell below the threshold, thus confirming the usefulness of the metric in predicting a bad performance, as well as the usefulness of the extension to address this problem. This improvement came at the expense of one extra node connected to the dominant client, which limits its final accuracy. The trade off between learning speed and final accuracy is controlled by the threshold: Lower values of the threshold prioritizes keeping final accuracy high, by connecting few nodes to the dominant client, whereas higher values of the threshold prioritize the learning speed of all nodes, at the expense of lower final accuracy for other nodes, connected to the dominant client. The choice of the specific threshold depends on the application and specific data distribution, and should be subject to a parameter study. The extension thus allows for the application of the algorithm to more complex scenarios with larger scale, where the basic algorithm would be too simple and lead to a bad performance.

4.4.4 Conclusions Group D

A table summarizing the comparisons is reported in Table 4.5.

Topology	Number of connections	Number of nodes above dominant	Round last crossing
Fully Connected	56	0	0
Routed	7	6	8
Random	7	6	238
Meshed	14	2	6
Routed	7	7	13
Random	7	1	»200
Basic algorithm	7	6	33
Extended algorithm	7	5	7

Table 4.5 – Table comparing the performances of the routing for the three different scenarios tested. The number of nodes above the dominant node refers to the accuracy at the end of the analyzed time frame.

From the results presented in this group, it follows that the proposed routing algorithm is very useful for reducing the communication overhead, up to 8 times, while keeping the learning speed similar to the unrouted case, with the largest difference in the experiments being of 8 rounds, while increasing the number of nodes that obtain a higher accuracy in the analyzed time frame. A comparison with a random selection of nodes shows that an unthoughtful choice of connections can lead to a very big decrease in the performance, especially in the learning speed. Finally, the extended algorithm provides a way to address the problematic cases of the basic version by using the Dilution metric in order to decide the best path to reach a node, and it has been shown that it improves the learning speed under these conditions.

4.5 Group E: Node reliability

One of the main advantages of the proposed DFL system is the robustness against the node failures, that is, the fact that the ML task can continue even if any of the nodes fails. This group aims to explore the impact of a node failure in the learning process on a simple topology at different moments, considering the extremely unbalanced non-i.i.d. data distribution.

The utilized topology is the Ring with 4 clients, shown in Figure 4.11. The chosen node to fail is client 1, the dominant client, in order to simulate the most extreme case where the main source of information gets lost. The failure is simulated by cutting all connections to client 1 at the chosen communication round. Because of this setting, the experiment is two-in-one, since we can also interpret it as a failure of all other three nodes of the topology at the same time, leaving only client 1 running. The scheme of the experiment is shown in Figure 4.39. The graphs of the repetitions of the experiments with the 95% confidence interval is reported in Appendix B.

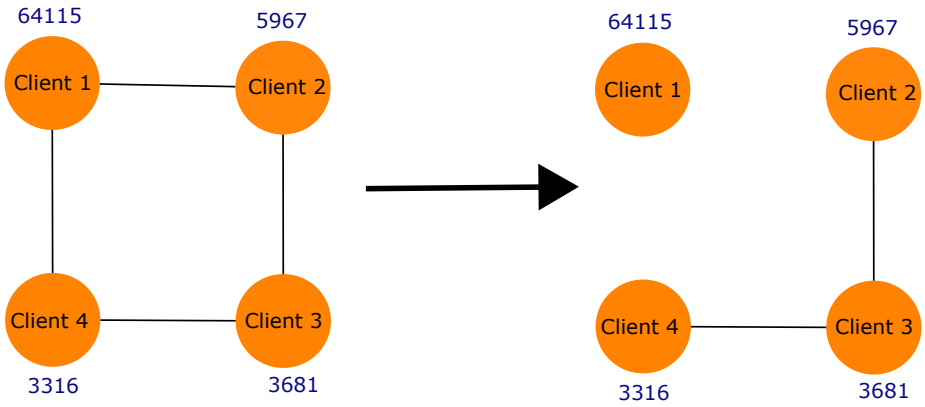


Figure 4.39 – Scheme of experiments with node failure for DFL 4 clients non-i.i.d. with Ring topology.

4.5.1 Experiment E1: Failure at round 50

The first experiment consists in simulating the failure at round index 50, when the two groups of nodes in the topology have already differentiated their performance, in order to understand how each one of them behaves. The results are reported in Figure 4.40.

From the graph it is possible to observe that client 1 continues to improve its accuracy, since it has a big enough dataset on its own. However, it achieves a slightly lower final accuracy than the version with no failures in the 300 epochs, since it does not have access to the information contained in other datasets. The three other clients, on the other hand, tend to slowly decrease their accuracy after the failure,

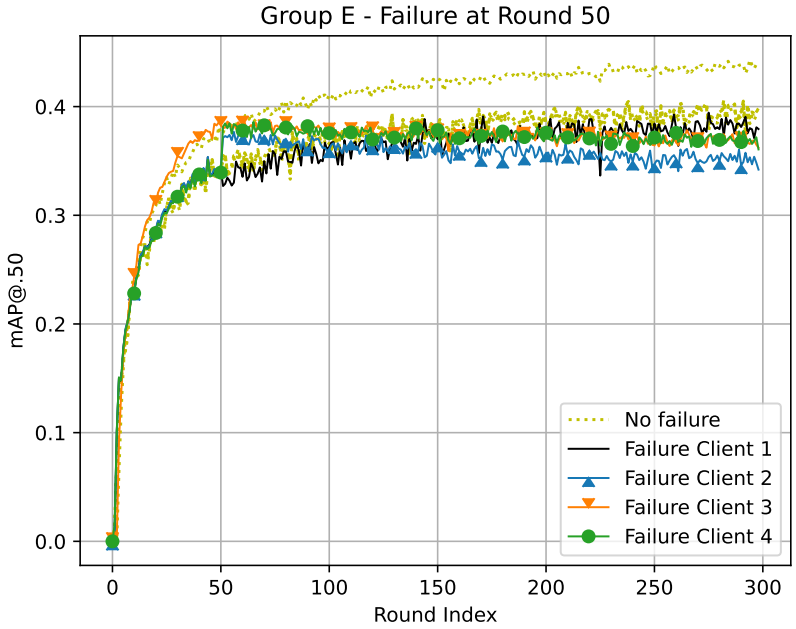


Figure 4.40 – Graph failure at round 50 DFL 4 clients non-i.i.d. with Ring topology.

since now they are training without the biggest dataset, and thus tend to converge to their unrepresentative local optima, achieving by the end of the 300 epochs a significantly lower accuracy. It is also worth noticing that clients 2 and 4 presented a sudden change in accuracy when the failure occurred. The reason for that is that those clients were directly connected to client 1, and when this connection got broken, their combination rule suffered a drastic change, giving way more weight than before to the updates coming from client 3, thus generating this sudden change towards its accuracy. Client 3, however, was not directly connected to client 1, and thus the failure did not induce any change in its combination rule, generating no drastic changes in accuracy.

4.5.2 Experiment E2: Failure at round 10

In this experiment, the failure is simulated at round index 10, in order to understand how a difference in the timing affects the performance. The results are shown in Figure 4.41.

Client 1 presented the same behavior as in the case of failure at round 50, further confirming the independence of this node from what happens to the rest of the topology. The other three clients in this case slowly improved their accuracy,

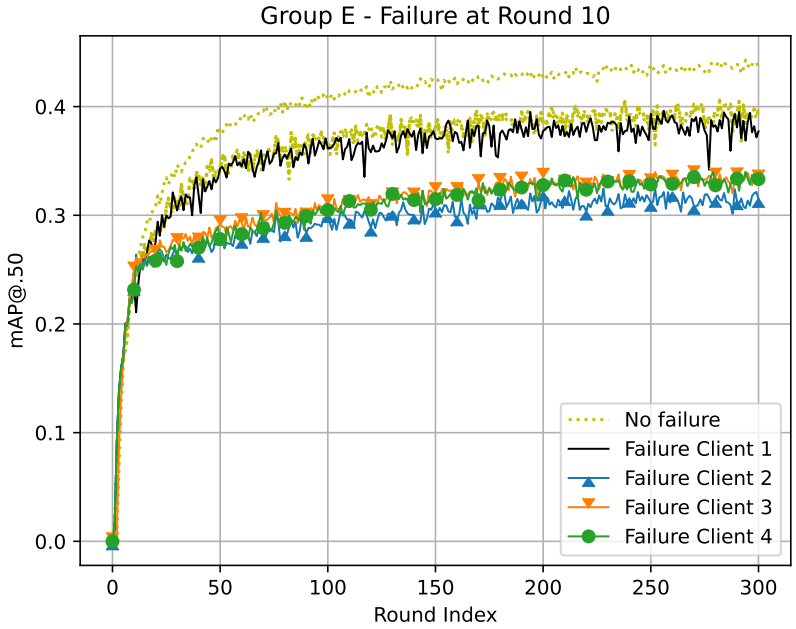


Figure 4.41 – Graph failure at round 10 DFL 4 clients non-i.i.d. with Ring topology.

exploiting the information that they have in total. By the end of the 300 epochs, they have obtained a lower accuracy than in the round 50 case, but it is expected that, if simulated for longer, both cases would tend to the same accuracy, determined by the information contained in their datasets.

4.5.3 Conclusions Group E.

From these experiments it is possible to conclude that, indeed, the system presents robustness against node failures. The loss of a node does impact the performance in a significant way, but the group of nodes still manages to continue the training, and even to improve, as observed in the second experiment. Node failures may impact the system in different ways, such as removing data, changing combination rules and altering the topologies, which may have more severe impacts in other configurations, that need to be explored in a deeper dedicated analysis.

Chapter 5

Conclusion

This work explored the impact of the topology of a DFL system for an extremely unbalanced non-i.i.d. data distribution among the clients, with one client concentrating most of the data, deriving from a non-i.i.d. partitioning of an OD dataset with a high degree of contextual information. Under these conditions, the topology assumes an even higher importance, since the nodes have different behaviors and impact differently their neighboring nodes, depending on the distribution and amount of data that they contain. From the insights of this analysis, a routing algorithm has been proposed in order to minimize the resource consumption by reducing the amount of connections between clients while limiting the degradation of the performance.

From the initial experiments with the Ring and the Line topology it was possible to observe their impact on the accuracy and learning speed of the clients, and the main factors affecting them were identified as: presence of direct connections to dominant node; number of nodes between a given node and the dominant one; order of the nodes in the chain connecting a given node to the dominant one, in terms of increasing or decreasing number of samples; size and representativeness of a node's own dataset.

Based on these factors, general guidelines for the improvement of the system's performance were derived: connect as few nodes as possible to the dominant node, as it will restrict their performance in terms of final accuracy; reduce the average distance between each node of the topology and the dominant node; for each chain of nodes, try to make it as decreasing as possible, in order to limit the dilution of the dominant node updates as it propagates.

Based on these guidelines, a routing algorithm was designed with the aim of selecting the minimum amount of connections necessary while preserving the performance of the system. This algorithm allows to reduce the communication overhead of the system, which may be a critical bottleneck, especially in cases with a massive number of devices and limited resources.

The algorithm was then validated on two different scenarios, one with a fully-connected topology, and the other with a meshed topology. Both showed that the algorithm is very useful in preserving the performance of the system: it was able to reduce by 8 times the communication overhead in one of the cases, while increasing only slightly the number of rounds needed for all nodes to reach the accuracy of the dominant node, and even allowing for more nodes to reach higher levels of accuracy. When compared to a random selection of the minimum number of connections its usefulness becomes even more apparent, since these cases showed a massive degradation of the learning speed.

In order to address problematic cases of the basic algorithm, an extension has been proposed, that generates alternative routings and leverages the Dilution metric and a threshold in order to make decisions about the best path to reach a node. This extension was then validated in one of the problematic cases of the basic algorithm, when the resulting topology presents a long chain of nodes, and it was shown to improve the learning speed, thus generalizing the algorithm to more complex situations.

As a final exploration of the system, the robustness to node failures has been explored, by simulating the failure of the dominant client in a simple configuration. The results showed that the system can improve even in the case of failures, thus supporting the claim that this approach is useful for scenarios with low node reliability, even with the specific data distribution under study.

In conclusion, this work presents a performance aware solution to the reduction of the communication overhead in a DFL system in resource constrained situations, in which a single entity of the system concentrates most of the information available, and thus acts as a dominant client. Such an approach can be used in conjunction with the traditional approaches to the reduction of the communication cost in order to enable the development of future applications based on massive number of devices with limited resources and complex models.

As future developments from this work, the proposed algorithm can be evaluated in more complex scenarios with larger networks, and other aspects can be weighted, such as the robustness of the generated topologies. A careful study of the trade offs generated by the threshold, and an extension of the metric to include other characteristics in the path decision are also possible developments. Another direction could be the proposition of a protocol in order to apply this algorithm in real time, adapting the topology to the evolution of the scenario.

Appendix A

Parameters

```
# Hyperparameters for COCO training from scratch
# python train.py --batch 40 --cfg yolov5m.yaml --weights '' --data coco.yaml --img 640 --epochs 300
# See tutorials for hyperparameter evolution https://github.com/ultralytics/yolov5#tutorials


lr0: 0.001  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.2  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937  # SGD momentum/Adam beta1
weight_decay: 0.0005  # optimizer weight decay 5e-4
warmup_epochs: 3.0  # warmup epochs (fractions ok)
warmup_momentum: 0.8  # warmup initial momentum
warmup_bias_lr: 0.1  # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.5  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 1.0  # obj loss gain (scale with pixels)
obj_pw: 1.0  # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
# anchors: 3  # anchors per output layer (0 to ignore)
fl_gamma: 0.0  # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015  # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
```

```
32 mosaic: 1.0  # image mosaic (probability)
33 mixup: 0.0  # image mixup (probability)
```

Appendix B

Experiments repetitions

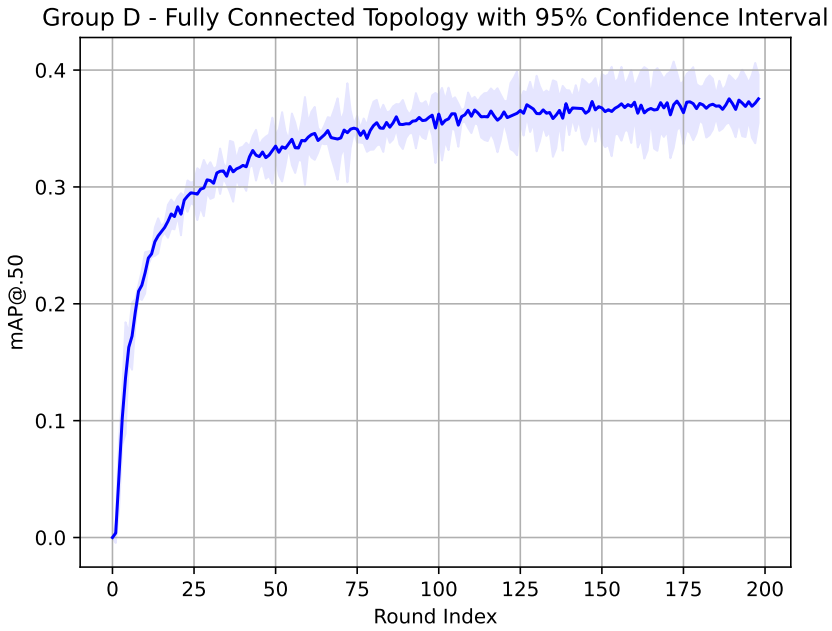


Figure B.1 – Graph of three repetitions of the fully connected topology results, with the 95% confidence interval reported.

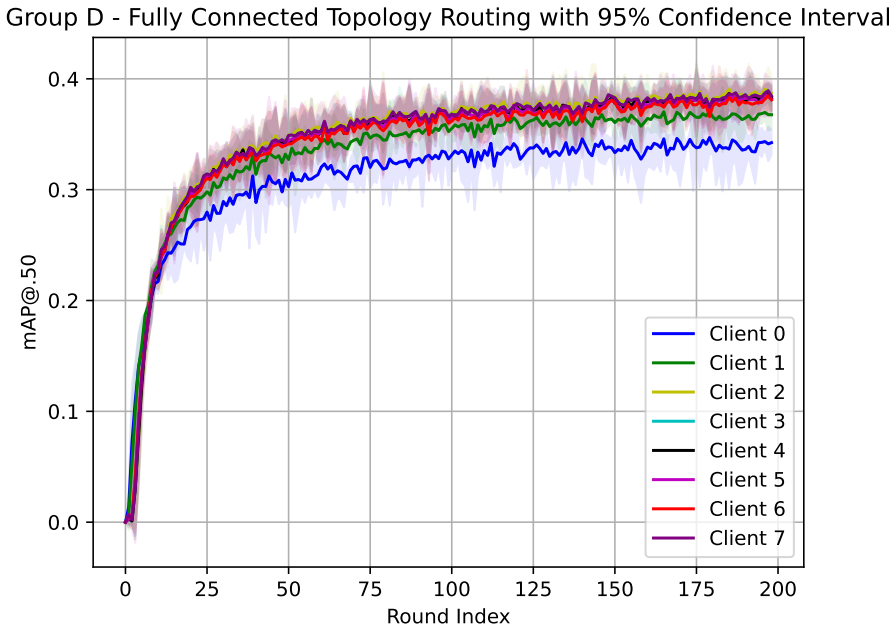


Figure B.2 – Graph of three repetitions of the routed fully connected topology results, with the 95% confidence interval reported.

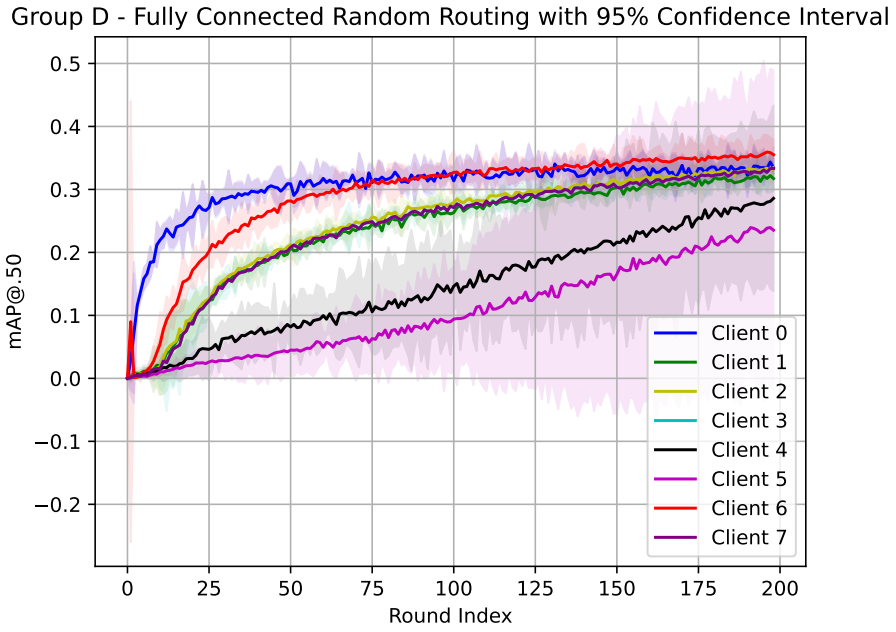


Figure B.3 – Graph of three repetitions of the randomly routed fully connected topology results, with the 95% confidence interval reported.

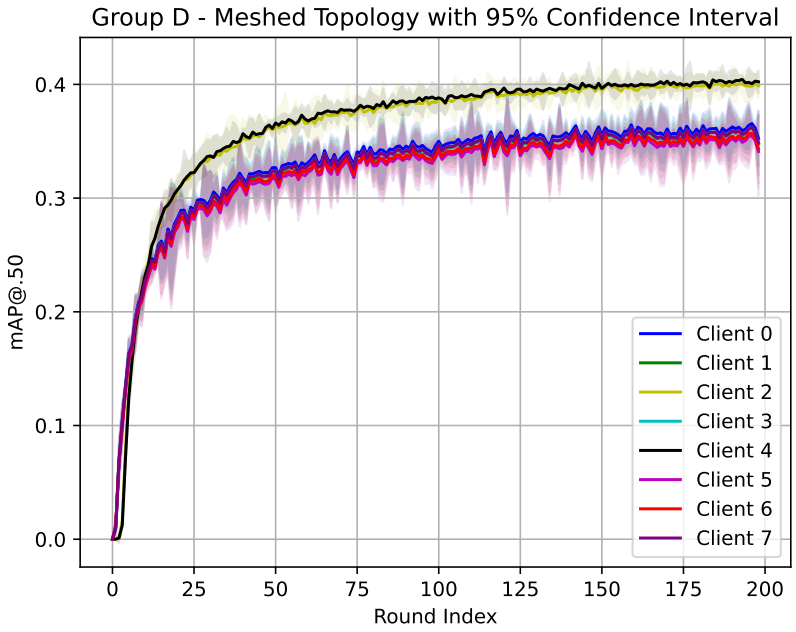


Figure B.4 – Graph of three repetitions of the meshed topology results, with the 95% confidence interval reported.

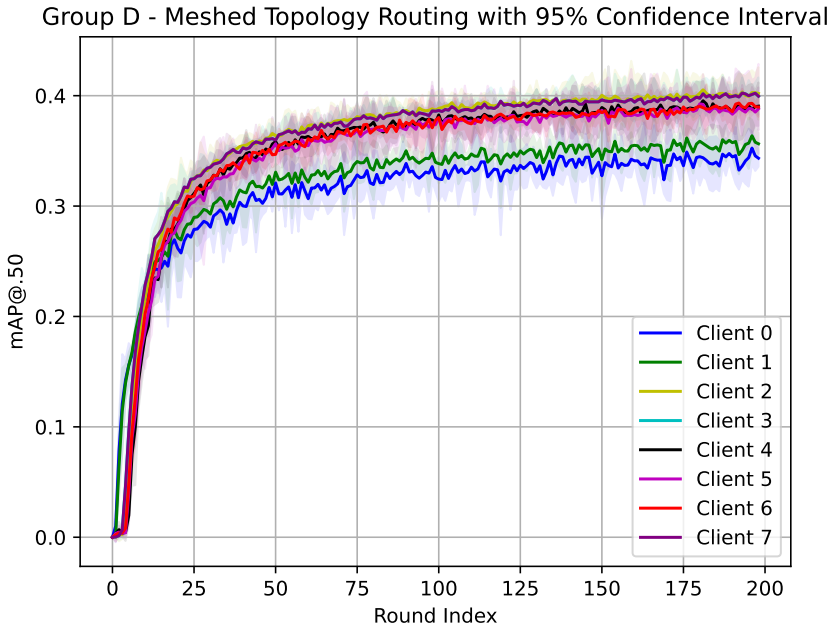


Figure B.5 – Graph of three repetitions of the routed meshed topology results, with the 95% confidence interval reported.

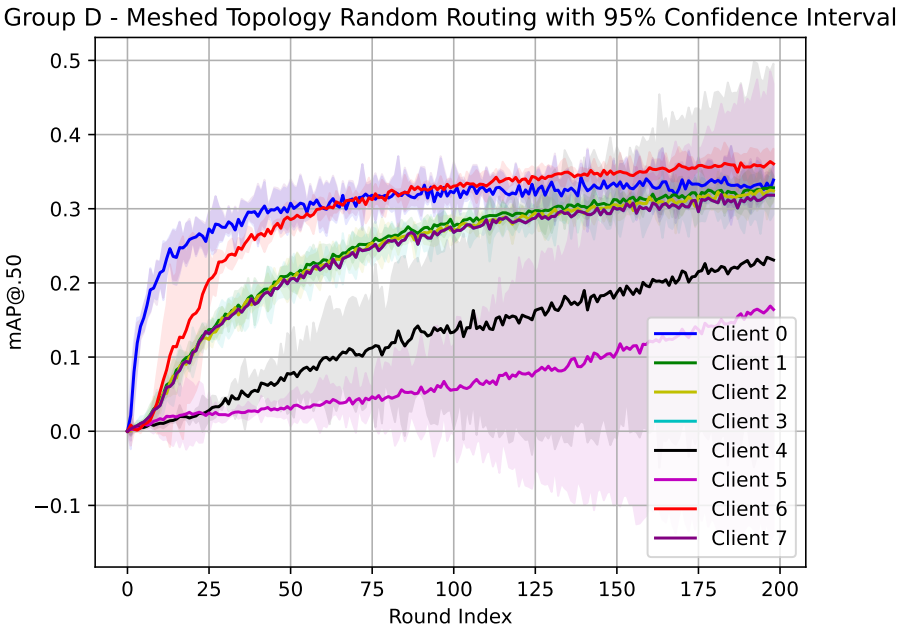


Figure B.6 – Graph of three repetitions of the randomly routed meshed topology results, with the 95% confidence interval reported.

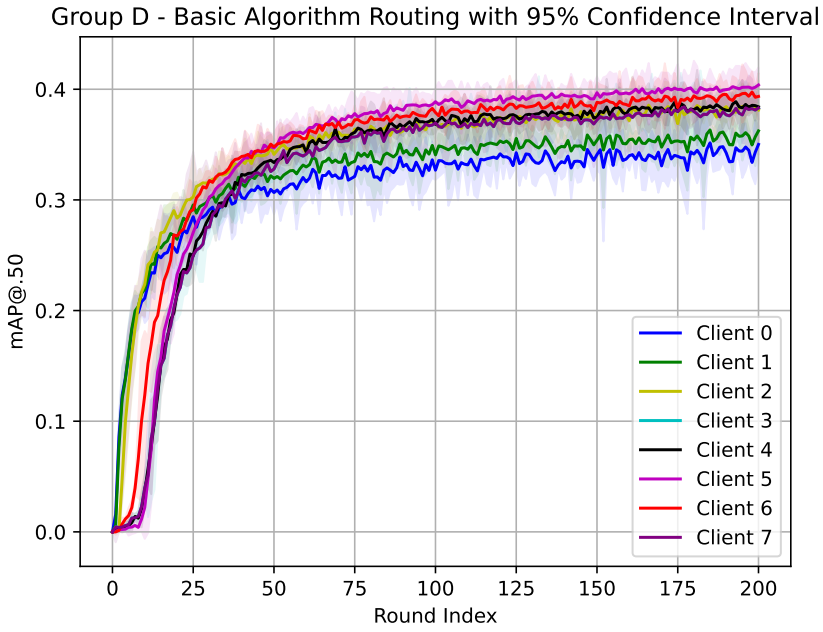


Figure B.7 – Graph of three repetitions of the long chain topology routed with the basic algorithm, with the 95% confidence interval reported.

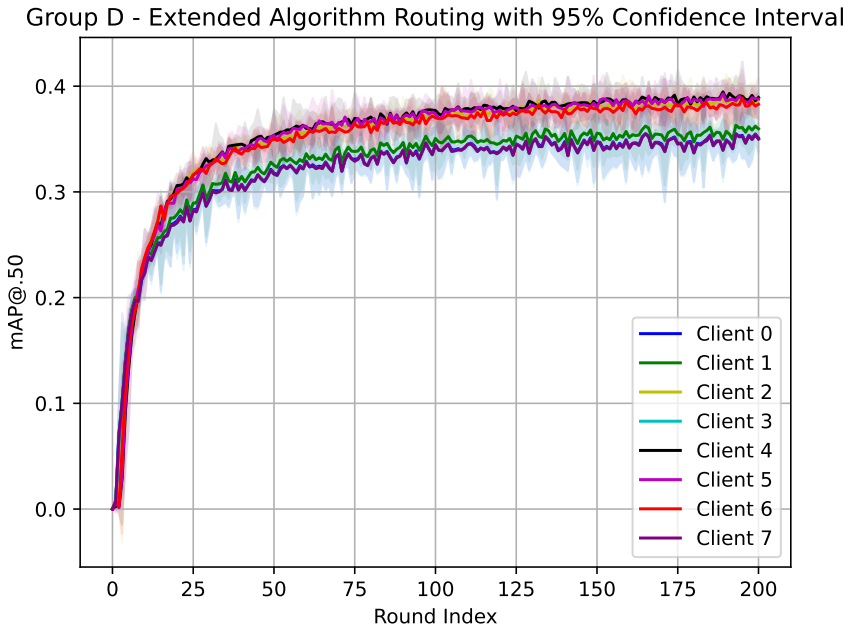


Figure B.8 – Graph of three repetitions of the long chain topology routed with the extended algorithm, with the 95% confidence interval reported.

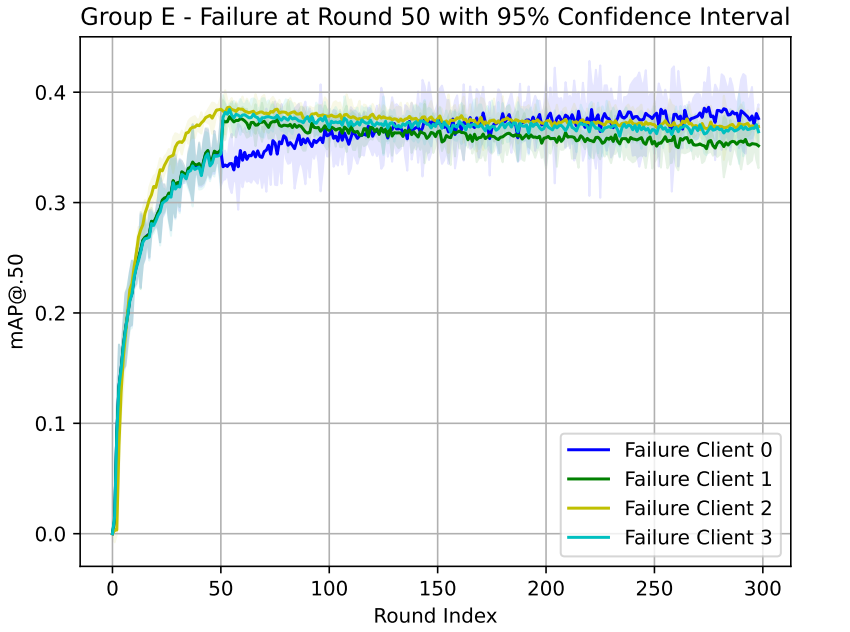


Figure B.9 – Graph of three repetitions of the node failure at round 50 experiment, with the 95% confidence interval reported.

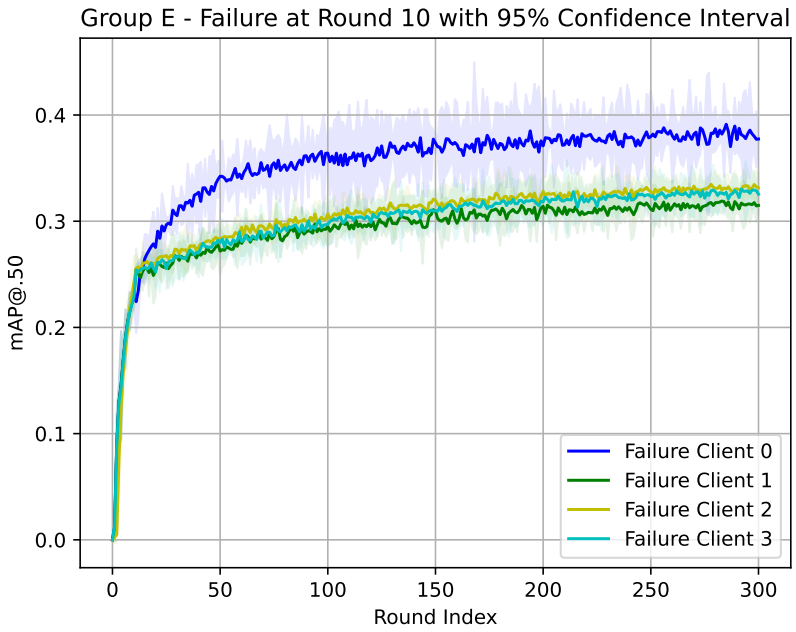


Figure B.10 – Graph of three repetitions of the node failure at round 10 experiment, with the 95% confidence interval reported.

List of Abbreviations

AP	Average Precision
API	Application Programming Interface
AWS	Abnormal Weight Supression
CFA	Consensus based Federated Averaging
CFA-GE	Consensus based Federated Averaging with Gradient Exchanges
CFL	Centralized Federated Learning
CNN	Convolutional Neural Network
CV	Computer Vision
DFedAvgM	Decentralized FedAvg with Momentum
DFL	Decentralized Federated Learning
DNN	Deep Neural Network
EMD	Earth Movers Distance
FedAvg	Federated Averaging
FL	Federated Learning
FN	False Negative
FP	False Positive
FTL	Federated Transfer Learning
GDPR	General Data Protection Regulation
GPU	Graphics Processing Unit
HFL	Horizontal Federated Learning
i.i.d.	Independently and Identically Distributed
IIoT	Industrial Internet of Things
IoT	Internet of Things
IoU	Intersection over Union
KLD	Kullback-Leibler Divergence
mAP	mean Average Precision
ML	Machine Learning
MPI	Message Passing Interface
NN	Neural Network
OD	Object Detection

SELF-UN	Serverless FL for UAV Networks
SGD	Stochastic Gradient Descent
SGDM	Stochastic Gradient Descent with Momentum
TP	True Positive
UAV	Unmanned Aerial Vehicles
VFL	Vertical Federated Learning
YOLO	You Only Look Once

List of Figures

List of Tables

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. DOI: 10.1038/nature14539.

[2] Y. Liu, Y. Gao, and W. Yin, "An Improved Analysis of Stochastic Gradient Descent with Momentum," arXiv, math.OC, Jul. 2020.

[3] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," arXiv, cs.LG, Feb. 2016.

[4] European Parliament and Council of the European Union, Regulation (EU) 2016/679 of the European Parliament and of the Council, of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), OJ L 119, 4.5.2016, p. 1–88, May 4, 2016.

[5] L. U. Khan, W. Saad, Z. Han, E. Hossain, and C. S. Hong, "Federated Learning for Internet of Things: Recent Advances, Taxonomy, and Open Challenges," arXiv, cs.NI, Sep. 2020.

[6] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated Machine Learning: Concept and Applications," arXiv, cs.AI, Feb. 2019.

[7] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, "A Survey on Federated Learning for Resource-Constrained IoT Devices," *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, Jan. 2022. DOI: 10.1109/jiot.2021.3095077.

[8] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, Y. Li, X. Liu, and B. He, "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 4, pp. 3347–3366, Apr. 2023. DOI: 10.1109/tkde.2021.3124599.

[9] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated Learning for Internet of Things: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 3, pp. 1622–1658, Jul. 2021. DOI: 10.1109/comst.2021.3075439.

[10] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated Learning: Challenges, Methods, and Future Directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, May 2020. DOI: 10.1109/msp.2020.2975749.

[11] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, Sep. 2020. DOI: 10.1109/tnnls.2019.2944481.

[12] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," arXiv, cs.LG, Dec. 2018.

[13] E. T. M. Beltrán, M. Q. Pérez, P. M. S. Sánchez, S. L. Bernal, G. Bovet, M. G. Pérez, G. M. Pérez, and A. H. Celdrán, "Decentralized Federated Learning: Fundamentals, State of the Art, Frameworks, Trends, and Challenges," arXiv, cs.LG, Nov. 2022.

[14] Y. Qu, H. Dai, Y. Zhuang, J. Chen, C. Dong, F. Wu, and S. Guo, "Decentralized Federated Learning for UAV Networks: Architecture, Challenges, and Opportunities," *IEEE Network*, vol. 35, no. 6, pp. 156–162, Nov. 2021. DOI: 10.1109/mnet.001.2100253.

[15] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object Detection in 20 Years: A Survey," arXiv, cs.CV, May 2019.

[16] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, pp. 303–308, Sep. 2009.

[17] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," arXiv, cs.CV, May 2014.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," arXiv, cs.CV, Jun. 2015. DOI: 10.48550/ARXIV.1506.02640.

[19] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv, cs.CV, Apr. 2018.

[20] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated Learning with Non-IID Data," arXiv, cs.LG, Jun. 2018. DOI: 10.48550/ARXIV.1806.00582.

[21] P. Yu and Y. Liu, "Federated Object Detection," in *3rd International Conference on Vision, Image and Signal Processing*, Vancouver, Canada: Association for Computing Machinery (ACM), Aug. 2019. DOI: 10.1145/3387168.3387181.

[22] J. Luo, X. Wu, Y. Luo, A. Huang, Y. Huang, Y. Liu, and Q. Yang, "Real-World Image Datasets for Federated Learning," arXiv, cs.CV, Oct. 2019.

[23] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, et al., "FedML: A Research Library and Benchmark for Federated Machine Learning," arXiv, cs.LG, Jul. 2020.

[24] C. He, A. D. Shah, Z. Tang, D. F. N. Sivashunmugam, K. Bhogaraju, M. Shimpi, L. Shen, X. Chu, M. Soltanolkotabi, and S. Avestimehr, "FedCV: A Federated Learning Framework for Diverse Computer Vision Tasks," arXiv, cs.CV, Nov. 2021.

[25] T. Sun, D. Li, and B. Wang, "Decentralized Federated Averaging," arXiv, cs.DC, Apr. 2021.

[26] W. Liu, L. Chen, and W. Zhang, "Decentralized Federated Learning: Balancing Communication and Computing Costs," arXiv, cs.LG, Jul. 2021.

[27] S. Savazzi, M. Nicoli, and V. Rampa, "Federated Learning With Cooperating Devices: A Consensus Approach for Massive IoT Networks," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4641–4654, May 2020. DOI: 10.1109/jiot.2020.2964162.

[28] E. Georgatos, C. Mavrokefalidis, and K. Berberidis, "Fully Distributed Federated Learning with Efficient Local Cooperations," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Rhodes Islands, Greece: Institute of Electrical and Electronics Engineers (IEEE), Jun. 2023. DOI: 10.1109/icassp49357.2023.10095741.

[29] H. Kavalionak, E. Carlini, P. Dazzi, L. Ferrucci, M. Mordacchini, and M. Coppola, "Impact of Network Topology on the Convergence of Decentralized Federated Learning Systems," in *2021 IEEE Symposium on Computers and Communications (ISCC)*, Athens, Greece: Institute of Electrical and Electronics Engineers (IEEE), Sep. 2021. DOI: 10.1109/iscc53001.2021.9631460.

[30] A. Bellet, A.-M. Kermarrec, and E. Lavoie, "D-Cliques: Compensating for Data Heterogeneity with Topology in Decentralized Federated Learning," arXiv, cs.LG, Apr. 2021.

[31] Y. Qu, H. Dai, Y. Zhuang, J. Chen, C. Dong, F. Wu, and S. Guo, "Decentralized Federated Learning for UAV Networks: Architecture, Challenges, and Opportunities," arXiv, cs.NI, Apr. 2021.