FedMCCS: Multicriteria Client Selection Model for Optimal IoT Federated Learning

Sawsan AbdulRahman, Hanine Tout, Azzam Mourad, *Senior Member, IEEE*, and Chamseddine Talhi

***Abstract*—As an alternative to centralized systems, which may prevent data to be stored in a central repository due to its privacy and/or abundance, federated learning (FL) is nowadays a game changer addressing both privacy and cooperative learning. It succeeds in keeping training data on the devices, while sharing locally computed then globally aggregated models throughout several communication rounds. The selection of clients participating in FL process is currently at complete/quasi randomness. However, the heterogeneity of the client devices within Internet-of-Things environment and their limited communication and computation resources might fail to complete the training task, which may lead to many discarded learning rounds affecting the model accuracy. In this article, we propose FedMCCS, a multicriteria-based approach for client selection in FL. All of the CPU, memory, energy, and time are considered for the clients resources to predict whether they are able to perform the FL task. Particularly, in each round, the number of clients in FedMCCS is maximized to the utmost, while considering each client resources and its capability to successfully train and send the needed updates. The conducted experiments show that FedMCCS outperforms the other approaches by: 1) reducing the number of communication rounds to reach the intended accuracy; 2) maximizing the number of clients; 3) handling the least number of discarded rounds; and 4) optimizing the network traffic.**

***Index Terms*—Bilevel optimization, cooperative learning, federated learning (FL), Internet of Things (IoT), linear regression, machine learning, multicriteria selection, privacy, resource management, resource utilization prediction.**

I. Introduction

WHO AMONG us did not allow Internet usage, through mobile devices, to work its way into their everyday life? By accessing and sharing data on the go, a world of information is nowadays generated not only from smartphones, but also from Internet-of-Things (IoT) devices [1]. Using such tremendous data to train machine learning models makes the latter more robust and produces more intelligent applications. However, this requires storing the data in a centralized entity, which entails the following issues: 1) compromise the privacy of shared data; 2) prevent some users from sharing their personal and useful data; and 3) entail high latency, etc.

As a solution, Google coined the term federated learning (FL) [2] to take us a step further in preserving privacy and addressing the aforementioned problems. It is a decentralized approach that distributes the training tasks on client devices to keep their private data locally. The FL life cycle is divided into several communication rounds, which are concluded once the model converges, or in other words, a desired test-set accuracy is reached. In each round, the server first communicates the global model parameters with some clients selected at random. Then, the data, that is supposed to be sent to the server in the typical approaches, remains on the devices, and is used for training. To benefit from peers' models, the locally computed updates are therefore shared with the server for aggregation.

From one round to another, different set of clients are selected at complete [2] or quasi [3] randomness. When the selection comes to some clients with limited resources, like IoT devices, not only longer processing time is engaged by the client, but also failure in completing the training task might occur, and accordingly affect the model accuracy. Moreover, since FL takes place when the devices are idle, the location of the clients highly affects their participation in the ongoing round. It has been reported [4] that the number of clients participating in FL highly depends on the time of day, where at night time it increases by 4× compared to day time. Therefore, the random selection of clients leads to less number of updates sent by the clients, yet some FL rounds will be discarded. Furthermore, training data set represents data generated by the clients based on their behavior. Yet, imbalanced class distribution is a common machine learning problem, where samples belonging to one class dominates those belonging to another class. If data of the selected clients in FL belongs to one class, FL process will not be sufficient enough to build robust models. To the best of our knowledge, none of the current approaches has addressed optimizing FL client selection while considering location, resources and survivability of the devices chosen for training the models.

In this article, we tackle the aforementioned limitations by proposing FedMCCS, a multicriteria-based client selection approach for FL, formulated as bilevel optimization problem while considering availability of resources, communications overhead and imbalanced distribution of data. First, stratified-based sampling is used to filter the clients located at same

Manuscript received June 16, 2020; revised August 26, 2020; accepted September 28, 2020. Date of publication October 5, 2020; date of current version March 5, 2021. This work was supported in part by MITACS, in part by Ericsson Canada, in part by ÉTS Montreal, and in part by Lebanese American University. *(Corresponding author: Azzam Mourad.)*

Sawsan AbdulRahman, Hanine Tout, and Chamseddine Talhi are with the Department of Software Engineering and IT, École de Technologie Supérieure, Montreal, QC H3C 1K3, Canada (e-mail: sawsan.abdul-rahman.1@ens.etsmtl.ca; hanine.tout.1@ens.etsmtl.ca; chamseddine.talhi@etsmtl.ca).

Azzam Mourad is with the Department of Computer Science and Mathematics, Lebanese American University, Beirut 961, Lebanon (e-mail: azzam.mourad@lau.edu.lb).

Digital Object Identifier 10.1109/JIOT.2020.3028742

time zone. This engages selecting homogeneous clients all able to be contacted for training requests. Next, communication with the filtered clients will be initiated to acquire about their resources utilized in the prior FL rounds. According to the responses, a linear regression-based algorithm predicts whether the client has enough CPU, memory, and energy to perform the training task. Estimation time for the download, update, and upload of the model is calculated and compared to the threshold for receiving all the updates defined by the server. Moreover, to handle the imbalanced distribution of the data set, we prioritize the clients with highest event rate according to the representation of data labeled as the minority class of the studied use case. Using all above metrics, our approach is able to maximize the number of selected clients, aggregate more updates per round and minimize the total communication rounds. We base our goal of maximizing the clients on the results obtained by Google in [2], showing that the desired accuracy is converged faster when more updates are aggregated per round. Moreover, in typical FL, the aim is to select a fraction of clients. However, some might not be able to respond, or even dropout during the process due to their limited connectivity and/or resources, and hence, a smaller number of updates will be received. Therefore, in our proposition, we aim the maximize such fraction of clients with respect to the aforementioned criteria. In the following, we provide a summary of this article contributions.

1) *Stratified-based sampling of clients* that allows to form homogeneous group of clients based on their location in order to filter the ones able to respond to the server request.
2) *Novel multicriteria-based optimization model* that allows to maximize the number of FL clients with sufficient resources and time to complete the training tasks without dropout. Our scheme provides also linear regression-based mechanism for predicting the utilization of the CPU, Memory and Energy of the ongoing FL round in addition to the predicted time for the download, update and upload of the model parameters.
3) *Optimal balanced distribution of the training set* to optimize the client selection by prioritizing the available clients for training with the highest event rates.
4) *Efficient Approach for Intrusion Detection System:* We took a case study of intrusion detection system to evaluate our approach, where NSL-KDD [5], a publicly available data set has been used. The experiments show, compared to two other baselines: a) a reduction of communication rounds by $8.0 - 8.4\times$; b) more client selection per round; c) significant decrease in the discarded rounds; and d) minimization of the network overhead.

The remainder of this article is organized as follows. In Section II, we present some relevant existing works. In Section III, we elaborate more on how FL works, what are the existing protocols for client selection, and the problems entailed in the practical FL. Section IV introduces the architecture and protocol of our framework. Next, our maximization problem and proposed model are presented in Section V. Finally, we show the evaluation methodology and experimental results in Sections VI and VII, respectively, followed by a conclusion in Section VIII.

II. Related Work

Once Google invented FL, different related aspects have been advanced in the field of models aggregation [2], [6], [7], applications [8]–[10], privacy [11]–[13], security [14]–[16], etc. Since the main objective of efficiently selecting clients to participate in the FL rounds is to reduce the communication cost, we study first the state-of-the-art in this field. In what follows, we emphasize on the communications cost aspect in FL and discuss the relevant existing works, while the rest of the aspects are out of the scope of this work.

Konecný *et al.* [17] investigated two methods to minimize the communication cost. *Structured updates* is the first proposed method, which restricts the model updates to be within a prespecified structure. The latter necessitates building two matrices for the update, where only the needed matrix or nonzero values should be sent by the client. As for Sketched updates, the second proposed method, the updates to be communicated are first computed, then compressed using subsampling, probabilistic quantization, and random rotation techniques. The extended FL proposed in [18] works as follows.

1) The size of the global model is reduced using Federated Dropout technique, where a submodel with less parameters is constructed.
2) The resulting submodel is lossily compressed on the server side and sent to the clients.
3) The latter decompress the model and perform the training task.
4) The generated updates are as well compressed and sent to the server.
5) The server decompresses to do the aggregation.

The work in [19] meets the requirements of FL from a communication-efficiency perspective: by compressing not only the upstream communication but also the downstream one, being robust to noniid and unbalanced data with small batch sizes, and by handling big number of clients as well as their partial participation. The proposed compression methods are performed via sparsification, optimal Golomb encoding, error accumulation, and ternarization. An enhanced FL protocol has been proposed in [20]. First, the communication cost is minimized by reducing the number of parameters shared between the server and the clients to be trained and updated. This is done by separating shallow layers from deep layers in a deep neural network, and communicating more frequently the shallow layers related parameters as their features are more important for the central model. Another proposition has been added to this work called Temporally Weighted Aggregation. Rather than considering the recent learning models only, the proposed aggregation method takes into account the local models of each client trained in all previous rounds. Instead of using the standard global averaging for the aggregation method, the work in [21] proposed an Adam-based per-coordinate averaging strategy. The latter is developed and tested for wake word detection and was able to reduce the communication rounds to reach a target recall value. As a large number of random clients are selected to train the parameters of the global model in FL, some yet many updates, that are irrelevant, will be sent to the server. Therefore, a communication-mitigated FL has been proposed in [22]. In each round, how relevant the client's update is to the global model, it will be either communicated or not. The use of FL in unmanned aerial vehicles (UAVs) has also been proposed in [23]. To reduce signaling delays caused by the communication with remote node in large-scale UAV networks, the authors proposed on-device ML, particularly FL, where UAVs storing different tasks work in a distributed and collaborative fashion. When, in such proposition, radio resources are allocated on many UAVs, wireless congestion and latency are reduced.

These approaches were able to reduce the communication cost by sending nonzero values of the update matrix [17], compressing the upstream and downstream communication updates [18], [19], updating more frequently the shallow layers related parameters [20], modifying the averaging method [21], sharing the relevant updates only [22], and collaboratively allocating radio resources in UAVs [23]. While not considering the devices resources and their capabilities to finish learning and communicating their updates, such practices still result in many discarded FL rounds. They also risk wasting significant device resources as the successfully learnt models will be dropped in a discarded round.

Another line of work focused on the efficiency of FL when selecting the clients under realistic wireless network. Chen *et al.* [24] shed light on how wireless factors affect FL performance in practice. In this context, FL might come across training errors due to the unreliability of the wireless channels, in addition to the wireless resource limitations, mainly in the power and bandwidth. To address such limitation, the authors proposed novel FL-based framework to minimize FL loss function under wireless networking metrics. Yang *et al.* [25] proposed a framework to study how FL algorithms converge in large-scale wireless networks. Besides the tractable expressions that are derived for the convergence rate, the latter is analyzed and compared using three different scheduling policies: 1) round robin; 2) random scheduling; and 3) proportional fair. Chen *et al.* [26] focused on how client selection mechanism can highly affect FL convergence time and performance. They proposed a scheme where clients with high-quality models are most likely to be connected to the base station (BS), which acts as a server in typical FL. Moreover, the convergence time is reduced by allowing BS, using artificial neural networks, to deal with the clients whose local models parameters have some relationship. Ren *et al.* [27] proposed a scheduling policy in order not to bias gradient aggregation. First, one client is scheduled per round and the importance of its model updates are measured and analyzed using gradient convergence. Then, multiple clients are scheduling in a proposed probabilistic scheduling framework.

While the aforementioned wireless network-based approaches focus on the resource constrained wireless networks and choosing only the clients generating high-quality local models to reduce communication, we take another direction by analyzing the resources inside the IoT devices and predicting their capabilities of completing the training task.

Protocol 1 FL. K Represents the Number of Participants in the Protocol. $C \in (0, 1]$ is a Hyperparameter Determining the Fraction of Clients Involved in Each Round

```
1: Initialization : The server first creates a generic model either randomly or pretrained using public data.
2: Client Selection : The server selects random $\lceil K \times C \rceil$ clients.
3: Distribution : The server disseminates the global model parameters to the selected clients.
4: Update and Upload : Selected clients use their local data to update the shared model and upload the new model parameters to the server.
5: Aggregation : The server performs an averaging process on the updated parameters to formulate an enhanced model.
6: Steps 2 till 5 are repeated until achieving a desired performance of the model.
```

On the other hand, none of the current approaches has considered the location and resources of the clients in the selection step. In this article, we first filter the clients based on their location, then we formulate a client selection mechanism that predicts the clients able to efficiently participate in the FL rounds considering their heterogeneous resources. The proposed protocol is able not only to reduce the communication overhead, but also to make full advantage of the learnt models, and reduce the number of discarded rounds.

III. Federated Learning and Problem Illustration

In this section, we describe the existing protocols of FL, followed by the encountered problems in the current and practical FL.

A. Existing Federated Learning

The sensitivity of the wealth of data generated by the devices is preventing the users from storing their data in a centralized entity. If all generated data is gathered, more intelligent applications would have been modeled. To achieve such tradeoff between privacy and intelligence, FL [2] was proposed by locally training data and sharing machine learning models with a server. Protocol 1 describes FL steps. Initially, the server generates a generic model for a certain task, then selects random clients to communicate the model parameters with. The number of clients selected is equal to $\lceil K \times C \rceil$, where K is the total number of clients, and C, a hyperparameter that defines the fraction of clients to be involved in each round. In the `Update and Upload` step, each selected client trains the model using its local data and shares the new generated parameters with the server. Once the server receives the clients updates, it starts averaging them to get an enhanced model. In case no enough updates are received and a delay is reported, the server abandons the round. At the end, the steps are iterated until achieving a desired model performance.

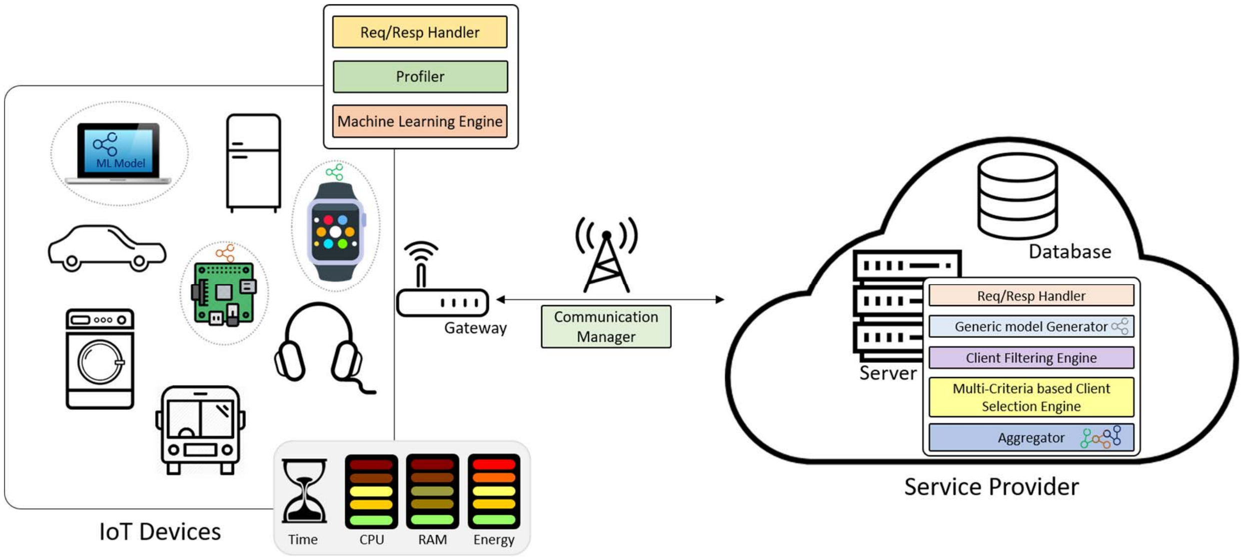


Fig. 1. High-level architecture of the proposed framework.

Optimizing resource-constrained devices has attracted many researchers [28]–[30]. In the context of FL, the work in [3] has proposed a client selection-based FL solution to efficiently manage the clients involved in the learning rounds. The authors implement Protocol 2, where the mobile edge computing (MEC) operator requests resource information from random clients, rather than directly sending them the model parameters. In the next step, `Client Selection`, the MEC operator maximizes the $\lceil K \times C \rceil$ clients to be participating in the training task for the current round. The selection depends on the time needed to distribute the parameters over the clients, in addition to the time taken by each client to upload and update the model. The maximization is hence achieved with the clients consuming the least time under a certain threshold. Next, the server disseminates the model parameters only to the selected clients to update the shared model. After the aggregation, the server again selects, from one round to another, new set of clients until achieving a desired performance, or when a new defined threshold is met.

B. Problems Entailed in Current Federated Learning

Existing protocols of FL, especially when applied in the IoT environment [3], [31], [32], engender many problems in the local training phase. The clients are very heterogeneous in nature. They have different communication and computation resources, variant amount of generated data, and different time zones. When selected for training, the devices have to complete the process and sends the response within a reasonable time. In case of failure, whether due to unreliable network connectivity, or the waiting time for a device to be idle to start the training, the server has to abandon the round. Such scenario wastes the utilized resources of all participating clients in the current round. Moreover, some devices might fail to perform some training tasks if they have large amount of data, and/or low computation resources. This affects the devices by causing system crash, and consequently increasing the number of communication rounds needed to reach a target accuracy for the model.

Protocol 2 FL With Client Selection. K Represents the Number of Participants in the Protocol. $C \in (0, 1]$ is a Hyperparameter Determining the Fraction of Clients to Answer the Resource Request in Each Round

```
1: Initialization in Protocol 1.
2: Resource Request : The MEC operator requests resource information from ⌈K × C⌉ random clients.
3: Client Selection : Based on their responses, the MEC operator selects the clients that are capable of performing the remaining steps within a certain deadline.
4: Distribution : The server disseminates the global model parameters to the selected clients.
5: Scheduled Update and Upload : Selected clients use their local data to update the shared model and upload the new model parameters to the server, using their allocated RBs.
6: Aggregation in Protocol 1.
7: Steps 2 till 5 are repeated until achieving a desired performance of the model or when the final deadline is met.
```

IV. FedMCCS: Architecture and Protocol

Fig. 1 shows the high-level architecture of our proposed FL-based framework. Our solution encompasses two entities; the service provider and IoT devices, both communicating using any communication infrastructure. While the devices act as clients, the service provider can offer many ML-based services for different applications, such as image recognition, intrusion detection, etc. In what follows, we describe the components of each entity.

1) *Req/Resp Handler:* It is responsible of handling the exchanged request and response messages between the server and the selected devices.
2) *Generic Module Generator:* It is responsible of building a global ML model to be communicated and used by the clients.

3) *Client Filtering Engine:* It is responsible of selecting homogeneous rather than random group of clients. The role of this component is detailed next, when describing the protocol.
4) *Multicriteria-Based Client Selection Engine:* It is the main module in our system. It allows to select the maximum number of available and performant clients (colored devices in the figure) per round in order to aggregate their updates. The proposed selection algorithm employs multiple criteria (time, CPU, memory, and energy) to achieve the maximization goal.
5) *Aggregator:* It is responsible of creating much improved update of the model using a federated average function.
6) *Profiler:* It is responsible of both monitoring the utilized resources while training a model, and creating resource profile for the device.
7) *Machine Learning Engine:* It is responsible of performing the training task using the local data stored on the device.

Our proposed solution, mainly for IoT system, is a multicriteria-based approach for client selection. It maximizes the number of selected clients capable of providing a highest possible performance when aggregating their models among devices with limited resources. Protocol 3 shows the steps of the proposed FL and reflects the actual interactions among the components described in the architecture.

1) *Initialization:* The server generates the global model either randomly or pretrained using public data.
2) *Client Filtering:* In the existing FL protocols, the selection of clients from one round to another is based on *simple sampling*, where all clients are equally likely to be chosen. This can lead to undesirable exchanged messages, the selection of irrelevant clients, and less reliable responses, which affect the model performance. For instance, significant delay can be entailed when selecting subsets of clients located at different time zone. It has been reported in [4] that the clients when at night are involved in the FL rounds more than when at day.
   Rather than selecting clients entirely at random, we leverage *stratified sampling* [33], which classifies the population into subgroups known as strata. The latter groups homogeneous users sharing similar characteristics together. Metadata about the clients is often already stored on the server or even can be shared with the latter. In our proposition, we assign clients to strata based on their region metadata.
3) *Resource Request:* From only the filtered clients, the server requests their resources information (e.g., the size of data they have, their historical data for the past training tasks, etc.).
4) *Multicriteria Client Selection:* The server analyzes the clients responses to select the best set able to participate in the coming learning rounds. How to estimate the clients resources, and which clients to select are presented in details in Section V.
5) *Distribution:* The server distributes the model parameters to the selected clients.

Protocol 3 FL With Multicriteria Client Selection. K Represents the Number of Participants in the Protocol. $C \in (0, 1]$ is a Hyperparameter Determining the Fraction of Clients to Select, After Being Filtered Based on Their Metadata, and After Analyzing Their Resources

```
1: Initialization in Protocol 1.
2: Client Filtering : The server applies Stratified-based filtering to select clients according to their metadata, avoiding communications with irrelevant clients.
3: Resource Request : The server requests resource information from the filtered clients.
4: Multicriteria Client Selection : Based on the clients responses, the server uses Multicriteria selection approach to determine a maximum of $\lceil K \times C \rceil$ clients to participate in the remaining steps.
5: Distribution : The server disseminates the global model parameters to the selected clients.
6: Update and Upload in Protocol 1.
7: Aggregation : The server averages the parameters, when more than 70% of the requested updates are received.
8: All steps but Initialization are iterated as in Protocol 2.
```

6) *Update and Upload:* The clients update and upload the new model parameters.
7) *Aggregation:* The constrained resources of the devices might result in many systems crash. Therefore, we compensate for the devices dropout by handling 30% of unresponsive selected clients at each round, following similar settlement approach as in [4]. When more than 30% of the updates are not received, the FL round is considered discarded. Otherwise, the aggregation is successfully performed.
8) A loop over the previous steps is defined until the desired model performance is achieved.

V. Multicriteria-Based FL Client Selection Model

A. Problem Definition

We address the following problem: considering the heterogeneity of the clients and their limited computation and communication resources, what is the maximum number of those that can complete FL rounds without system crash, while maintaining high-performance FL model?

Formally, let $X = (X_1, X_2, \dots, X_K)$ be the set of all clients, each having a set of m pairs $\{n, l\}$ where n is a network related data and $l \in \{\text{normal}, \text{abnormal}\}$ its label. First, find $X_f = (X_{f_1}, X_{f_2}, \dots, X_{f_i})$, where $X_f \subseteq X$, representing the filtered clients based on the stratified method to classify clients in homogeneous groups, which better represents the whole set (step 2, Protocol 3). Next, find the maximized set of clients to be selected in the `multicriteria client selection` (step 4, Protocol 3) represented by $X_S = (X_{S_1}, X_{S_2}, \dots, X_{S_j})$, where $X_S \subseteq X_f$ and $j \leq \lceil K \times C \rceil$.

B. Problem Formulation

We formulate our problem as a bilevel maximization with knapsack and other constraints as follows:

$$\begin{aligned}
&\max_{X_S} \ |\mathrm{X_S}| \\
&\text{subject to} \\
&\qquad \begin{cases} \forall X_{f^i_{z=1}} \sum \mathrm{Util}^{X_{f_z}}_{r\in\{\mathrm{CPU},\mathrm{Memory},\mathrm{Energy}\}} < \mathrm{Budget}^{X_{f_z}}_r [co_1] \\ \forall X_{f^i_{z=1}} \sum \left(T^{X_{f_z}}_d + \mathrm{Util}^{X_{f_z}}_{r=T_{ud}} + T^{X_{f_z}}_{ul} \right) < T[co_2] \end{cases} \\
&\text{subject to} \\
&\qquad \max ER_{X_{f^i_{z=1}}} = \left[\frac{|X_{f_z}.l_A|}{|X_{f_z}.l_A| + |X_{f_z}.l_N|} \times 100 \right] [co_3].
\end{aligned} \tag{1}$$

The aim is to maximize the set of selected clients X_S under three constraints.

1) *co_1:* The limited budget for the resources utilization of each device type, in a way not to cause dropouts. We define dynamic budgets based on the resources types per device type. Such budget, which represents the maximum on-device task consumption, allows to complete the training process until completion. $\mathrm{Util}^{X_{f_z}}_{r\in\{\mathrm{CPU},\mathrm{Memory},\mathrm{Energy}\}}$ denotes the predicted utilization of the resources r for the client X_{f_z} when training a model, where r represents the CPU, memory, or energy. $\mathrm{Budget}^{X_{f_z}}_r$ is the resource budget per device type.
2) *co_2:* The defined threshold T not to be exceeded when downloading, updating, and uploading a model. $\mathrm{Util}_{r=T_{ud}}$ represents the predicted utilization of the update time resource when training a model. Whereas, $T^{X_c}_d$ and $T^{X_c}_{ul}$ denote, respectively, the time required to download and upload the model by a client X_c.
3) *co_3:* The selection of the needed number of clients based on their event rate. The clients have nonindependent and nonidentically distributed data set, where the latter is generated depending on each client behavior and usage of the application related to the ML model. Therefore, some clients might produce models with imbalanced classification, with a majority of the samples belonging to only one class. We denote by *ER* the event rate of the client data set, showing the representation of the minority class distribution, whereas $X_{f_z}.l_A$ and $X_{f_z}.l_N$ represent, respectively, the abnormal and normal samples of the client X_{f_z}.

Theorem 1: Our multicriteria FL client selection problem is NP-hard.

Our optimization problem is a bilevel maximization problem with a knapsack constraint [34]. The objective is to maximize the number of clients to be selected $|X_S|$ subject to a defined threshold T that the download, update, and upload time of the model taken by all X_S cannot exceed. Another constraint in the maximization problem is the resources utilization of the clients that should be within a limited budget. Those two constraints are applied subject to the type of the clients data sets. Since our problem is a bilevel maximization with a knapsack constraint, hence it is NP-hard [35], [36].

Proof: Our formulated maximization problem can reduce to the classical knapsack problem where the aim is to determine the items to be included in a sack in a way that the total weight is less than or equal to a specific limit and the total value of the selected items is maximized.

Now given an instance of the client selection problem, we transform it into an instance of the knapsack problem as we defined as follows.

1) The set $|X_S|$ as the sack in the knapsack problem.
2) The clients to be selected are the items in the knapsack problem.
3) The clients' weights through their resources ($\mathrm{Util}^{X_{f_z}}_{r\in\{\mathrm{CPU},\mathrm{Memory},\mathrm{Energy}\}}$, which should be $< \mathrm{Budget}^{X_{f_z}}_r$) as well as the total time needed to download, update, and upload the model ($\sum(T^{X_{f_z}}_d + \mathrm{Util}^{X_{f_z}}_{r=T_{ud}} + T^{X_{f_z}}_{ul})$ which should be $< T$), whereas the clients values as their event rates ($ER_{X_{f^i_{z=1}}}$). This is exactly as the items in knapsack which have given weights and values.

This reduction yields our client selection problem is NP-hard. ■

C. Heuristic-Based on Greedy Algorithm for Client Selection

Solving (1) requires combinatorial optimization. We propose a heuristic based on greedy algorithm [3], described in what follows.

After filtering the clients (in step 2, Protocol 3), the server requests from each client in the filtered set X_f its resource information. This includes the size of data samples relevant to the training task (e.g., the number of samples $|l_N| : l \in \{\text{normal}\}$ and $|l_A| : l \in \{\text{abnormal}\}$ in the data set), in addition to the historical resource data set collected during previous FL participations of the client (e.g., for each size of trained data its utilized resources, mainly the training/update time, CPU, memory, and energy denoted by $\mathrm{Util}^{X_c}_r$). After receiving the clients responses, the server starts its client selection algorithm.

First, one of the most common machine learning problems is the imbalanced class distribution, where the volume of samples belonging to one class dominates the volume of those belonging to another class. Thus, the predictive detection model could be inaccurate and biased with the imbalanced classification. Duan *et al.* [37] showed significant decrease in the global model accuracy when the training data is imbalanced. As a solution, the authors proposed data augmentation strategy for the minority classes. Moreover, the use of mediators between the clients and the server is introduced in order to rebalance training and achieve a partial equilibrium. In other words, a mediator selects the clients that make the distributed data close to a uniform distribution. However, such approach requires more traffic with time overhead in each communication round compared to the typical FL. Another work [38] has been advanced studying the issue of fairness in FL. The proposed framework is based on minimax optimization, which prevents data overfitting. While such approach optimizes the centralized model and mitigates bias in the training procedure rather than training data, we focus in our work on the clients themselves to ensure utmost uniform distribution. Particularly, we prioritize the clients having highest event rate, which leads to less bias, even though they have the largest number of abnormal samples. If, for instance, client A has 4000 samples, among which 70 are abnormal, and another client B has 200 samples, among which 50 are abnormal, we prioritize client B since the event rate of its data equals to 25% is much higher than that of Client A with only 1.75%.

Next, from the prioritized clients, we maximize the number of clients to be selected based on their resources. Hence, avoid opting clients incapable of completing the training phases, which can corrupt the ongoing FL rounds or entail late replies. This is technically formulated by proposing a prediction method for the resource utilization based on linear regression [39]. It estimates the resources utilization $\text{Util}_r^{X_c}$ for the next training round for a given client according to the history of the past utilized resources. The proposed method, RUPred-LR, is presented in details in the next Section V-D, and its pseudocode algorithm is given in Algorithm 2. Based on the predicted $\text{Util}_{r\in\{\text{CPU},\text{Memory},\text{Energy}\}}$, each client should be able to perform the training task under a fixed resource budget, to avoid over-loaded devices. We use $\text{Budget}_r^{X_{f_z}}$ to denote the budget of type-r resource for each device type. Besides $\text{Util}_{r\in\{\text{CPU},\text{Memory},\text{Energy}\}}$, we estimate the time needed by the clients to download, update, and upload a model. Those able to complete the process within a defined threshold T will be selected, which maximizes the expected benefit. $T_{ud}^{X_c}$ is estimated using RUPred-LR, whereas $T_d^{X_c}$ and $T_{ul}^{X_c}$ are estimated using

$$T_{x\in\{d,ul\}} = \frac{|\text{modelPar}|}{B} + \text{latency} \tag{2}$$

where *modelPar* and B represent the model parameters size and the network bandwidth, respectively.

The full pseudocode of the multicriteria-based client selection is given in Algorithm 1. The client having maximum *ER* is first chosen (line 3 in Algorithm 1), then is added to X_S if its resources required for the training task are sufficient (lines 5 and 6 in Algorithm 1). Besides handling the imbalanced class distribution problem, the selection model based on the proposed *ER* reduces the algorithm complexity: Instead of predicting the resources utilization for all X_f clients, then selecting $\lceil K \times C \rceil$ with highest event rate, the complexity of the algorithm is reduced by only performing the prediction for the clients with highest *ER* until selecting $\lceil K \times C \rceil$, or until no more clients are available. The order of the algorithm is hence $\mathcal{O}(n|X_f||X_S|)$. Note that in the proposed client selection algorithm, the extra overhead lies in both predicting the resources utilization of the clients and estimating whether such resources are under the specified budgets in order to be selected. Since these additional calculations are performed on the FL server and we focus in this work on providing low computation burden for the devices themselves, we tolerate such computational cost in resource-constrained IoT environments.

Algorithm 1 Multicriteria Client Selection in Protocol 3

```
Input: {$X_f$:$\forall X_{f_{z=1}^i}$, $\exists(|l_N|, |l_A|$, and $History\_X_{f_z})$}, where:
  • $History\_X_{f_z} = \bigcup_{i=1}^{n}\{x_i, y_i = Util_r\}$
  • $x_i$ is the data set size previously trained by $X_z$
  • $Util_r$ is the resource utilization to train $x_i$ data samples, where $r \in \{CPU, Memory, Energy, T_{ud}^{X_z}\}$
  • $n$ is the size of historical resource data set collected during FL participation of $X_z$
Output: The set of selected clients $X_S$

1: Initialize $X_S = \varnothing$
2: while $X_f \neq \varnothing$ and $|X_S| \neq \lceil K \times C \rceil$ do
3:     $X_{f_z} \leftarrow \text{argmax}_{X_{f_k} \in X_f}\left[\frac{|X_{f_k}.l_A|}{|X_{f_k}.l_A|+|X_{f_k}.l_N|} \times 100\right]$
4:     remove $X_{f_z}$ from $X_f$
5:     if sufficientResources($History\_X_{f_z}$) then
6:         add $X_{f_z}$ to $X_S$
7:     end if
8: end while
9: Return the set $X_S$

sufficientResources($History\_X_{f_z}$)
($Util_{CPU} < Budget_{CPU}^{X_{f_z}}$ &&
$Util_{Memory} < Budget_{Memory}^{X_{f_z}}$ &&
$Util_{Energy} < Budget_{Energy}^{X_{f_z}}$ &&
$[T_d + Util_{T_{ud}} + T_{ul}] < T$) ? true:false;
```

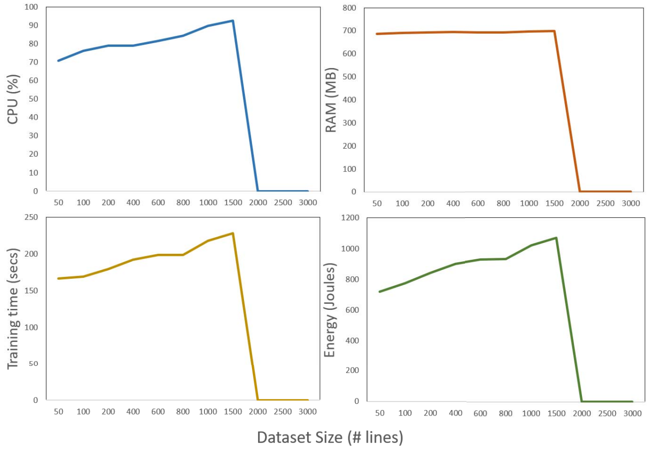


Fig. 2. Resources utilization variation in terms of data set size.

D. RUPred-LR—Resource Utilization Prediction Based on Linear Regression

To predict the future resources utilization [40], [41] for a particular training task, we studied the variation of the resource utilization with respect to the data set size before the system crashes. In these experiments, we examined the training time, along with the utilized CPU, Memory, and Energy for Raspberry Pi devices. The latter are characterized by quad-core processor of 1.2 GHz, 1 GB of RAM, 16 GB of storage, and Rasberian Debian OS. The results in Fig. 2, which are averaged from ten devices, reveal linear relation with a negligible error for each of the studied parameters when the devices are active and performing the training task. The linear modeling continues to be observed as long as we increase the data set size until reaching 2000 samples exclusively. Once the model is fed with 2000 training set samples, the devices shutdown, which explains the falling linear-based resource utilization variation. Hence, to efficiently use the devices resources and avoid system crash, which corrupts FL round, we propose new method based on linear regression [39] to predict resource utilization. Its full pseudocode is given in Algorithm 2. According to the history of the clients in the past FL rounds, the server estimates a prediction function for each of the training time, CPU, Memory, and Energy. This function, presented in (3), shows linear relationship between the input variable x and the output variable y_r. We denote by x the data set size of the client and by y_r the device resource utilization Util$_r$, where r represents the training time, CPU, Memory, or Energy

$$\text{Util}_{r\in\{T_{ud},\text{CPU},\text{Memory},\text{Energy}\}} = y_r = \alpha_r x + \beta_r \tag{3}$$

where α_r and β_r are the regression coefficients of the model.

To measure the goodness of fit - in other words, how well the resource utilization is predicted, we get the residual ε_r in (4) for each of the n historical records. The smaller the residuals, the better the fit

$$\varepsilon_{r_i} = \left|y_{r_i} - \hat{y}_{r_i}\right| \tag{4}$$

where y_{r_i} and $\hat{y}_{r_i}$ are the real and the predicted utilization values, respectively.

The objective is to obtain the regression coefficients by minimizing the residuals leading to the best fit line. One of the common methods for the residual minimization is least-squares regression [39], which finds α_r and β_r with the minimal possible value of the sum of the squared deviations over the n records. Thus, least-squares regression tends to minimize the following function:

$$S(\alpha_r, \beta_r) = \sum_{i=1}^{n} \varepsilon_{r_i}^2 = \sum_{i=1}^{n}\left(y_{r_i} - \hat{\alpha}_r x_i - \hat{\beta}_r\right)^2 \tag{5}$$

where $\hat{\alpha}_r$ and $\hat{\beta}_r$ are the least-squares estimators of α_r and β_r.

The minimization is computed by taking partial derivatives of S with respect to $\hat{\alpha}_r$ and $\hat{\beta}_r$, and assigning zero to each partial derivative as follows:

$$\frac{\partial S}{\partial \beta_r} = -2\sum_{i=1}^{n}\left(y_{r_i} - \hat{\alpha}_r x_i - \hat{\beta}_r\right) = 0 \tag{6}$$

$$\frac{\partial S}{\partial \alpha_r} = -2\sum_{i=1}^{n}\left(y_{r_i} - \hat{\alpha}_r x_i - \hat{\beta}_r\right) = 0. \tag{7}$$

Solving (6) and (7) yields the following least-squares estimators:

$$\begin{aligned}\hat{\alpha}_r &= \frac{\sum_{i=1}^{n} y_{r_i} x_i - \frac{1}{n}\left(\sum_{i=1}^{n} y_{r_i}\right)\left(\sum_{i=1}^{n} x_i\right)}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2}\\ &= \frac{\sum_{i=1}^{n}(x_i - \bar{x})\left(y_{r_i} - \bar{y}_r\right)}{\sum_{i=1}^{n}(x_i - \bar{x})^2}\end{aligned} \tag{8}$$

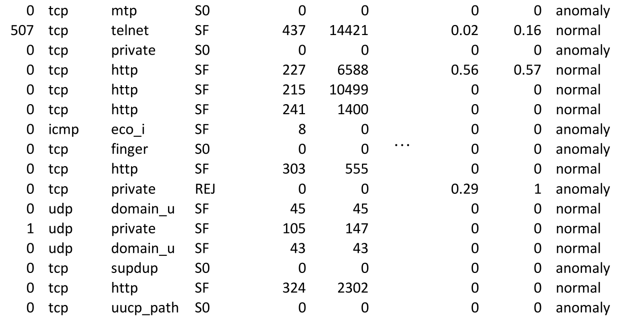
$$\hat{\beta}_r = \frac{1}{n}\sum_{i=1}^{n} y_{r_i} - \frac{\hat{\alpha}_r}{n}\sum_{i=1}^{n} x_i = \bar{y}_r - \hat{\alpha}_r \bar{x} \tag{9}$$

where $\bar{x}$ and $\bar{y}_r$ are the means of x_i and y_{r_i} observations, respectively.

Algorithm 2 RUPred-LR: Resource Utilization Prediction Based on Linear Regression

```
Input: $History\_X_z = \bigcup_{i=1}^{n}\{x_i, y_{r_i} = Util_r\}$
Output: Predicted $Util_r$

$\bar{x}$ = Mean of $\{x_1, x_2, ..., x_n\}$
$\bar{y}_r$ = Mean of $\{y_{r_1}, y_{r_2}, ..., y_{r_n}\}$
Initialize $l = 0, m = 0$
for $i = 1$ to $n$ do
    $l \mathrel{+}= (x_i - \bar{x})(y_{r_i} - \bar{y}_r)$
    $m \mathrel{+}= (x_i - \bar{x})^2$
end for
$\hat{\alpha}_r = \frac{l}{m}$                          ▷ Equation (8)
$\hat{\beta}_r = \bar{y}_r - \hat{\alpha}\bar{x}$                 ▷ Equation (9)
Predicted $Util_r = \hat{\alpha}x + \hat{\beta}$          ▷ Equation (3)
Return Predicted $Util_r$
```



0	tcp	mtp	S0	0	0		0	0	anomaly
507	tcp	telnet	SF	437	14421		0.02	0.16	normal
0	tcp	private	S0	0	0		0	0	anomaly
0	tcp	http	SF	227	6588		0.56	0.57	normal
0	tcp	http	SF	215	10499		0	0	normal
0	tcp	http	SF	241	1400		0	0	normal
0	icmp	eco_i	SF	8	0		0	0	anomaly
0	tcp	finger	S0	0	0	···	0	0	anomaly
0	tcp	http	SF	303	555		0	0	normal
0	tcp	private	REJ	0	0		0.29	1	anomaly
0	udp	domain_u	SF	45	45		0	0	normal
1	udp	private	SF	105	147		0	0	normal
0	udp	domain_u	SF	43	43		0	0	normal
0	tcp	supdup	S0	0	0		0	0	anomaly
0	tcp	http	SF	324	2302		0	0	normal
0	tcp	uucp_path	S0	0	0		0	0	anomaly

Fig. 3. Portion of the NSL-KDD data set.

VI. Evaluation Methodology

In this section, we provide brief description about the used data set, the FL setup, how data is distributed over clients, the architecture of the global model, the baseline approaches, an example scenario of how the clients are selected, and the evaluation criteria.

A. About the Data Set

As a case study, we are interested in network intrusion detection, where good models can distinguish between good/normal connections and intrusions/attacks. For this task, we adopt the publicly available NSL-KDD [5] data set, which consists of 125 973 training and 22 544 testing samples, with 41 features, such as "duration," "protocol_type," "src_bytes," "dst_bytes," etc. Fig. 3 shows a snippet of the data set, where data is classified either as normal or as anomaly. In the typical FL scenarios, labeling the data set could be performed naturally from user interaction. While such technique can be easily done for image classification or language models, it is not the case for intrusion detection system. Since the latter is our case study, we've distributed the already labeled data on the clients. However, we can consider some exiting approaches [42]–[44] or frameworks [45], [46] for future directions when dealing with new data set in order to automatically label it.

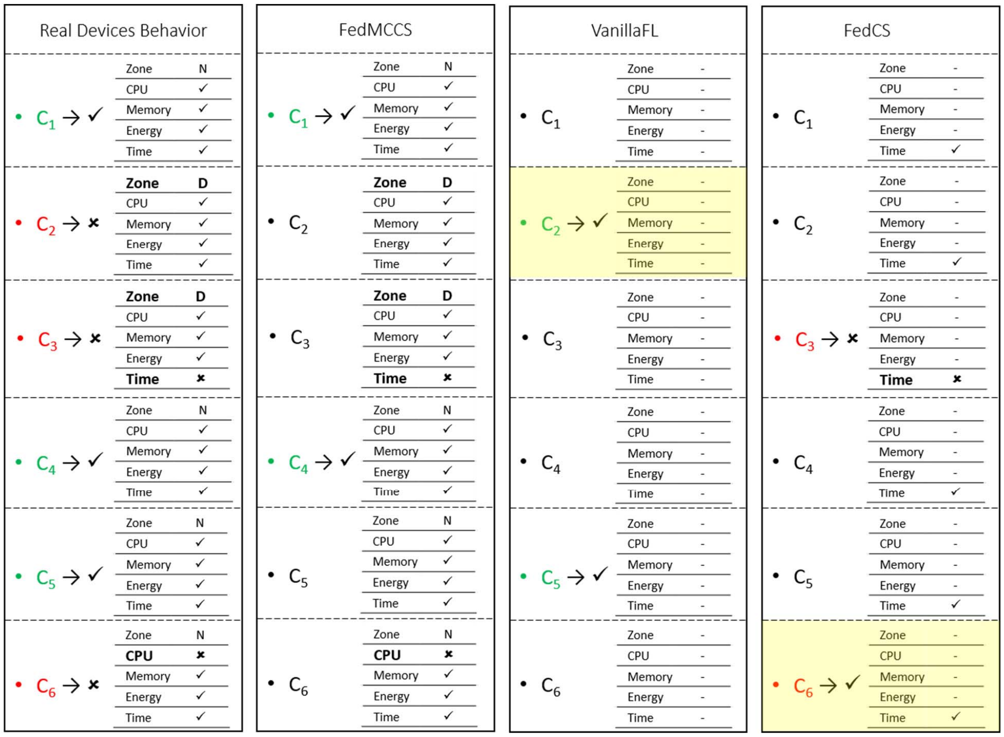

Real Devices Behavior
C1 → ✓ Zone N CPU ✓ Memory ✓ Energy ✓ Time ✓
C2 → × Zone D CPU ✓ Memory ✓ Energy ✓ Time ✓
C3 → × Zone D CPU ✓ Memory ✓ Energy ✓ Time ×
C4 → ✓ Zone N CPU ✓ Memory ✓ Energy ✓ Time ✓
C5 → ✓ Zone N CPU ✓ Memory ✓ Energy ✓ Time ✓
C6 → × Zone N CPU × Memory ✓ Energy ✓ Time ✓
FedMCCS
C1 → ✓ Zone N CPU ✓ Memory ✓ Energy ✓ Time ✓
C2 Zone D CPU ✓ Memory ✓ Energy ✓ Time ✓
C3 Zone D CPU ✓ Memory ✓ Energy ✓ Time ×
C4 → ✓ Zone N CPU ✓ Memory ✓ Energy ✓ Time ✓
C5 Zone N CPU ✓ Memory ✓ Energy ✓ Time ✓
C6 Zone N CPU × Memory ✓ Energy ✓ Time ✓
VanillaFL
C1 Zone - CPU - Memory - Energy - Time -
C2 → ✓ Zone - CPU - Memory - Energy - Time -
C3 Zone - CPU - Memory - Energy - Time -
C4 Zone - CPU - Memory - Energy - Time -
C5 → ✓ Zone - CPU - Memory - Energy - Time -
C6 Zone - CPU - Memory - Energy - Time -
FedCS
C1 Zone - CPU - Memory - Energy - Time ✓
C2 Zone - CPU - Memory - Energy - Time ✓
C3 → × Zone - CPU - Memory - Energy - Time ×
C4 Zone - CPU - Memory - Energy - Time ✓
C5 Zone - CPU - Memory - Energy - Time ✓
C6 → ✓ Zone - CPU - Memory - Energy - Time ✓

Fig. 4. How the clients are selected in the three different approaches based on their metrics when $\lceil K \times C \rceil = 2$.

B. Federated Learning Setup

We implemented FL and the proposed Client Selection algorithm using, on the server side, *flask*[1] and *flask_socketio*[2] libraries and, on the client side, *socketIO-client*.[3] Regarding the implementation of the deep neural network and its parameters, we used Keras[4] with TensorFlow backend. In these experiments, the clients are Raspberry Pi devices, having quad-core processor of 1.2 GHz, 1 GB of RAM, 16 GB of storage, and Rasberian Debian OS.

C. Data Distribution Over Clients

We study the following partitioning of NSL-KDD data set: First, we set K to 100 clients. Then, we assign between 100 and 2500 samples of the training set to each client. Since we are investigating binary classification problem, the training samples are labeled as either normal or abnormal. However, the latter was initially classified into DoS, Probe, R2L and U2R attacks with 45927, 11656, 995, and 52 samples, respectively. With few training sets available for the R2L and U2R-based attacks, small number of clients are assigned with such type of data, while the majority have a combination of DoS and Probe attacks. We consider that this partition follows a non-iid data distribution fashion, as different clients have different data partition. As for the testing set, it is used on the server side to measure the classification performances. In each FL round, C is set to 0.1 as in [2] and [3], where a maximum of ten clients may be selected.

D. Global Model Architecture

Before generating the global model and sharing it among the clients, we have built a centralized model, where the full training set has been used. We kept tuning the hyper-parameters until the test model didn't improve anymore. Such best network structure was considered as the global model, where a deep neural network [47] of three fully connected layers are used as follows: 1) 288 units with tanh activation; 2) 120 units activated by ReLu; and 3) a final sigmoid output layer. This results in a total of 70 058 model parameters. Using such architecture, the selected clients train the model in each round with ten mini-batches and five epochs.

E. Baseline Approaches

We compared our approach, FedMCCS, to the following two baselines.

1) The original FL approach [2] presented in Protocol 1, where $\lceil K \times C \rceil$ clients are randomly selected by the server to participate in the FL rounds. We refer to this baseline as VanillaFL.
2) FL with client selection [3] presented in Protocol 2, where random $\lceil K \times C \rceil$ clients are first selected by the server to share their resources. The latter, representing mainly the time needed to obtain clients' updates, is used for client selection. We refer to this baseline as FedCS.

For all the conducted experiments, we ran each of the three approaches five times and considered their averages in order to do the analysis and the comparison.

F. Client Selection Example Scenario

Based on the considered metrics in each approach, Fig. 4 shows how the clients are selected when, for instance, $K = 6$,

[1] https://www.fullstackpython.com/flask.html
[2] https://flask-socketio.readthedocs.io/en/latest/
[3] https://github.com/socketio/socket.io-client
[4] https://keras.io/

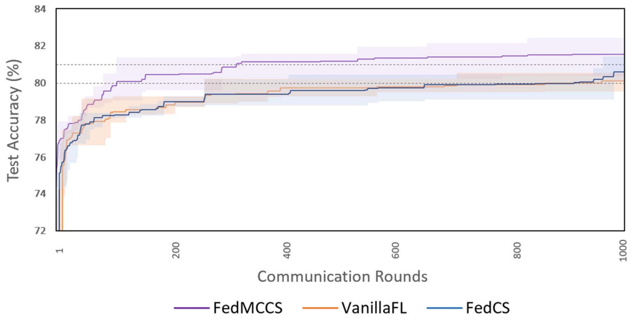


Fig. 5. Test accuracy versus communication rounds.

and $\lceil K \times C \rceil = 2$. The first column indicates the real devices behavior, where 3 out of 6 clients are able to successfully finish their training task. When applying the `Client Selection` step in FL, FedMCCS starts with the stratified method to filter the clients based on their zone, then requests from the clients their resources for the selection prediction. By choosing C_1 and C_4 for training, FedMCCS succeeds in receiving the needed model parameters in the ongoing FL round. As for VanillaFL, the complete random selection of clients is the rule of luck in receiving a number of updates close to $\lceil K \times C \rceil$. For instance, if C_2 and C_5 are selected, only C_5 fails to send its updates to the server. Regarding FedCS approach, it first selects random $\lceil K \times C \rceil$ clients. If the ones selected are C_3 and C_6, then its selection algorithm will decide to continue only with C_6, since C_3 has a time constraint.

G. Evaluation Criteria

The test data set that we are dealing with is somehow balanced, with a distribution of 57% and 43% for the abnormal and normal samples, respectively. Therefore, we rely on the accuracy only rather than other metrics, e.g., F1 score and recall rates to evaluate and compare the models performance. Our FedMCCS model, in addition to VanillaFL and FedCS, are evaluated to answer the following questions.

1) How many communication rounds are needed to achieve a desired accuracy?
2) How many selected clients are able to finish training, without dropping out?
3) How many FL rounds are discarded in the process?
4) How the network traffic is varying during the process?

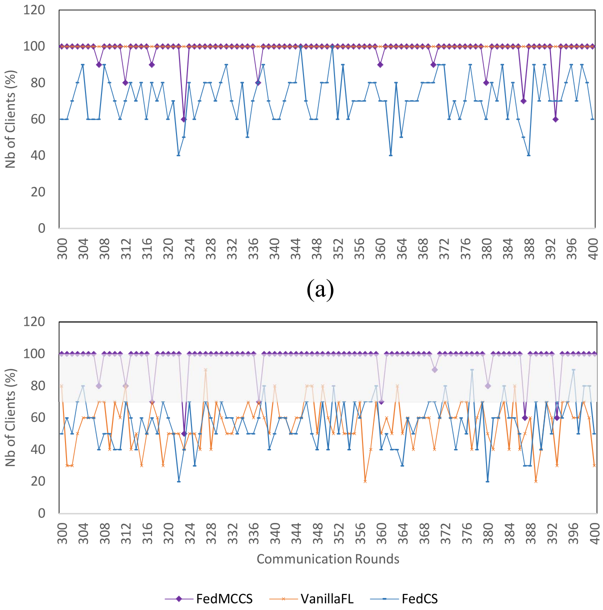
VII. Experimental Results

In this section, we discuss the results of the conducted experiments.

A. Test Accuracy Throughout Communication Rounds

TABLE I
Number of Communication Rounds Needed in the Three Different Approaches to Reach the 80% and 81% Accuracy

Approaches	**80% Accuracy**	**81% Accuracy**
FedMCCS	108	319
VanillaFL	861 (8.0×)	-
FedCS	912 (8.4×)	-



(a)

(b)

Fig. 6. Variation of the number of selected clients for round 300 till 400. (a) # of clients slelected out of $\lceil K \times C \rceil$. (b) # of performant clients.

We start by answering question 1: "How many communication rounds are needed to achieve a desired accuracy?" The goal is to minimize the number since the upload bandwidth is typically limited and the clients are expected to participate in small number of rounds per day [2]. Fig. 5 shows learning curves over 1000 rounds for FedMCCS, VanillaFL, and FedCS. Each curve is improved by taking the maximum test accuracy reached among the prior rounds. The dotted lines in the figure shows the accuracy at 80% and 81%, representing our targets, which were attained in centralized settings. As for the shaded areas, they represent the standard deviation of five executions for the three approaches. We calculate the number of rounds when the dotted lines cross the curves, and Table I quantifies the results. With the same data distribution over the clients and the same used model architecture, our FedMCCS outperforms the other two approaches in terms of communication rounds. To reach 80% accuracy, FedMCCS needs 108 rounds, while this number increases by 8.0× for the VanillaFL, and by 8.4× for the FedCS. As for the second target of 81% accuracy, FedMCCS was able to reach it using 319 rounds, while the other approaches were not able to achieve such model accuracy throughout the 1000 rounds. We interpret such result by the fact that we have more clients participating in the FL rounds: The more clients we have, the more accuracy we get, and the smaller number of rounds we perform. We detail this observation in the next section, while highlighting the selected number of clients throughout the FL process.

B. Number of Clients Selected Throughout FL Process

Moving to question 2: "How many selected clients are able to finish training, without dropping out?", Fig. 6 depicts the number of clients throughout the communication rounds. For

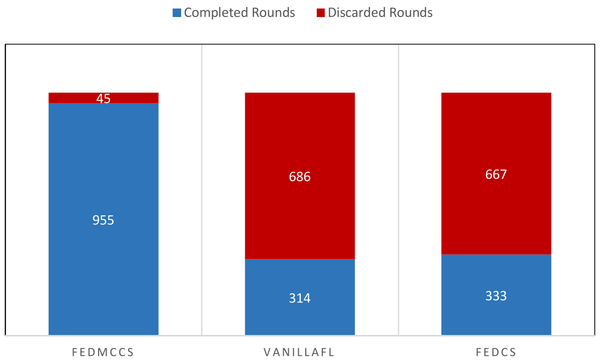


Fig. 7. Completed versus discarded rounds in 1000 FL rounds.

better visual readability, we display in the figure the results for rounds 300 till 400 only. First, we show in Fig. 6(a) how many clients each client selection algorithm in the three approaches chooses to request updates. For each round, VanillaFL selects $\lceil K \times C \rceil$ clients at random. This results in a horizontal line for the 100% selection of clients. As for FedCS, among the random selection of $\lceil K \times C \rceil$ clients, the ones satisfying the time constraint are selected; therefore, a selection of 100% of the clients barely happens. On the other hand, our proposed FedMCCS filters the clients based on their time zone, contacts them for their resources, then predicts the ones able to participate in the FL rounds. The target number of clients to be selected might not be able to be reached sometimes, which is normal as no enough clients would be available to volunteer in some FL rounds.

In all the studied approaches, some of the selected clients will not be able to answer back with the model updates. This depends on the amount of data each device has, and the devices computation and communication resources. FedCS and VanillaFL, which consider the time metric or no constraints at all respectively, yield to many devices dropouts and few number of updates sent by the clients as shown in Fig. 6(b). However, FedMCCS selects the highest number of performant clients, by considering the criteria (CPU, Memory, and Energy) affecting the dropouts within the time threshold. The gray area in the figure shows the range (70%-100%) of responsive clients, which the server can tolerate to still perform the updates aggregation.

C. Number of Discarded Rounds in FL

As previously mentioned, when less than 70% of the selected clients do not send their model parameters to the server, the ongoing round is discarded. To answer question 3: "How many FL rounds are discarded in the process?", Fig. 7 shows, in the three approaches, the number of completed/discarded rounds out of the 1000. Our FedMCCS results in only few discarded rounds, compared to the two others, which have more than half the rounds without model aggregation. This is justified from the results of Fig. 6, that show how the majority of the clients were misselected in FedMC and VanillaFL.

D. Network Traffic Variation During FL Process

To evaluate question 4: "How the network traffic is varying during the process?", we analyzed the load in the network by studying the network traffic when sending and receiving some amount of data from/to the server/clients. In Fig. 8(a), VanillaFL shows the least amount of data moving across the network as no resource related data is shared between the components prior to the updates requests. FedCS shows higher traffic due to the exchanged messages for the time-based resource request, in addition to the updates messages. On the other hand, our proposed FedMCCS approach has peak values every 100 rounds, referring to the historical data sent by the filtered clients (after the stratified method). Only when significant change in the resources, which may affect the FL rounds, take place, the server gets notified from the clients. Therefore, we did the analysis assuming that such modifications happen every 100 rounds. The historical data set is usually of small size, which is around 0.48-kB per client. Although the cost is negligible, sending the historical data every 100 rounds helps minimizing the communication cost in our proposed approach. Then, the shared historical data is used, in the remaining rounds, to directly select the expected clients that can well perform in the training. This justifies the less network traffic generated in FedMCCS compared to FedCS, apart from the peak values.

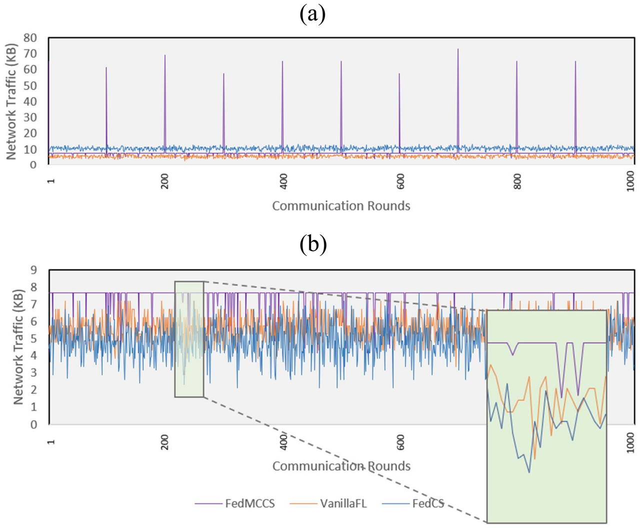


Fig. 8. Variation of network traffic in the FL process. (a) Full network traffic. (b) Network traffic for parameters updates

In Fig. 8(b), we shows the amount of data transmitted only when sending requests and receiving responses in the `Distribution` and `Update and Upload` steps in Protocols 1–3. Our FedMCCS maximizes the number of clients by reaching $\lceil K \times C \rceil$. For this reason, we have the highest network traffic compared to the other approaches. However, the difference is negligible, since the average network traffic for FedMCCS is around 7.6-kB per round, compared to 5.7 kB for VanillaFL, and 4.9 kB for FedCS.

Among the full network traffic represented in Fig. 8, some are wasted due to: 1) the requests sent to misselected clients, and 2) the requests/responses successfully sent from/to the clients in a discarded round. As shown in Fig. 9, our FedMCCS reports the least wasted traffic, since the resources related data is used to efficiently select the correct clients resulting

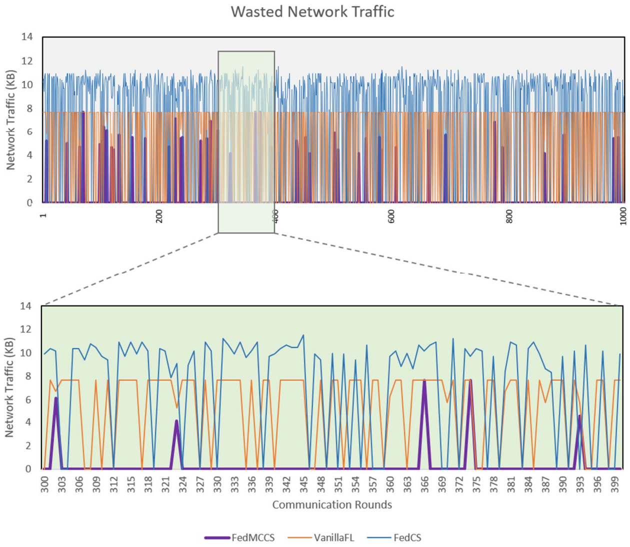


Fig. 9. Amount of wasted network traffic in the FL process.

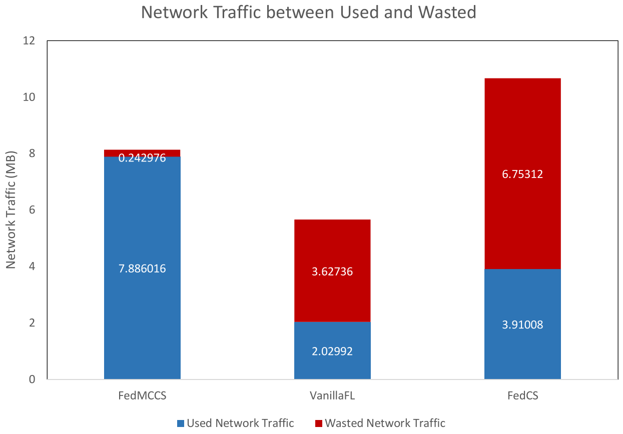


Fig. 10. Total amount of used and wasted network traffic in the 1000 FL rounds.

in only 45 discarded rounds out of 1000 (results of Fig. 7). This is not the case for FedCS and VanillaFL approaches, where they show many discarded rounds. Moreover, FedCS has more wasted traffic because of the first random selection of the $\lceil K \times C \rceil$ clients for resource requests.

For better vision of how much traffic is wasted in each approach, we illustrated in Fig. 10 the total used and wasted traffic in the 1000 rounds. Although FedMCCS has more traffic than Vanilla FL, but the wasted is negligible with considerable decrease in the communication rounds to reach the target accuracy (as shown in the results of Fig. 5).

VIII. Conclusion

We proposed in this article FedMCCS; an enhanced FL with multicriteria client selection. Particularly, we proposed a bilevel optimization scheme, which can efficiently select and maximize the number of clients to participate in each of the FL rounds, while considering their heterogeneity and their limited communication and computation resources. The approach also leverages stratified-based sampling to filter the available set of clients while implementing an efficient client selection algorithm based on multicriteria, including CPU, memory, energy, and time.

Real life experiments explore that FedMCCS is able to train and produce high-performant ML models with few rounds of communication, compared to the state-of-the-art protocols and approaches. This was achieved by selecting the maximum number of clients in each FL round, while benefiting, at most, from their computation resources, energy, and the network traffic shared between the server and the clients.

As future direction, we tend to offer a solution that studies how efficient is the update of each client will be to the global model in order to participate in the FL process. This can significantly save resources on the other clients that will not have to contribute to the global model training.

References

[1] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Mar. 2017.

[2] 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. AISTATS*, 2016, pp. 1273–1282.

[3] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[4] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," 2019. [Online]. Available: arXiv:1902.01046.

[5] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD cup 99 data set," in *Proc. IEEE Symp. Comput. Intell. Security Defense Appl.*, 2009, pp. 1–6.

[6] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 1175–1191.

[7] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Edge-assisted hierarchical federated learning with non-IID data," 2019. [Online]. Available: arXiv:1905.06641.

[8] T. S. Brisimi *et al.*, "Federated learning of predictive models from federated electronic health records," *Int. J. Med. Informat.*, vol. 112, pp. 59–67, Apr. 2018.

[9] D. Preuveneers, V. Rimmer, I. Tsingenopoulos, J. Spooren, W. Joosen, and E. Ilie-Zudor, "Chained anomaly detection models for federated learning: An intrusion detection case study," *Appl. Sci.*, vol. 8, no. 12, p. 2663, 2018.

[10] M. J. Sheller, G. A. Reina, B. Edwards, J. Martin, and S. Bakas, "Multi-institutional deep learning modeling without sharing patient data: A feasibility study on brain tumor segmentation," in *Proc. Int. MICCAI Brainlesion Workshop*, 2018, pp. 92–104.

[11] R. C. Geyer, T. Klein, and M. Nabi, "Differentially private federated learning: A client level perspective," 2017. [Online]. Available: arXiv:1712.07557.

[12] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proc. IEEE Symp. Security Privacy (SP)*, May 2019, pp. 691–706. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2019.00029

[13] F. Fioretto and P. Van Hentenryck, "Privacy-preserving federated data sharing," in *Proc. 18th Int. Conf. Auton. Agents MultiAgent Syst.*, 2019, pp. 638–646.

[14] S. Truex *et al.*, "A hybrid approach to privacy-preserving federated learning," in *Proc. 12th ACM Workshop Artif. Intell. Security*, 2019, pp. 1–11.

[15] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM)*, 2019, pp. 2512–2520.

[16] H. Li and T. Han, "An end-to-end encrypted neural network for gradient updates transmission in federated learning," in *Proc. Data Compression Conf. (DCC)*, Mar. 2019, p. 589.

[17] J. Konecný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. (2016). *Federated Learning: Strategies for Improving Communication Efficiency*. [Online]. Available: http://arxiv.org/abs/1610.05492.

[18] S. Caldas, J. Konečny, H. B. McMahan, and A. Talwalkar. (2018). *Expanding the Reach of Federated Learning by Reducing Client Resource Requirements*. [Online]. Available: arXiv:1812.07210.

[19] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and communication-efficient federated learning from non-IID data," 2019. [Online]. Available: arXiv:1903.02891.

[20] Y. Chen, X. Sun, and Y. Jin, "Communication-efficient federated deep learning with asynchronous model update and temporally weighted aggregation," 2019. [Online]. Available: http://arxiv.org/abs/1903.07424.

[21] D. Leroy, A. Coucke, T. Lavril, T. Gisselbrecht, and J. Dureau, "Federated learning for keyword spotting," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2019, pp. 6341–6345.

[22] W. Luping, W. Wei, and L. Bo, "CMFL: Mitigating communication overhead for federated learning," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2019, pp. 954–964.

[23] J. Qiu, D. Grace, G. Ding, M. D. Zakaria, and Q. Wu, "Air-ground heterogeneous networks for 5G and beyond via integrating high and low altitude platforms," *IEEE Wireless Commun.*, vol. 26, no. 6, pp. 140–148, Jan. 2019.

[24] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," 2019. [Online]. Available: arXiv:1909.07972.

[25] H. H. Yang, Z. Liu, T. Q. S. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE Trans. Commun.*, vol. 68, no. 1, pp. 317–333, Jan. 2020.

[26] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Convergence time optimization for federated learning over wireless networks," 2020. [Online]. Available: arXiv:2001.07845.

[27] J. Ren, Y. He, D. Wen, G. Yu, K. Huang, and D. Guo, "Scheduling in cellular federated edge learning with importance and channel awareness," 2020. [Online]. Available: arXiv:2004.00490.

[28] T. Dbouk, A. Mourad, H. Otrok, H. Tout, and C. Talhi, "A novel ad-hoc mobile edge cloud offering security services through intelligent resource-aware offloading," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1665–1680, Sep. 2019.

[29] H. Tout, C. Talhi, N. Kara, and A. Mourad, "Selective mobile cloud offloading to augment multi-persona performance and viability," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 314–328, Apr. 2019.

[30] H. Sami and A. Mourad, "Dynamic on-demand fog formation offering on-the-fly IoT service deployment," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 2, pp. 1026–1039, Jun. 2020.

[31] Y. Lu, X. Huang, Y. Dai, S. Maharjan, and Y. Zhang, "Blockchain and federated learning for privacy-preserved data sharing in industrial IoT," *IEEE Trans. Ind. Informat.*, vol. 16, no. 6, pp. 4177–4186, Jun. 2020.

[32] J. Mills, J. Hu, and G. Min, "Communication-efficient federated learning for wireless edge intelligence in IoT," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 5986–5994, Jul. 2020.

[33] F. Hartmann. (2018). *Federated Learning*. Accessed: Feb. 20, 2020. [Online]. Available: http://www.mi.fu-berlin.de/inf/groups/ag-ti/theses/download/Hartmann_F18.pdf

[34] M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *Oper. Res. Lett.*, vol. 32, no. 1, pp. 41–43, 2004.

[35] A. Krause and D. Golovin, "Submodular function maximization," in *Tractability: Practical Approaches to Hard Problems*. Cambridge, U.K.: Cambridge Univ. Press, Feb. 2014.

[36] Q. Yu, L. Xu, and S. Cui, "Streaming algorithms for news and scientific literature recommendation: Monotone submodular maximization with a *d*-knapsack constraint," *IEEE Access*, vol. 6, pp. 53736–53747, 2018.

[37] M. Duan *et al.*, "Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications," in *Proc. IEEE 37th Int. Conf. Comput. Design (ICCD)*, 2019, pp. 246–254.

[38] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," 2019. [Online]. Available: arXiv:1902.00146.

[39] G. A. Seber and A. J. Lee, *Linear Regression Analysis*, vol. 329. New York, NY, USA: Wiley, 2012.

[40] H. Tout, N. Kara, C. Talhi, and A. Mourad, "Proactive machine learning-based solution for advanced manageability of multi-persona mobile computing," *Comput. Elect. Eng.*, vol. 80, Art. no. 106497, Jan. 2019.

[41] P. Farhat, H. Sami, and A. Mourad, "Reinforcement R-learning model for time scheduling of on-demand fog placement," *J. Supercomput.*, vol. 76, pp. 1–23, Oct. 2019.

[42] J. Kim, C. Sim, and J. Choi, "Generating labeled flow data from mawilab traces for network intrusion detection," in *Proc. ACM Workshop Syst. Netw. Telemetry Anal.*, 2019, pp. 45–48.

[43] R. K. Rakesh *et al.*, "An autonomous labeling pipeline for intrusion detection on enterprise networks," in *Proc. IEEE Syst. Inf. Eng. Design Symp. (SIEDS)*, 2019, pp. 1–6.

[44] R. Fontugne, P. Borgnat, P. Abry, and K. Fukuda, "MAWILab: combining diverse anomaly detectors for automated anomaly labeling and performance benchmarking," in *Proc. 6th Int. Conf.*, 2010, pp. 1–12.

[45] N. Rajasinghe, J. Samarabandu, and X. Wang, "INSECS-DCS: A highly customizable network intrusion dataset creation framework," in *Proc. IEEE Can. Conf. Elect. Comput. Eng. (CCECE)*, 2018, pp. 1–4.

[46] F. J. Aparicio-Navarro, K. G. Kyriakopoulos, and D. J. Parish, "Automatic dataset labelling and feature selection for intrusion detection systems," in *Proc. IEEE Mil. Commun. Conf.*, 2014, pp. 46–51.

[47] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2805–2824, Jan. 2019.

Sawsan AbdulRahman received the master's degree in computer science from the Lebanese American University, Beirut, Lebanon, in 2017. She is currently pursuing the Ph.D. degree with École de Technologie Supérieure, Montreal, QC, Canada.

She is working in collaboration with Ericsson, Montreal, in the area of AI, machine learning, and security.

Mrs. AbdulRahman is a Reviewer in several prestigious journals and conferences.

Hanine Tout received the Ph.D. degree in software engineering from the École de Technologie Supérieure (ÉTS), Montreal, QC, Canada, in 2018.

She is a Postdoctoral Fellow with ÉTS and Ericsson, Montreal, QC, Canada., where she is leading two industrial projects in the areas of AI, federated learning, machine learning, security, 5G, and cloud-native IMS.

Dr. Tout is the TPC Member and a Reviewer of prestigious conferences and journals.

Azzam Mourad (Senior Member, IEEE) received the M.Sc. degree in CS from Laval University, Québec, QC, Canada, in 2003, and the Ph.D. degree in ECE from Concordia University, Montréal, QC, in 2008.

He is currently an Associate Professor of computer science with the Lebanese American University, Beirut, Lebanon, and an Affiliate Associate Professor with the Software Engineering and IT Department, ÉTS, Montreal. He published more than 100 papers in international journal and conferences on security, network and service optimization and management targeting IoT, cloud/fog/edge computing, vehicular and mobile networks, and federated learning.

Dr. Mourad has served/serves as an Associate Editor for IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, IEEE NETWORK, IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY, *IET Quantum Communication*, and IEEE COMMUNICATIONS LETTERS, the General Chair of IWCMC2020, the General Co-Chair of WiMob2016, and the Track Chair, a TPC member, and a reviewer for several prestigious journals and conferences.

Chamseddine Talhi received the Ph.D. degree in computer science from Laval University, Quebec, QC, Canada, in 2007.

He is an Associate Professor with the Department of Software Engineering and IT, ÉTS, University of Quebec, Montreal, QC, Canada. He is leading a research group that investigates smartphone, embedded systems, and IoT security. His research interests include cloud security and secure sharing of embedded systems.