i-WSN League: Clustered Distributed Learning in Wireless Sensor Networks

Joannes Sam Mertens, Laura Galluccio, and Giacomo Morabito

***Abstract*—In this work, *i-WSN League*, a comprehensive hardware/software framework for the support of distributed training and inference is introduced. For what concerns the hardware, in *i-WSN* League two types of nodes are considered, namely, head nodes and common nodes. Head nodes are resource-rich nodes that have the capabilities for training artificial neural network. Common nodes collect data and can execute inference only. In *i-WSN* League, all nodes are grouped in Clusters, each with a Cluster Head (selected among the head nodes), which is the only node responsible for training. To this end, the data coming from all nodes in the Cluster can be utilized. This, however, involves a large exchange of data which might be unsustainable by common nodes. Thus, only part of the data collected by common nodes is sent to the Cluster Heads and a network of Cluster Heads will implement distributed learning in a peer-to-peer fashion. As compared to state-of-the-art literature, the key contributions of our work are related to the combination of gossiping and clustering to adapt the operations executed by each node to its capabilities, with the aim of minimizing the energy consumption in resource-limited nodes, while preserving accuracy. In this article, *i-WSN* League is assessed in a simple scenario in which a wireless sensor network monitors the air pollution in a large city. Performance results obtained by considering auto-encoders prove the effectiveness of the proposed scheme as well as its balanced energy consumption and fairness in resource consumption distribution.**

***Index Terms*—Clustering, distributed learning, wireless sensor networks.**

I. Introduction

THE EXECUTION of the machine learning (ML) algorithms into small and low-power devices has attracted the attention of researchers and has opened the path to new use cases in the context of wireless sensor networks, WSNs. The introduction and success of tools, like Tensorflow Lite,[1] represent both the evidence of the interest of the ML community toward such scenarios and a fundamental step toward their realization. However, performing on-device training requires energy, memory, and computing capabilities which are not available in most hardware platforms employed for WSNs.

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

1) Such type of transmissions might require the use of a large amount of communication and energy resources.
2) There might be security and privacy issues as the data transmitted by the nodes can be the target of attacks in its way toward the server.

In this article, we address such issues and propose a framework for the realization of intelligent wireless sensor networks (*i*-WSNs) which minimize the exchange of information between WSN nodes. We assume that the WSN includes some nodes that have enough resources to execute the ML model training [1]. All network nodes are divided into clusters and in each cluster there is a resource-rich node which is in charge for the training of the ML model that will be used by all nodes in the cluster. Such node is, thus, a Cluster Head and executes training by using only the data which is locally available. The resulting model is sent to all nodes in the cluster which will execute it to evaluate a fitness metric. Nodes that obtain a low value of such fitness metric will transmit a part of their data to the Cluster Head which will use it to retrain the model.

Clusters, in parallel, will exchange their models in a peer-to-peer manner and, thus, cooperate in forming a so-called *league*. For this reason, we denote the proposed solution as *i-WSN League*.

In this article, we present *i-WSN* League and assess it in a simple application scenario in which a wireless sensor network monitors the air pollution in a large city. The network nodes will cooperate to train a neural network that can be used for anomaly detection. Note that this is just one example of several scenarios in which distributed learning in wireless sensor networks can be exploited. Other relevant examples include, but are not limited to, chemical attack identification, early wildfire, and other natural disasters detection. In this article, the assessment has been carried out by considering different settings for what concerns the number of Cluster Heads, the type of clustering, and other parameters characterizing the distributed training algorithm

The remainder of this article is organized as follows. In Section II, we provide an overview of the most relevant literature on distributed learning. In Section III, we introduce i-WSN League. Its performance is evaluated in Section IV. Finally, in Section V, we draw our concluding remarks.

Manuscript received 19 January 2023; revised 18 April 2023; accepted 15 May 2023. Date of publication 31 May 2023; date of current version 9 October 2023. This work was supported in part by the POR Sicilia under Contract SAFE-DEMON, and in part by the European Union (NextGeneration EU) through the MUR-PNRR Project SAMOTHRACE under Grant ECS00000022. *(Corresponding author: Giacomo Morabito.)*

The authors are with the CNIT Research Unit, Department of Electrical, Electronics and Computer Engineering, University of Catania, 95125 Catania, Italy (e-mail: giacomo.morabito@dieei.unict.it).

Digital Object Identifier 10.1109/JIOT.2023.3280843

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

II. Background and Motivations

ML can be applied in WSNs in several contexts and a rich literature on the subject exists, as summarized in [2].

A large part of the research effort in this domain has focused on the application of distributed learning [3] because it is more secure and efficient for what concerns the use of communication resources. Distributed ML has been recently proposed for anomaly detection. For example, in the scenario considered in the following Section IV, cluster members (CMs) inherit from the cluster heads a trained model and they only perform inference based on their data. If the current collected values differ from the one obtained by using the model inherited by the cluster head for more than a given amount, an anomalous condition is identified and, thus, an alert is generated.

In the context of distributed learning, a fundamental contribution was given in [4], where the problem and the relevant constraints for distributed learning in WSNs are clearly stated and a general model is given. In [4], several crucial key issues and concepts are explored. In particular, relevant examples are discussed as well as possible architectures, including the use of clustering and the possibilities opened to distributed learning by the multihop communication paradigm, which is typical of most WSN scenarios.

The major issues in the execution of ML in WSNs are related to the limits in available processing and communication capabilities typical of usual WSN scenarios. To cope with such limitations, one possibility is to apply techniques for the representation of information which trade accuracy for efficiency. An overview of the literature in this context is given in [3], while recent relevant efforts include the works presented in [5] and [6].

In this article, we focus on the exploitation of the multihop communication paradigm for the effective and efficient support of ML in WSNs. In the recent literature, some efforts toward the definition of strategies for the diffusion and use of information in networks of nodes, that, collectively aim at achieving knowledge about the status of the environment, emerged [7]. In the context of distributed learning, federated learning (FL) represents today the state-of-the-art approach. In FL there are several, say K, *federated learners*, each of which, say the kth, maintains a part of the data set and uses it to train a neural network, which we represent through its parameters collected in the array $\mathbf{w}_k$. Federated learners send their model parameters to a central node, referred to as *aggregation point*, which creates an aggregated model $\mathbf{w}$ as

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

where the parameter p_k may depend on the size of the portion of the data set maintained by the kth federated learner and must be such that $p_1 + p_2 + \cdots + p_K = 1$. The aggregated model $\mathbf{w}$ is then distributed to the federated learners. They will retrain it using the local data. The above steps, collectively referred to as *iteration*, are repeated several times until the model converges as desired.

Note that FL is efficient in terms of accuracy and convergence of the model. However, it is obvious that at each of the above-mentioned iterations, it is necessary to transmit the model parameters to the aggregation point. Consequently, many researchers have recently raised concerns regarding the communication cost associated to the transmission of the model parameters [8].

FL has been deployed in practice by major companies. Some examples of the use of FL by OTP and vendors are listed in the following [9].

1) Google is applying FL in the Gboard mobile keyboard as well as in Pixel phones [10] and in Android Messages [11].
2) Apple is using cross-device FL in iOS 13 for applications like the vocal classifier for "Hey Siri" [12].
3) doc.ai is developing cross-device FL solutions for medical research [13].
4) Snips has explored cross-device FL for hotword detection [14].

FEDAVG [15] is a well-known and the most studied FL algorithm in the literature. It combines stochastic gradient descent (SGD) on each client node with an aggregation point that performs model averaging. Since the introduction of FEDAVG, there have been many works on the parallel implementation of SGD. One of such works, presented in [16], proposes several SGD algorithms to reach a tradeoff between the training time and the transmission time. On applying this approach with deep neural networks, the end-to-end training time was reduced significantly. Similarly, a theoretical analysis was carrier out on SGD with k-sparsification [17]. The above scheme keeps track of the accumulated errors in memory to reduce communication time. However, the convergence rate was reported to be the same as the vanilla SGD algorithm. A decentralized stochastic optimization approach is proposed in [18], in which the nodes communicate the model updates only to their neighbors. In order to reduce the communication bottleneck, the model updates are compressed (quantized or sparsified) by the nodes. A similar approach is proposed in [19] for decentralized learning in which the communication estimates among the users are quantized. Similar compression and quantization solutions are studied in [20] and [21] to solve the distributed optimization problems with communication constraints.

FL has certain limitations and constraints. One of such critical limitations is *Catastrophic Forgetting* [22], [23], [24] which can be a critical problem in all distributed ML techniques. Catastrophic forgetting is the tendency of a neural network model to abruptly forget previously learned information upon learning new information. The local retraining in FL may result in catastrophic forgetting of the knowledge learned from other participants. However, several approaches have been proposed in the literature to lock the learned knowledge to overcome catastrophic forgetting in FL and other distributed ML techniques. One of such knowledge lock approaches is proposed in [25] in which knowledge distillation techniques are utilized to preserve the previously learned knowledge. This solution exhibits better performance in terms of accuracy when compared to other FL approaches, such as FedAvg. In the solution introduced in [26], the old and new neural network models are fused to avoid the catastrophic forgetting issue of

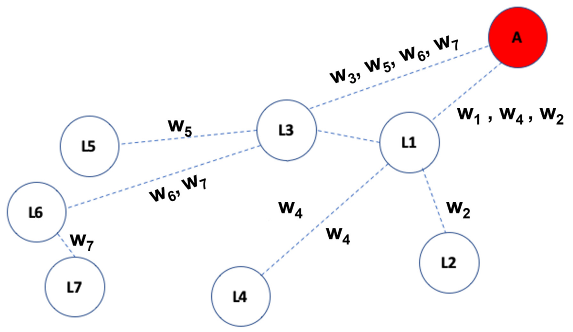

Fig. 1. Example of application of an FL mechanism in an exemplary network topology.

the new model. Additionally, the most representative samples from the old data are chosen to help the new model review the old knowledge.

FL cannot be efficiently applied in WSN for a number of reasons. Indeed, let us consider a WSN consisting of eight nodes as sketched in Fig. 1, and let us assume that FL is applied and that node A is the aggregation point, whereas all other nodes denoted as L1, L2, ..., L7 are the federated learners.

Models generated by the federated learners will traverse several hops before arriving at the aggregation point A. For example, the model generated by learner L7 will pass through L6 and L3 before reaching A. This will require a large number of transmissions by wireless sensor nodes. More specifically, if we consider the tree spanning all network nodes, whose root is node A, we can calculate the number of model transmissions needed at each learning round as

$$\mathcal{N} = \sum_{l=1}^{H} N_{D_l} \cdot l \tag{2}$$

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

Furthermore, it is clear that the nodes that are close to the root, A, will be involved in several relaying operations. For example in Fig. 1, node L3 is responsible for forwarding models $\mathbf{w}_3$, $\mathbf{w}_5$, $\mathbf{w}_6$, and $\mathbf{w}_7$ whereas nodes L2, L4, L5, and L7 will be responsible for the transmissions of their models only. Therefore, the communication burden on node L3 is four times the burden on nodes L2, L4, L5, and L7, which is extremely unfair and may cause the rapid exhaustion of the batteries of node L3. This is the well-known funneling effect problem which is the reason for unfairness and may result in a rapid exhaustion of batteries in nodes closer to the aggregation point A and, thus, as a consequence, can lead to a significant reduction in network lifetime.

One obvious way to overcome this problem is to exploit the multihop communication paradigm by aggregating models at intermediate nodes. In other terms, a node will wait for the models coming from all its child nodes in the routing tree. The node will aggregate them with its own model and will send only the result of such operation to its parent node. For example, L3 will wait for the models coming from L5 and L6 and will send the aggregation of $\mathbf{w}_3$, $\mathbf{w}_5$, and $\mathbf{w}_6$ to the aggregation point. Note that in this case, $\mathbf{w}_6$ is estimated as the result of the aggregation of the model specific for L6 and of model $\mathbf{w}_7$.

Such an approach would reduce the transmissions significantly. Indeed, in the scenario reported in Fig. 1, aggregation at intermediate nodes L1 and L3 allows to reduce the number of transmissions of model parameters executed at each round to 7 which implies a reduction in the overhead of approximately $(6/13) = 46\%$. Also, it increases the fairness significantly. However, it involves that the aggregated model is built in the aggregation point and then flooded into the network. Furthermore, it relies on knowledge of a route toward the aggregation point in each node, which is not always the case.

To address such issues, *gossiping* can be exploited as we explain in the following.

Gossiping is a mechanism conceived to solve the—so called—*consensus problem* by exploiting the computing resources at each node to reduce the amount of data that needs to be transmitted in the network. Therefore, gossiping can be used to save energy and communication resources, so extending network lifetime, and reducing latency [27], [28], [29].

Gossiping, thus, captures the condition where a set of network agents must achieve a shared *opinion* through exchanges of local information with neighbors. In WSN, gossiping has applications in distributed inference and detection [30], [31].

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

Early examples of such schemes are presented in [32], [33], [34], and [35]. Nodes exploit the locally observed data and collaborate with their one-hop neighbors to collectively learn a model that best fits the data collected by the entire network.

One of such schemes is proposed in [36] which is supported by theoretical results regarding its performance. More specifically, a scheme is proposed that adapts the transmission rates of individual nodes to control network density while keeping the communication time required to exchange the models below appropriate thresholds. A significant step toward the practical applicability of gossiping is achieved in [37] where the network constraints are taken into account. The authors apply gossiping to the Industrial Internet of Things (IIoT) scenario. To this aim, they consider relevant network constraints like setup, convergence speed, communication overhead, and average execution time on embedded devices.

A novel class of FL algorithms are introduced to improve convergence and a prototype implementation is presented. Similarly, a graph federated architecture is presented in [38] to overcome the computational and communication overloads in the original FL approach. More specifically, a multiserver architecture was designed in which each server has a set of clients and the servers run a consensus-type algorithm among themselves similar to our approach. However, differently from our work, in [38] both the servers and the clients are nodes that have the capability to train an ML model.

Several clustered FL schemes have also been proposed in the literature. In [39], a clustered FL approach for non-i.i.d data is proposed in which nodes are partitioned into clusters based on their objectives, i.e., the learning task. Therefore, each cluster has nodes with the same learning task and the goal is to train a model for each cluster. However, differently from [39], in our approach all the nodes do have the same objectives or learning tasks. Also, we consider a more general case where nodes have different computing and communication capabilities which is not the case in [39] and, thus, based on their capabilities, they can behave differently and execute diverse functionalities (i.e., training and/or inference). In [40], another clustered FL approach is proposed in which the clustering is done by K-means in a centralized manner based on the node's data, and a common model is trained for each cluster using data of all nodes that belong to the cluster. The nodes that converge more slowly or have little correlation with other nodes in each cluster are dropped to speed up the convergence while maintaining the accuracy of all the nodes. The performance of the proposed approach was evaluated using an NVIDIA edge testbed. However, similar to the previous clustered FL approach, the ML and communication capabilities of the users are not considered. Another clustered FL approach is proposed in [41] in which the geometric properties of the FL loss surface is exploited to group the nodes population into clusters with jointly trainable data distributions. However, in this approach, the clustering is only carried out after a few rounds of FL and the convergence has reached a stationary point. In all the above-mentioned approaches, the ML and communication capabilities of the nodes are not considered. Also, in most of the clustered FL approaches, the nodes are grouped based on their data distributions.

Differently from other state-of-the-art solutions, in our approach, the clustering strategy will consider devices' hardware features, meaning that more powerful nodes will execute resource-intensive operations and will act as Cluster Heads, while common nodes will only execute simpler operations.

Unfortunately, none of the schemes discussed so far considers that in most cases while all nodes can perform inference, only a subset of them can perform model training because of the associated hardware constraints.

Conversely, *i-WSN* League takes such constraint into account and aims at minimizing the energy consumption in resource-limited nodes as detailed in the following Section III.

III. *i-WSN* League

In this section, we introduce i-WSN League which is a methodology for supporting ML in a WSN network consisting of heterogeneous nodes. In fact, we assume that the two types of nodes are available in *i-WSN* League: 1) common nodes, that are resource constrained and, therefore, cannot execute model training and 2) head nodes, that are resource-rich devices and have the capabilities to execute all ML functions. In such a context, the objective of *i-WSN* League is to minimize the consumption of resources, and more specifically, the consumption of energy, at the common nodes. This is because the exhaustion of their batteries will result in their definitive failure which might involve the failure of the entire network.

This section is organized as follows. In Section III-A, we give an overview of the *i-WSN* League operations. In this context, we will also characterize the major features of the hardware platforms that are used for head nodes and common nodes. Then, in Section III-B, we will present the *i-WSN* League protocol in detail.

A. Overview of Operations and Characteristics of the Hardware Platforms

As discussed above, we consider a WSN consisting of a few head nodes and a majority of common nodes.

Head nodes are equipped with two wireless interfaces. One of them allows connection to a wide-area network, e.g., LoRa [42] or IEEE 802.11, the other enables short-range communications, e.g., BLE or IEEE 802.15.4. Common nodes are equipped with the short-range wireless communication interface only.

Nodes will be divided into Clusters, each of which has a *Cluster Head* and several CMs.[2] Head nodes can be Cluster Heads, common nodes are always CMs. Cluster Heads can communicate using both wireless interfaces, CMs can communicate using the short-range wireless communication interface, only. Thus, head nodes that are CMs will maintain their wide-area network interface turned off.

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

Cluster Heads execute the model training using their own data. The *fitness* of the model will be evaluated through an appropriate *loss* function.[3] Each Cluster Head broadcasts the value of its loss function throughout the CH-network after it calculates it at the end of every training execution. The Cluster Head that attains the lowest loss will transmit its model parameters to its one-hop neighbors in the CH-network using the long-range wireless interface.

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

$$\mathbf{w}_k = \alpha \cdot \mathbf{w}_j + (1 - \alpha) \cdot \mathbf{w}_k \tag{3}$$

where α is a weight parameter that we set larger than 0.5 as detailed later. After updating their models, the Cluster Heads will train the model obtained by applying (3) using the data they have available locally. Note that, choosing a value of α different from 1, implies that we both avoid fluctuations in

[2]In the remainder of this article, we assume that a CM can belong to one cluster only, i.e., it cannot be shared by multiple clusters. This is to avoid the risk of overfitting certain data patterns since several models generated by different Cluster Heads and propagated to other Cluster Heads will represent the same set of data generated by the few shared CMs. As a consequence, the general model will give too much importance to the data generated by such shared CMs

[3]Several loss functions have been proposed for different application scenarios. Interested readers can refer to the overview provided in [43].

weights and we also "store" a memory of the data set in the node receiving the model.

In some cases, CMs are characterized by very simple hardware, and, thus, complex tasks cannot be executed on them. Accordingly, note that, in case of particular limitations in the device performing as CMs, the generic *j*th Cluster Head will create a compressed version of its model, $\widehat{\mathbf{w}}_j$, in such a way that it can be used for inference by common nodes as well. How the compressed model, $\widehat{\mathbf{w}}_j$ can be used for inference by common nodes is outside the scope of this article as it depends on the tool utilized for model compression. We refer interested readers to the Tensorflow Lite documentation [44] which specifies the workflow to be used.

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

This may raise some privacy concerns. However, we observe that as follows.

1) The data is shared by the CM with its Cluster Head only which is responsible for storing and processing it, i.e., it is not disseminated further.
2) As shown in the following Section IV, the number of times these events occur is indeed small and can be tuned by appropriately setting the above threshold value. The exact setting is a matter of tradeoff between minimizing the model loss, i.e., increasing the fidelity of the model, and minimizing the privacy concerns.

We denote the above sequence of actions as *protocol iteration* or *iteration*, in short.

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

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

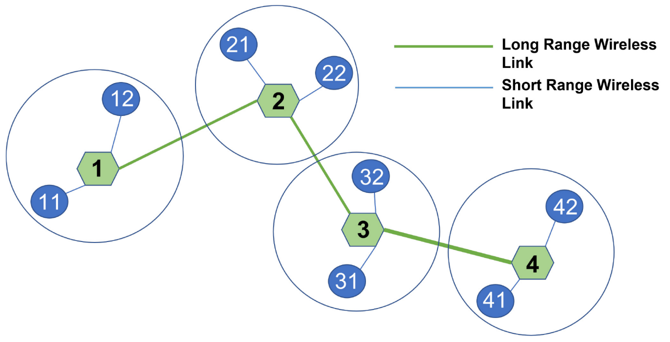

Fig. 2. Clustering scenario.

B. Protocol Details

In this section, we present the details of the *i-WSN* League protocol and, for the sake of clarity, describe its operations in the simple scenario depicted in Fig. 2. This comprises eight common nodes grouped into four Clusters. Nodes 1, 2, 3, and 4 are Cluster Heads[4] equipped with two wireless interfaces. Accordingly, the CH-network consists of four nodes connected according to a linear topology. All the other nodes are CMs, each of which belongs to one Cluster only. Observe, that communication in each Cluster can happen in a multihop manner when needed. Algorithm 1 represents the pseudocode for the protocol run by the generic Cluster Head *CH*. The resulting actions executed by the Cluster Heads are represented in Fig. 3.

At the startup sketched in lines 1–6 of Algorithm 1, Cluster Head *CH* initializes the boost factor B_{CH} to 1. Furthermore, the model is trained using the local data X_{CH} and starting from random initial conditions, RND. The training operation is executed for a given number of epochs. The *scaled loss* parameter, denoted as, SL_{CH} is calculated from the training loss and the boost factor as in (4), i.e.,

$$SL_{CH} = F_{CH}(\mathbf{w}_{CH})/B_{CH} \tag{4}$$

where $F_{CH}(\mathbf{w}_{CH})$ is the loss of the Cluster Head, CH, and B_{CH} is its current boost factor. Observe that the scaled loss is supposed to decrease for Cluster Heads that have the good fitting models (i.e., low $F_{CH}(\cdot)$) and that did not transmit their model parameters recently (i.e., high B_{CH}). By using the boost factor and normalizing the loss function using this factor allows to guarantee more fairness inside the system by fostering a kind of turnover among nodes that disseminate their models. Also, choosing better fitting nodes as those disseminating the models has the advantage of fostering, as we will see in the following, faster convergence of the model. For example, in Fig. 3, the loss values for the four Cluster Heads at the end of the initialization phase are $F_1(\mathbf{w}_1) = 69.11$, $F_2(\mathbf{w}_2) = 26.72$, $F_3(\mathbf{w}_3) = 66.32$, and $F_4(\mathbf{w}_4) = 98.75$. Therefore, given that the boost factors for all of them is equal to 1, the corresponding scaled losses are $SL_1(\mathbf{w}_1) = 69.11$, $SL_2(\mathbf{w}_2) = 26.72$, $SL_3(\mathbf{w}_3) = 66.32$, and $SL_4(\mathbf{w}_4) = 98.75$.

[4]We will denote the Cluster with the identifier of the corresponding Cluster Head, therefore, if we say Cluster 1, we mean the Cluster for which node 1 is the Cluster Head.

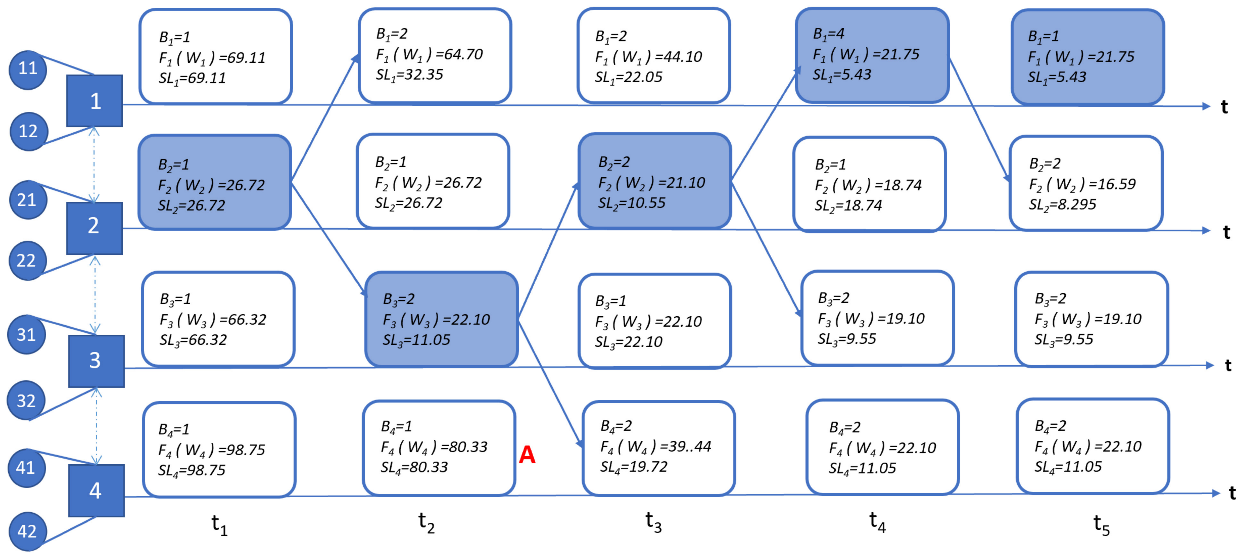
11
12
21
22
31
32
41
42
1
2
3
4
B1=1 F1(W1)=69.11 SL1=69.11
B2=1 F2(W2)=26.72 SL2=26.72
B3=1 F3(W3)=66.32 SL3=66.32
B4=1 F4(W4)=98.75 SL4=98.75
B1=2 F1(W1)=64.70 SL1=32.35
B2=1 F2(W2)=26.72 SL2=26.72
B3=2 F3(W3)=22.10 SL3=11.05
B4=1 F4(W4)=80.33 SL4=80.33
A
B1=2 F1(W1)=44.10 SL1=22.05
B2=2 F2(W2)=21.10 SL2=10.55
B3=1 F3(W3)=22.10 SL3=22.10
B4=2 F4(W4)=39..44 SL4=19.72
B1=4 F1(W1)=21.75 SL1=5.43
B2=1 F2(W2)=18.74 SL2=18.74
B3=2 F3(W3)=19.10 SL3=9.55
B4=2 F4(W4)=22.10 SL4=11.05
B1=1 F1(W1)=21.75 SL1=5.43
B2=2 F2(W2)=16.59 SL2=8.295
B3=2 F3(W3)=19.10 SL3=9.55
B4=2 F4(W4)=22.10 SL4=11.05
t
t
t
t
t1
t2
t3
t4
t5

Fig. 3. *i-WSN* league protocol in action (cluster heads).

At the end of the initialization phase, the Cluster Heads broadcast their model parameters to their own CMs (line 5). Also, the Cluster Heads broadcast their scaled loss in the Cluster Head network (line 6). In this way, Cluster Heads identify which of them has the lowest scaled loss and, thus, who will broadcast its model parameters.

When the initialization phase is completed, Cluster Heads execute the regular operations sketched in lines 7–30 of Algorithm 1, which are event-based. More specifically, three types of events can occur.

1) *Broadcast:* In this event, the Cluster Head broadcasts its model parameters, i.e., the Cluster Head with the lowest scaled loss broadcasts its model parameters to its CMs and its neighboring Cluster Heads. In this case, the node executes the operations sketched in lines 10–18 of Algorithm 1.
2) *Receive Model Parameters:* In this event, the Cluster Head receives the model parameters $\mathbf{w}_k$. When the model parameters are received, the Cluster Head executes the operations sketched in lines 19–27 of Algorithm 1, including retraining of the model.
3) *Receive Data Chunk:* In this event, the Cluster Head receives a chunk of data from one of its CMs for retraining. The Cluster Head will execute the operations sketched in lines 28–31 of Algorithm 1.

At the end of the initialization phase t_1 and at the end of each iteration, each Cluster Head broadcasts its scaled loss to all other Cluster Heads. Thus, each Cluster Head has the scaled losses of other Cluster Heads to make a comparison and realize if it has the lowest scaled loss. If this is the case, it will perform the operations reported in lines 10–18 of Algorithm 1. The Cluster Head will broadcast the model parameters $\mathbf{w}_{CH}$ to the one-hop neighbors in the CH-network and compresses the model to broadcast the compressed model parameters $\widehat{\mathbf{w}}_{CH}$ to its own CMs. In our experiments, the Tensor-flow model is compressed into a Tensor-flow lite model.

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

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

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

As we explain later it might happen that the Cluster Head receives a chunk of data by one of its CMs, say j. As reported in lines 28–31 of Algorithm 1, where $\mathbb{DA}_j$ represents the received chunk of data, in this case, the Cluster Head trains the model using the received data along with the rest of the data locally available. Then, the new model is sent back to the CM j.

Algorithm 2 illustrates the functions of the generic CM when it receives the model parameters from its Cluster Head. Before updating its model, the node evaluates the loss which it would obtain using the model $\widehat{\mathbf{w}}_{CH}$ transmitted by the Cluster Head, $F_j(\widehat{\mathbf{w}}_{CH})$ and compares its own loss $F_j(\widehat{\mathbf{w}}_j)$ with the

Algorithm 1 *i-WSN* Protocol-*Cluster Head*

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

value coming from using $\widehat{\mathbf{w}}_{CH}$. If the difference between the two values is larger than a given threshold σ_{TH}, it means that the data set available at the Cluster Head is not representative of the data available at the CM. Accordingly, the CM sends a chunk of its data to the Cluster Head. As explained earlier, the Cluster Head will use such data for training its model. This is the reason why there is a change in the loss in the Cluster Head at the second iteration though it has not received the model parameters (point A in Fig. 3). Since the Cluster Head has received the data chunk and trained on it, there is a change in the loss from 98.75 to 80.33 in Fig. 4 where we detail what happens in Fig. 3 at the second iteration in Cluster Head 4.

IV. Performance Evaluation

In this section, we assess the performance of *i-WSN* League by analyzing its behavior in a case study. In such context, we will also analyze the impact of some scenarios and protocol parameters.

Algorithm 2 *i-WSN* Protocol-*CM*

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

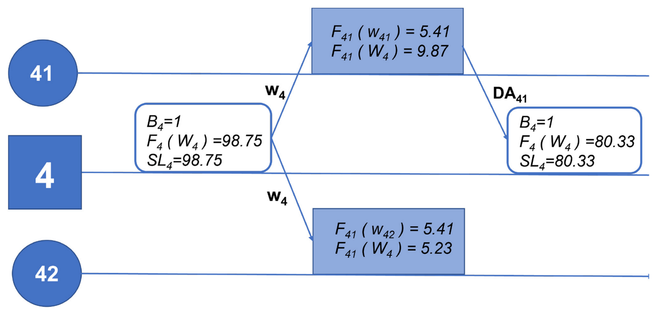

Fig. 4. *i-WSN* league protocol in action (inside Cluster 4).

Accordingly, in the following section, i.e., Section IV-A, we describe the scenario and the data set considered in our experiments. Then, in Section IV-B, we present and discuss the numerical results.

A. Scenario

We consider a wireless sensor network consisting of sensor nodes collecting environmental parameter to perform anomaly detection. We use a well-known data set containing the environmental parameters measured in 2017 by a wireless sensor network of 56 nodes deployed in Krakov. The data are IID and the positions of nodes are available in the data set as well. The data set is available in Kaggle.[5]

Since the sensors collect one value for each parameter every hour, the data set for one month (i.e., 30 days) considered for the kth sensor should consist of $n_k = 24 \cdot 30 = 720$ entries of seven values. These seven values represent the day, time, temperature, humidity, and PM1, PM2.5, and PM10 parameters measured in July 1–30, 2017. Unfortunately, certain values for 31 sensors out of the 56 are missing, and, therefore, we will consider only the 25 sensors for which we have all the values available.

The topology of the sensor network is shown in Fig. 5. The above topology has been constructed considering the real position of the sensors and assuming that each of them is equipped with a wireless interface giving a radio coverage of 4 km.

For the training in Cluster Heads, we consider the data from the first 25 days and we consider the data from the last five days for the inference.

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

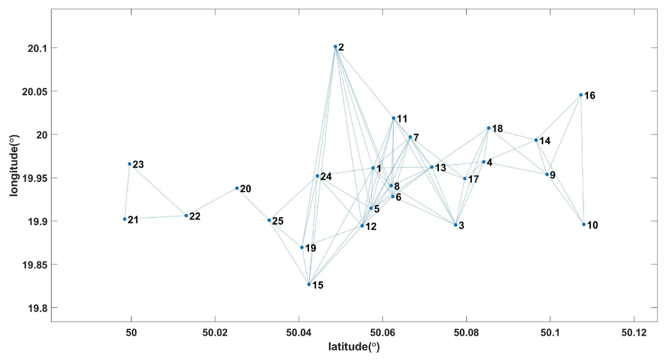

Fig. 5. Wireless network scenario for the simulation.

The training data comprises of 25 (days) * 24 (h) = 600 entries of five parameters. The parameters are temperature, humidity, PM1, PM2.5, and PM10. More specifically, one training sample consists of five values and each sensor has 600 samples. For the training of the auto-encoder model, the batch size was varied to generate batches of different sizes of the input data.

Similarly, in the case of the CMs, the data from the first 25 days is split in chunks, while the last five days' data is utilized for the inference.

For our experiments, we consider four cases with different clustering strategies which result in the configurations shown in Fig. 6.

1) *Case 1:* The network in this case has five Cluster Heads with an equal number of CMs. Therefore, each cluster Head has four CMs. The Cluster Heads are nodes 1, 3, 14, 19, and 22.
2) *Case 2:* The network in this case has eight Cluster Heads. Therefore, the sizes of the clusters are smaller in this case as compared to Case 1. The Cluster Heads are nodes 1, 4, 6, 7, 9, 15, 21, and 25.
3) *Case 3:* The clustering, in this case, is designed to have five Cluster Heads in the network but each exhibiting a different number of CMs. More specifically, the Cluster Heads are nodes 1 (9 CMs), 10 (4 CMs), 15 (4 CMs), 18 (3 CMs), and 22 (4 CMs).
4) *Case 4:* In this case, the network is divided into two large clusters only, and the Cluster Heads are nodes 4 and 20.

In all the above cases, there are no nodes that belong to more than one cluster. Note that Cases 1 and 3 are similar as the number of Clusters considered is 5 in both cases. However, the number of CMs in Case 1 is equal to 4 in all clusters. Instead, in Case 3 there are different numbers of CMs. Case 3 has been considered for analyzing the impact on i-WSN performance of the distribution of nodes in the different clusters. In Cases 2 and 4, there are different number of Clusters, i.e., 8 and 2, respectively, and, therefore, have been considered to analyze the impact of the number and size of Clusters on performance.

i-WSN League can be applied whatever is the ML approach utilized. Nevertheless, in our experiments, we assume that each Cluster Head trains an autoencoder [45] which is highly utilized for anomaly detection [46]. Also, we assume that the all nodes can perform inference with a trained model.

An autoencoder is a type of artificial neural network that operates in an unsupervised way to learn efficient data encoding. In Fig. 7, we present the architecture of an autoencoder.

Specifically, backpropagation is used so that the values given as output by the neural network, denoted as X' in Fig. 7, are as close as possible to the input ones, denoted as X in the same figure.

Besides learning an efficient representation of data, autoencoders can denoise and decorrelate data. Furthermore, autoencoders can be used for anomaly detection as well. In fact a large difference between X and X' means that the current input data is significantly different from the data used to train the autoencoder, which is the evidence of an anomaly.

Since our focus is not on the specific ML approach, in our experiments, we have employed the simplest type of autoencoder which consists of one-hidden layer, only.

Accordingly, in our experiments, the autoencoder consists of a three-layer neural network, thus, having one input, one-hidden and one-output layer.

In the encoding process, the autoencoder first converts the input vector $\mathbf{X}$ into a hidden representation $\mathbf{Z}$ using weights $\mathbf{w}'$. During the decoding, instead, the autoencoder maps the hidden representation back to obtain the original format and obtains $\mathbf{X}'$ through other weights $\mathbf{w}''$. The model parameter optimization is aimed at minimizing a loss measure which is proportional to the average reconstruction error, i.e., the difference between $\mathbf{X}'$ and $\mathbf{X}$. More specifically, the loss is calculated as the mean square error (MSE) [47].

Concerning the specific values, we focus on the values of temperature, humidity, PM1, PM2.5, and PM10; therefore, the input size of the autoencoder is 5. The intermediate size, which is also known as compressed dimension is equal to 3.

Therefore, the uth entry in the data set of the kth sensor, denoted as $\mathbf{X}_{k,u}$, is $\mathbf{X}_{k,u} = (X^{(0)}_{k,u}, X^{(1)}_{k,u}, \ldots, X^{(4)}_{k,u})$.

B. Numerical Results

In this section, we report the numerical results obtained in the scenario described in the previous Section IV-A by applying the *i-WSN* League approach in all four Cases. We carried out a large simulation campaign to evaluate the relevant performance parameters which we have selected, i.e., the average loss of any node, the number of times a Cluster Head is trained, and the number of times CMs have sent their data to their Cluster Heads.

More specifically, ten simulations in each case were executed for different values of α (i.e., $\alpha = 0.5$, 0.7, 0.9, and 1) in (3), and a different number of *epochs* (i.e., 10, 15, and 20), which denote the number of passes that the optimization algorithm utilized to train the autoencoder will work through the entire local data set. Note that, instead, an iteration denotes the execution of the complete algorithm, that is starting from the broadcast of the model parameters by the Cluster Heads to the operations performed by the CMs as discussed in the previous section.

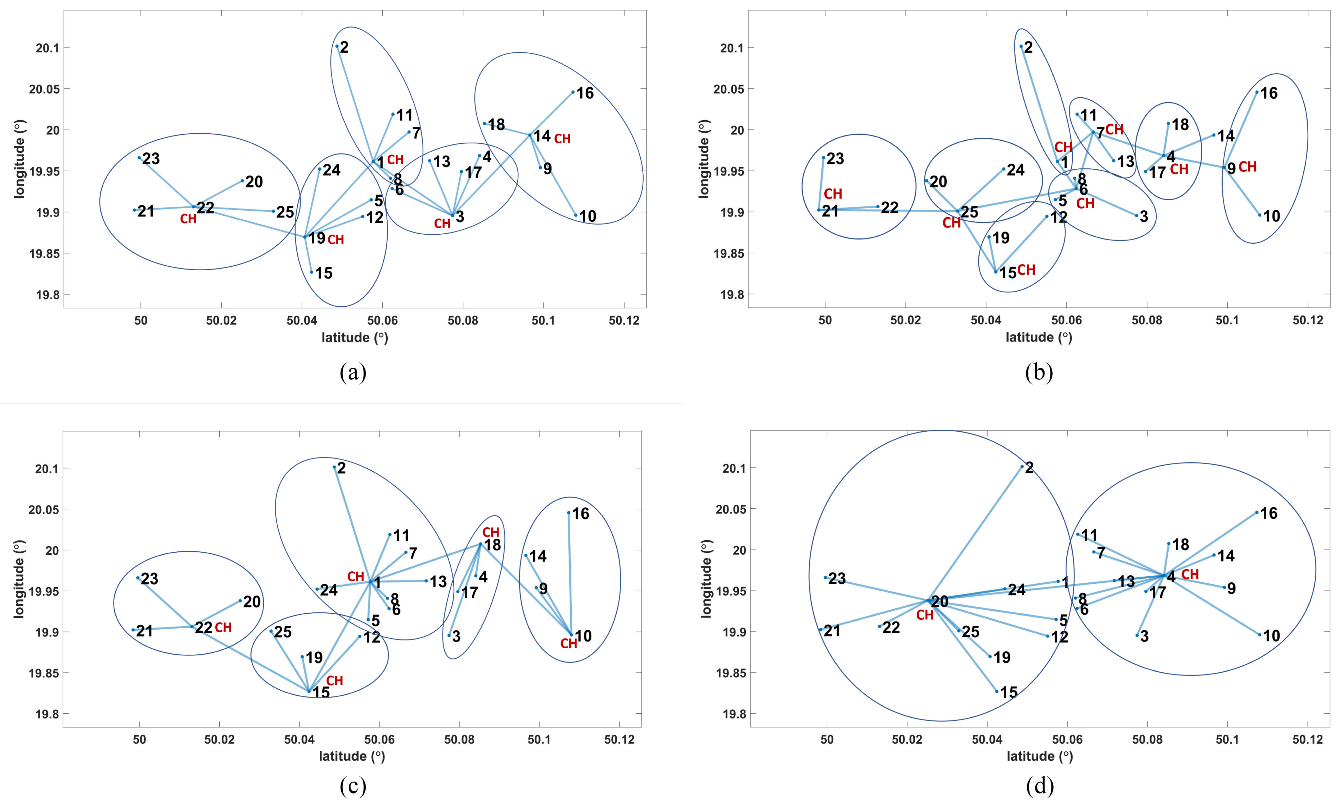

Fig. 6. Clustering cases considered in the experiments. (a) Clustering topology 1. (b) Clustering topology 2. (c) Clustering topology 3. (d) Clustering topology 4.

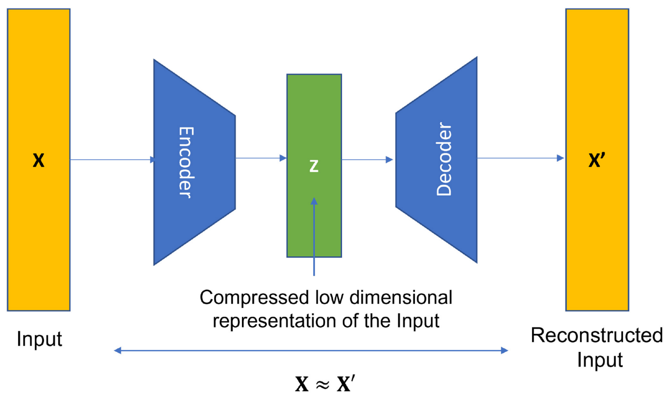
X
Encoder
z
Decoder
X′
Input
Compressed low dimensional representation of the Input
Reconstructed Input
$\mathbf{X} \approx \mathbf{X}'$

Fig. 7. Model of an autoencoder.

We explicitly observe that in *i-WSN* League each iteration involves the broadcast of the model parameters of one-Cluster Head which is the one exhibiting the best fitness performance in terms of loss function.

In Fig. 8, we report the average loss with respect to the number of iterations for different values of epochs in each case and $\alpha = 0.7$. We observe that, when the number of iterations increases the average loss significantly decreases in all cases. However, interesting observations can be drawn regarding the comparison between the performance of Cases 1 and 3 with the performance of Case 2 and Case 4. In fact, when the number of clusters is small (i.e., Case 4) convergence is fast because consensus must be reached between a small number of Cluster Heads, however, the loss obtained is high because the model of each Cluster Head must be representative of a large number of CMs. This is reflected in Fig. 8 where the performance of Case 4 are the worst. In the same figure, however, we can observe that a higher number of clusters does not always result in better performance. In fact, in Case 2 the average loss reaches a smaller number than in Case 4 but convergence is slower than in Cases 1 and 3. This is because more iterations are required to reach a shared model between all Cluster Heads. Nevertheless, in Fig. 12, we will observe that in Case 2 the number of transmissions by CMs is minimal which represents the key goal in our scenario.

In Fig. 9, we report the average loss with respect to the number of iterations for different values of α and 15 epochs. We observe that higher values of α cause convergence to be, in general, faster in all cases. In fact, the parameter α is the weight that controls the model parameters updates as given in (3). More specifically, if α is higher than 0.5, then in the aggregation the node which receives the model by another one will give more importance to the received model than to its own. The objective of the gossiping in our approach is to spread the best fitting models throughout the network. Therefore, more importance is given to the received models, and, thus, high values of α are used to spread

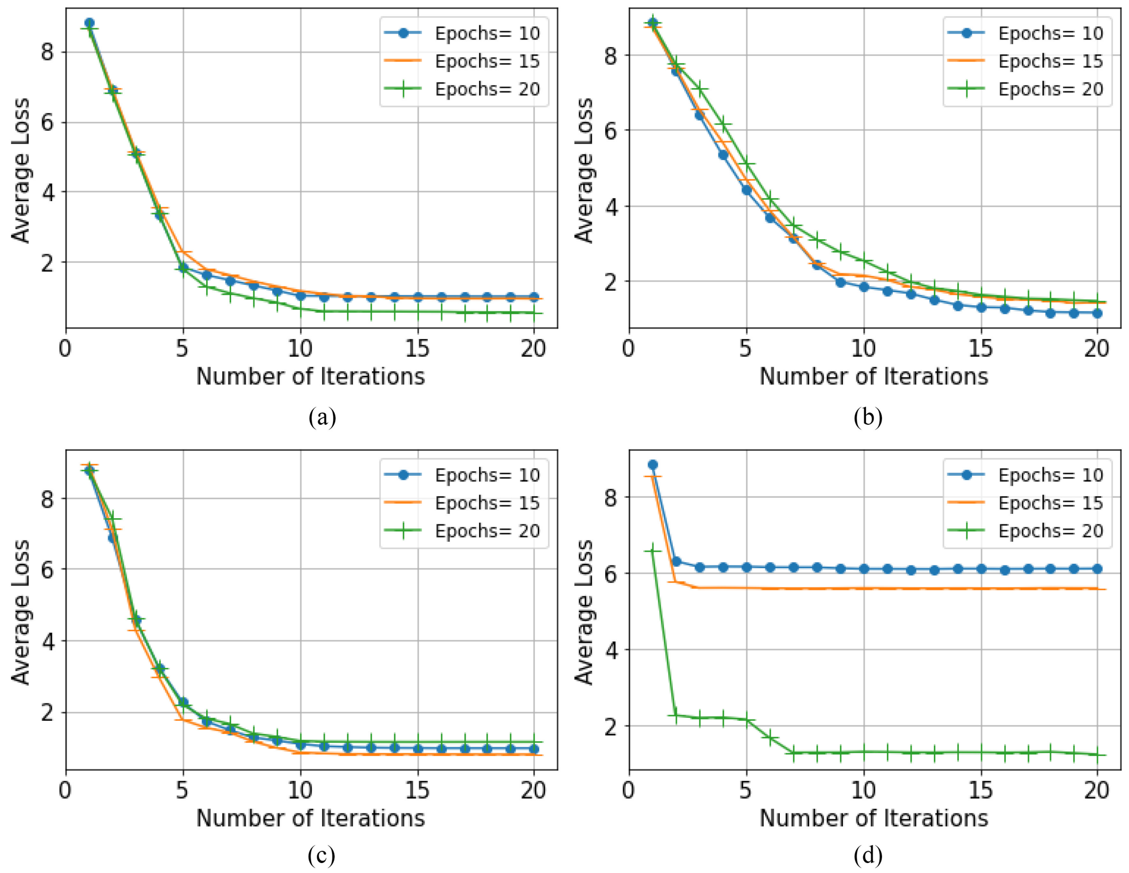

Fig. 8. Average loss versus the number of iterations for different values of epochs. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4.

the fitting models quickly in the network so achieving faster convergence.

In Fig. 10, Note that for $\alpha = 0.7$ still the slope of the average loss convergence is slower for the Case 1 as compared to Case 3 because of the more balanced distribution of CMs in Case 1 as compared to Case 3.

In Fig. 11, we report the average number of times a Cluster Head gets trained after the 20th iterations versus the number of epochs and when α is 0.9. Note that, a Cluster Head does not only perform training at the beginning of an iteration but it also retrains its model exploiting the data chunks it receives from its CMs. Observe that the average number of times a Cluster Head gets trained is the highest in Case 4 and lowest in Case 2 and it is quite similar in Cases 1 and 3. It is the highest in Case 4 because the Cluster Heads have to train on the data chunk sent by its CMs more times because there are only 2 Cluster Heads and, thus, it is more likely that their losses will be larger. Therefore, the two Cluster Heads which have more than ten CMs each will go through a larger number of training cycles.

The training counts in Cases 1 and 3 are approximately the same because the number of clusters is equal. Thus, the training load of the Cluster Heads is not as high as in Case 4 and not as low as in Case 2. Similarly, Case 2 has the lowest count because each Cluster Head has approximately two CMs and, therefore, the training load will be the lowest.

Finally, observe that the impact of the number of epochs is not significant.

Analogously, in Fig. 12, we report the average number of times CMs have transmitted their data chunks to the corresponding Cluster Heads when α is 0.9. Note that in i-WSN League scenarios this performance metric is extremely important because it gives a measure of the amount of resources consumed by common nodes, which are resource constrained. Observations can be drawn similar to those regarding Fig. 11.

This result is not surprising. In fact, in our setting, the evaluation of the model parameters performed by the Cluster Head can be seen as a consensus problem within the cluster as well. Therefore, when the number of clusters increases the number of CM in each cluster decreases. Therefore, reaching the consensus on the model parameters within the cluster becomes easier. Accordingly, we have added the following note in Section IV-B.

In Fig. 13, we report the average number of times each Cluster Head has been trained in Case 1 when the number of epochs is 15 and the value of α is 0.9. The Cluster Heads in Case 1 are nodes 1, 3, 14, 19, and 22. Observe that all Cluster Heads are trained for the same number of times which is the evidence of fairness in processing and communication load. Fairness is an important factor in WSN scenarios and maintaining good fairness in the network will lead to balanced energy consumption and, thus, a longer network lifetime.

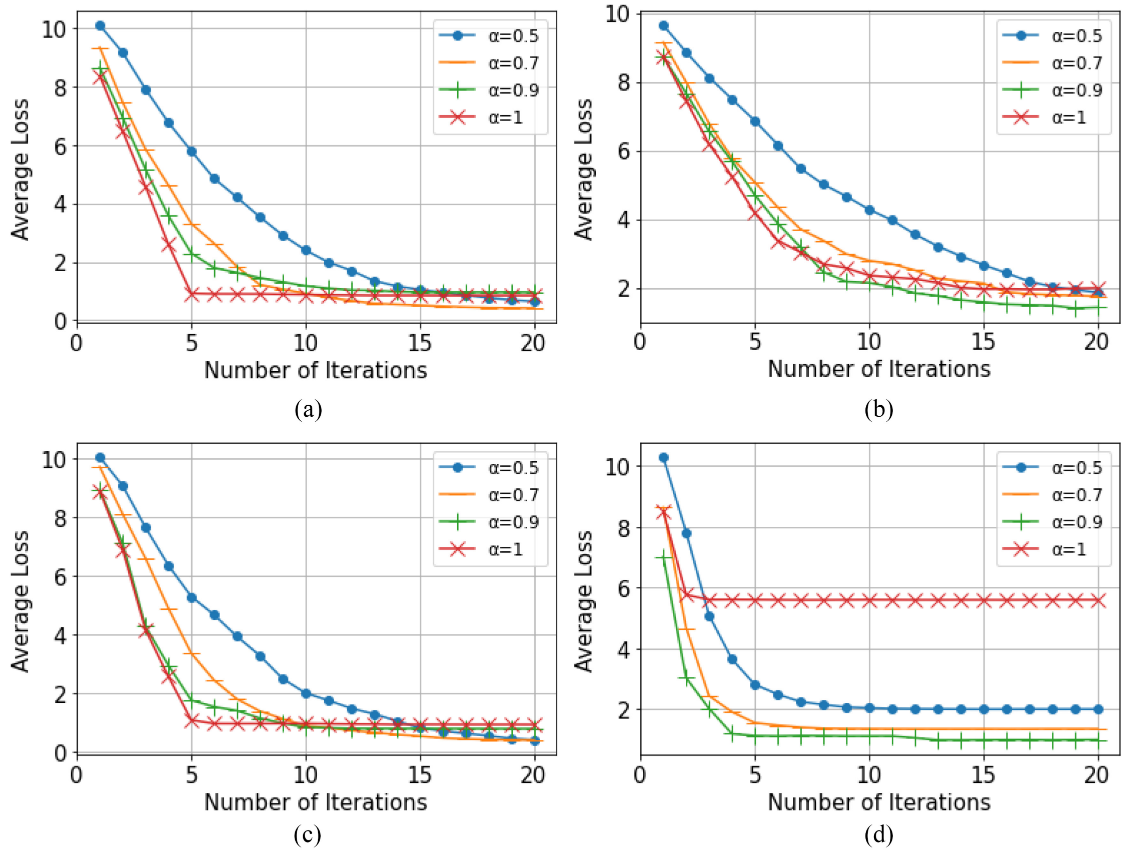

Fig. 9. Average loss versus the number of iterations for different values of α. (a) Case 1. (b) Case 2. (c) Case 3. (d) Case 4.

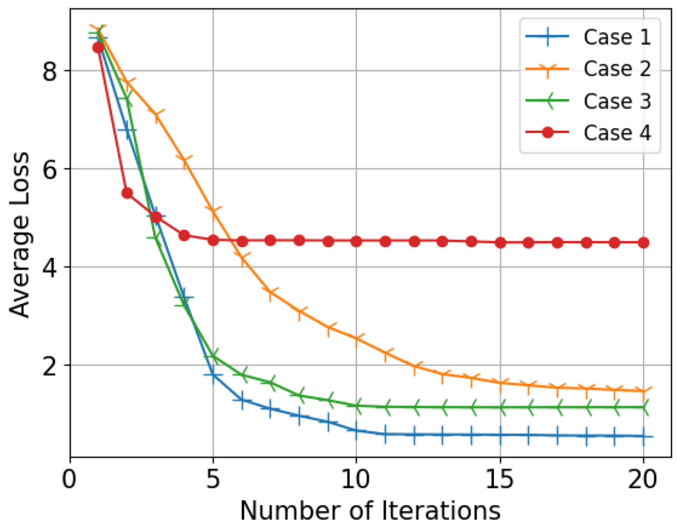

Fig. 10. Average loss versus the number of iterations in each case.

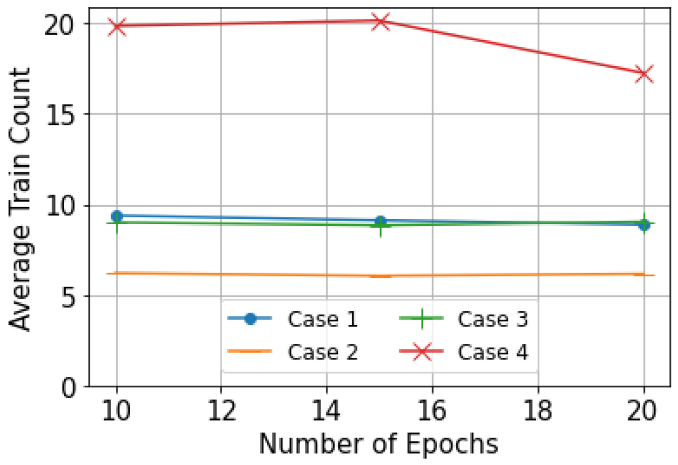

Fig. 11. Average train count versus the number of epochs.

Similar observations can be made regarding Fig. 14 in which we show the average number of times each CM has sent data chunks to its Cluster Head in Case 1, when the number of epochs is 15 and the value of α is 0.9. Approximately, all nodes transmit data chunks the same number of times which, on average, is lower than 1. This means that, after a few data chunk transmissions from the CMs, the Cluster Heads have succeeded in training a fitting model for their clusters.

Concerning the policy for the selection of the data chunks that CMs will transmit, we applied an approach in which these are chosen randomly in the data set of the CM and after they are transmitted they are marked so that they will not be chosen anymore in the following rounds. However, note that other policies for data chunks selection can be applied and that will not limit the effectiveness of the i-WSN solution.

To assess fairness, we calculated the Jain's Fairness index [48] regarding the number of times the Cluster Heads have been trained in all the cases.

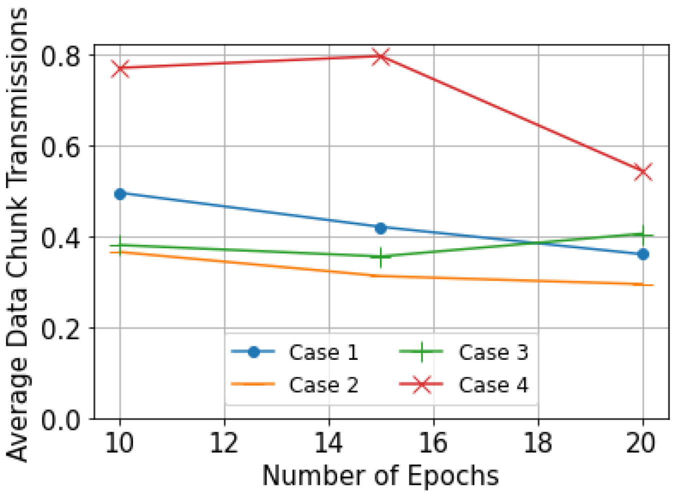

Fig. 12. Average data chunk transmissions versus the number of epochs.

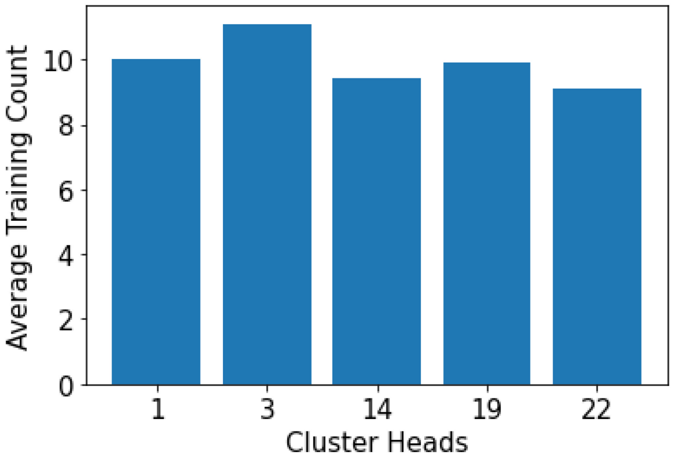

Fig. 13. Average number of times each cluster head has been trained.

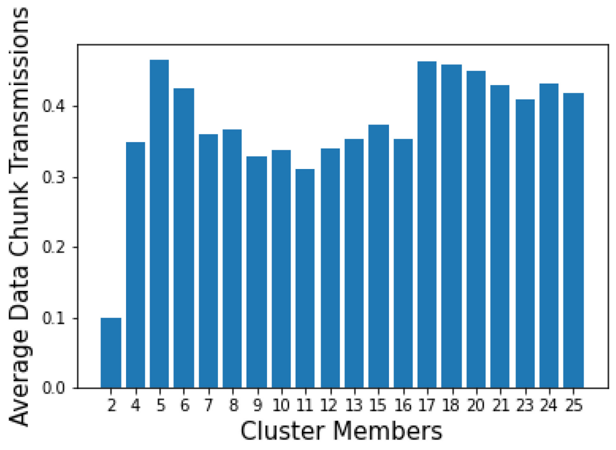

Fig. 14. Average number of times each CM has transmitted data chunks.

The Jain's Fairness index is calculated as given below

$$\mathcal{F}(x_1, x_2, \ldots, x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2} \quad (5)$$

where n represents the number of Cluster Heads (in our case $n = 25$) and x_n represents the number of times the nth Cluster Head is trained.

TABLE I
FAIRNESS INDEX REGARDING THE NUMBER OF TIMES CLUSTER HEADS HAVE BEEN TRAINED FOR DIFFERENT VALUES OF α AND EPOCHS IN CASE 1 AND CASE 2

α	Epochs	Case 1	Case 2	Case 3	Case 4
0.5	10	0.97660	0.90735	0.90275	0.98857
0.5	15	0.96661	0.92125	0.89931	0.98497
0.5	20	0.97696	0.91307	0.89592	0.98563
0.7	10	0.95835	0.91007	0.91950	0.99206
0.7	15	0.96343	0.91550	0.92598	0.98348
0.7	20	0.96176	0.88494	0.91894	0.99308
0.9	10	0.93305	0.90062	0.89575	0.94063
0.9	15	0.90417	0.89184	0.91436	0.91169
0.9	20	0.90105	0.89360	0.89167	0.94064
1	10	0.92991	0.89283	0.87492	0.95675
1	15	0.89929	0.89084	0.93640	0.91654
1	20	0.91732	0.88954	0.92499	0.92258

In Table I, we report Jain's Fairness index values for different values of α and the number of epochs in all the cases. We observe high fairness values in all the cases, i.e., fairness index closer to 1, which illustrates balanced energy consumption in all clusters. Observe that the highest fairness index values have been obtained in Case 4. However, this is due to the fact that there are only two clusters in Case 4. Cases 1, 2, and 3 show similar fairness index values. We also observed that the number of epochs and the α parameter value do not have a significant impact on fairness.

V. CONCLUSION

In this article, we have presented the *i-WSN* League framework which involves the definition of a specific architecture of the wireless sensor networks nodes besides the introduction of a protocol to support distributed ML in the wireless sensor networks. In the proposed framework, we have considered a heterogeneous network where all nodes can execute inference, whereas only some of them, called Cluster Heads, have enough resources to execute the model training. Accordingly, nodes are divided into Clusters, one for each Cluster Head, and the Cluster Head broadcasts the trained model to the other nodes of the Cluster, called CMs. The proposed protocol aims at minimizing the consumption of resources of CMs, even if some extra effort is required at the Cluster Head. Performance results, achieved by considering autoencoders as an example of an ML scenario, demonstrate that the proposed scheme is effective and is characterized by high fairness.

REFERENCES

[1] M. Rapp, R. Khalili, and J. Henkel, "Distributed learning on heterogeneous resource-constrained devices," 2020, *arXiv:2006.05403*.
[2] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 1996–2018, 4th Quart., 2014.
[3] D. Peteiro-Barral and B. Guijarro-Berdiñas, "A survey of methods for distributed machine learning," *Prog. Artif. Intell.*, vol. 2, no. 1, pp. 1–11, 2013.
[4] J. B. Predd, S. B. Kulkarni, and H. V. Poor, "Distributed learning in wireless sensor networks," *IEEE Signal Process. Mag.*, vol. 23, no. 4, pp. 56–69, Jul. 2006.

[5] A. Danaee, R. C. de Lamare, and V. H. Nascimento, "Energy-efficient distributed learning with coarsely quantized signals," *IEEE Signal Process. Lett.*, vol. 28, pp. 329–333, Jan. 2021.

[6] M. Carpentiero, V. Matta, and A. H. Sayed, "Distributed adaptive learning under communication constraints," 2021, *arXiv:2112.02129*.

[7] A. H. Sayed, S.-Y. Tu, J. Chen, X. Zhao, and Z. J. Towfic, "Diffusion strategies for adaptation and learning over networks: An examination of distributed strategies and network behavior," *IEEE Signal Process. Mag.*, vol. 30, no. 3, pp. 155–171, May 2013.

[8] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-effective federated learning design," 2020, *arXiv 2012.08336*.

[9] S. Niknam, H. S. Dhillon, and J. H. Reed, "Federated learning for wireless communications: Motivation, opportunities, and challenges," *IEEE Commun. Mag.*, vol. 58, no. 6, pp. 46–51, Jun. 2020.

[10] "AI.Google. under the hood of the pixel 2: How AI is supercharging hardware." 2018. [Online]. Available: https://ai.google/stories/ai-in-hardware/

[11] "Google. Your chats stay private while messages improves suggestions." Accessed: Jan. 2023. [Online]. Available: https://support.google.com/messages/answer/9327902

[12] "Apple. designing for privacy (video and slide deck). apple WWDC." 2019. [Online]. Available: https://developer.apple.com/videos/play/wwdc2019/708

[13] "W.de Brouwer. The federated future is ready for shipping." Mar. 2019. [Online]. Available: https://doc.ai/blog/federated-future-ready-shipping/

[14] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *Proc. ICASSP*, 2019, pp. 6341–6345.

[15] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist. (AISTATS)*, Fort Lauderdale, Florida, 2017, pp. 1273–1282.

[16] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1709–1720.

[17] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified SGD with memory," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 4452–4463.

[18] A. Koloskova, S. U. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3478–3487.

[19] R. Nassif, S. Vlaski, M. Antonini, M. Carpentiero, V. Matta, and A. H. Sayed, "Finite bit quantization for decentralized learning under subspace constraints," in *Proc. 30th Eur. Signal Process. Conf. (EUSIPCO)*, 2022, pp. 1851–1855.

[20] M. Carpentiero, V. Matta, and A. H. Sayed, "Adaptive diffusion with compressed communication," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2022, pp. 5672–5676.

[21] N. Michelusi, G. Scutari, and C.-S. Lee, "Finite-bit quantization for distributed algorithms with linear convergence," *IEEE Trans. Inf. Theory*, vol. 68, no. 11, pp. 7254–7280, Nov. 2022.

[22] J. Kirkpatrick et al., "Overcoming catastrophic forgetting in neural networks," in *Proc. Nat. Acad. Sci.*, vol. 114, no. 13, pp. 3521–3526, 2017.

[23] R. Kemker, M. McClure, A. Abitino, T. L. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, 2018, pp. 3390–3398.

[24] E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, and V. Shmatikov, "How to backdoor federated learning," in *Proc. Int. Conf. Artif. Intell. Statist. (PMLR)*, 2020, pp. 2938–2948.

[25] G. Wei and X. Li, "Knowledge lock: Overcoming catastrophic forgetting in federated learning," in *Proc. Pacific-Asia Conf. Knowl. Discov. Data Mining*, 2022, pp. 601–612.

[26] K. Guo, T. Chen, S. Ren, N. Li, M. Hu, and J. Kang, "Federated learning empowered real-time medical data processing method for smart healthcare," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, early access, Jun. 23, 2022, doi: 10.1109/TCBB.2022.3185395.

[27] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proc. IEEE*, vol. 98, no. 11, pp. 1847–1864, Nov. 2010.

[28] D. Shah., "Gossip algorithms," *Found. Trends®Netw.*, vol. 3, no. 1, pp. 1–125, 2009.

[29] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2508–2530, Jun. 2006.

[30] S. Kar and J. M. F. Moura, "Consensus based detection in sensor networks: Topology optimization under practical constraints," presented at the Workshop Inf. Theory Sensor Netw., Santa Fe, NM, USA, Jun. 2007.

[31] V. Saligrama, M. Alanyali, and O. Savas, "Distributed detection in sensor networks with packet losses and finite capacity links," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4118–4132, Nov. 2006.

[32] M. Blot, D. Picard, M. Cord, and N. Thome, "Gossip training for deep learning," 2016, *arXiv 1611.09726*.

[33] J. Daily, A. Vishnu, C. Siegel, T. Warfel, and V. Amatya, "GossipGraD: Scalable deep learning using Gossip communication based asynchronous gradient descent," 2018, *arXiv 1803.05880*.

[34] J. S. Mertens, L. Galluccio, and G. Morabito, "Federated learning through model gossiping in wireless sensor networks," in *Proc. IEEE Int. Black Sea Conf. Commun. Netw. (BlackSeaCom)*, 2021, pp. 1–6.

[35] J. S. Mertens, L. Galluccio, and G. Morabito, "MGM-4-FL: Combining federated learning and model gossiping in WSNs," *Elsevier Comput. Netw.*, vol. 214, Sep. 2022, Art. no. 109144.

[36] A. Lalitha, O. C. Kilinc, T. Javidi, and F. Koushanfar, "Peer-to-peer federated learning on graphs," 2019, *arXiv 1901.11173*.

[37] S. Savazzi, M. Nicoli, and V. Rampa, "Federated learning with cooperating devices: A consensus approach for massive IoT networks," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4641–4654, May 2020.

[38] E. Rizk and A. H. Sayed, "A graph federated architecture with privacy preserving learning," in *Proc. IEEE 22nd Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, 2021, pp. 131–135.

[39] A. Ghosh, J. Chung, D. Yin, and K. Ramchandran, "An efficient framework for clustered federated learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 19586–19597.

[40] X. Ouyang, Z. Xie, J. Zhou, J. Huang, and G. Xing, "ClusterFL: A similarity-aware federated learning system for human activity recognition," in *Proc. 19th Annu. Int. Conf. Mobile Syst. Appl. Services*, New York, NY, USA, 2021, pp. 54–66. [Online]. Available: https://doi.org/10.1145/3458864.3467681

[41] F. Sattler, K.-R. Müller, and W. Samek, "Clustered federated learning: Model-agnostic distributed multitask optimization under privacy constraints," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 8, pp. 3710–3722, Aug. 2021.

[42] H. Huh and J. Y. Kim, "LoRa-based mesh network for IoT applications," in *Proc. IEEE 5th World Forum Internet Things (WF-IoT)*, 2019, pp. 524–527.

[43] Q. Wang, Y. Ma, K. Zhao, and Y. Tian, "A comprehensive survey of loss functions in machine learning," *Ann. Data Sci.*, vol. 9, pp. 187–212, Apr. 2022.

[44] "Deploy machine learning models on mobile and IoT devices." Accessed: Mar. 1, 2022. [Online]. Available: https://www.tensorflow.org/lite

[45] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, Apr. 2017.

[46] J. K. Chow, Z. Su, J. Wu, P. S. Tan, X. Mao, and Y. H. Wang, "Anomaly detection of defects on concrete structures with the convolutional autoencoder," *Elsevier Adv. Eng. Inform.*, vol. 45, Aug. 2020, Art. no. 101105.

[47] A. Mood, F. Graybill, and D. Boes, *Introduction to the Theory of Statistics*, 3rd ed. New York, NY, USA: McGraw-Hill, 1974.

[48] B. Vandalore, S. Fahmy, R. Jain, R. Goyal, and M. Goyal, "General weighted fairness and its support in explicit rate switch algorithms," *Comput. Commun.*, vol. 23, no. 2, pp. 149–161, 2000.

Joannes Sam Mertens received the Bachelor of Engineering degree in electronics and communication engineering and the Master of Engineering degree in communication systems from the SSN College of Engineering, Chennai, India, in 2017 and 2019, respectively, and the Ph.D. degree in telecommunications engineering from the University of Catania, Catania, Italy, in 2022.

Since March 2023, he has been with the Dipartimento di Ingegneria Elettrica Elettronica e Informatica, University of Catania, where he is currently working as an Assistant Professor. His research interests include wireless sensor networks, machine learning, and IoT.

Laura Galluccio (Senior Member, IEEE) received the Laurea degree in electrical engineering and the Ph.D. degree in electrical, computer and telecommunications engineering from the University of Catania, Catania, Italy, in 2001 and March 2005, respectively.

Since 2002, she has been also with CNIT (Italian National Consortium of Telecommunications), Parma, Italy, where she worked as a Research Fellow with the Virtual Immersive Communications and the SATNEX Projects. From November 2010 to October 2019, she was an Assistant Professor with the Department of Electrical, Electronics and Computer Engineering, University of Catania, where she has been an Associate Professor since 2019. From May 2005 to July 2005, she was a Visiting Scholar with the COMET Group, Columbia University, New York, NY, USA. Since September 2015, she has been a Visiting Professor with Central Supelec, Gif-sur-Yvette, Paris, France. Her research interests include sensor and underwater networks, protocols and algorithms for wireless networks, and network performance analysis. She works also on unconventional communication networks and specifically ultrasonic and microfluidic networks.

Dr. Galluccio was the recipient of the ACM Nanocom Outstanding Milestone Award for her seminal contribution to neural system interfacing for applications in the Internet of Bio-NanoThings and Microfluidic Nano Networks Research in 2022. In 2022, she was added to the "Star in Computer Networking and Communications" List issued yearly by ACM N^2 Women and listing the top 10 female researchers who impacted mostly the scientific community with their research.

Giacomo Morabito received the Ph.D. degree in electrical, communications, and computer engineering from the University of Palermo, Palermo, Italy, in 2000.

He is a Full Professor of Telecommunications with the University of Catania, Catania, Italy. From November 1999 to April 2001, he was with the Broadband and Wireless Networking Laboratory, Georgia Institute of Technology, Atlanta, GA, USA. He has been involved has led several research projects. His main research interests are in Internet of Things and software-defined networks.

Dr. Morabito has served in the editorial boards of the IEEE NETWORKING LETTERS, *IEEE Wireless Communications Magazine*, the *Computer Networks* (Elsevier), and *Wireless Network*. He has also been the General Co-Chair of the Med-Hoc-Net 2006 and ACM NanoCom 2016 Conference and the TPC Chair of Med-Hoc-Net 2004 and ACM SIGCOMM-ICN.

Open Access funding provided by 'Università degli Studi di Catania' within the CRUI CARE Agreement