Artificial Intelligence of Things at the Edge: Scalable and Efficient Distributed Learning for Massive Scenarios[⋆]

Saira Bano[a,b,*], Nicola Tonellotto[a], Pietro Cassarà[b] and Alberto Gotta[b]

[a]Department of Information Engineering, University of Pisa, Pisa, Italy
[b]Information Science and Technologies Institute "A. Faedo", National Research Council of Italy, Pisa, Italy

ARTICLE INFO

Keywords:
6G
Federated Learning
In-Network Caching
Publish/Subscribe
Clients Selection
Edge Computing
Information-Centric Networking

ABSTRACT

Federated Learning (FL) is a distributed optimization method in which multiple client nodes collaborate to train a machine learning model without sharing data with a central server. However, communication between numerous clients and the central aggregation server to share model parameters can cause several problems, including latency and network congestion. To address these issues, we propose a scalable communication infrastructure based on Information-Centric Networking built and tested on Apache Kafka®. We design an algorithm for effective client participation in FL within the proposed architecture. The proposed architecture consists of a two-tier communication model. In the first layer, client updates are cached at the edge between clients and the server, while in the second layer, the server computes global model updates by aggregating the cached models. The data stored in the intermediate nodes at the edge enables reliable and effective data transmission and solves the problem of intermittent connectivity of mobile nodes. While many local model updates provided by clients can result in a more accurate global model in FL, they can also result in massive data traffic that negatively impacts congestion at the edge. Therefore, it is necessary to avoid congestion at the edge and the resulting transmission delays to the cloud server. For this reason, we couple a client selection procedure based on a congestion control mechanism at the edge for the given architecture of FL. The proposed algorithm selects a subset of clients based on their resources through a time-based backoff system to account for the time-averaged accuracy of FL while limiting the traffic load. Experiments show that our proposed architecture has an improvement of over 40% over the network-centric based FL architecture, i.e., Flower. The architecture also provides scalability and reliability in the case of mobile nodes. It also improves client resource utilization, avoids overflow, and ensures fairness in client selection. The experiments show that the proposed algorithm leads to the desired client selection patterns and is adaptable to changing network environments.

1. Introduction

While 5G mobile systems are still actively deployed worldwide, the research community has already begun investigating the latest technological developments for the next generation of 6G mobile systems [1]. It is widely believed that 6G systems will foster a novel, pervasive concept of Artificial Intelligence (AI), leading to a hyper-flexible architecture that integrates human-like intelligence into every component of mobile network systems [2]. The proliferation of Artificial Intelligence (AI) applications and Internet-of-Things (IoT) devices continues to drive the evolution of mobile network systems, and 6G is anticipated to shift the paradigm of mobile communications systems from *"connected things"* to *"connected intelligence"* [3]. In this context, one of the primary goals of 6G mobile systems is to orchestrate communications, computation, and control as components of a holistic system to achieve better sustainability and energy efficiency [4]. In this 6G eco-system, ubiquitous AI-based technologies and Machine Learning (ML) are essential in optimizing the orchestration of the above components. Traditionally, ML systems are trained by collecting the required data on a single server, but this approach raises several issues, such as data reliability and confidentiality. In addition, centralized training of ML models with data collected from multiple devices increases communication costs and latency. Numerous applications such as unmanned aerial systems, virtual reality, and connected and autonomous vehicles (CAVs) have stringent latency and privacy requirements [5]. Therefore, using centralized ML approaches to optimize these new applications is not practical. Furthermore, data security and

*Corresponding author

Email addresses: saira.bano@phd.unipi.it (S. Bano); nicola.tonellotto@unipi.it (N. Tonellotto); pietro.cassara@isti.cnr.it (P. Cassarà); alberto.gotta@isti.cnr.it (A. Gotta)
ORCID(s): 0000-0001-8126-4638 (S. Bano); 0000-0002-3704-4133 (P. Cassarà); 0000-0002-8134-7844 (A. Gotta)

user privacy are among the most critical requirements that 6G systems for smart applications and services must meet, and the General Data Protection Regulation (GDPR) discourages the acquisition and transfer of user data [6].

Federated Learning (FL) is a new approach in which ML algorithms are trained on multiple decentralized devices, with all data remaining on the device. In the FL process, a group or subset of clients independently trains an ML model with their local data. These clients then exchange the parameters of the local ML models with a Model Aggregation Server (MAS), which averages these model parameters to create a global ML model. The MAS then propagates this global ML model to all participating clients [7]. The FL process consists of multiple rounds of communication between client devices and the MAS. This eliminates the need for a central repository for shared data, as required by traditional centralized ML training. Although still at an early stage of development in real-world scenarios, distributed learning architectures inspired by FL are already seen as one of the most promising ways to achieve the goal of ubiquitous AI in 6G systems [8]. However, implementing FL on resource-constrained client devices is difficult in real-world scenarios because the parameters describing ML models, such as neural networks (NNs), can be as large as a billion in many applications. Nevertheless, several FL frameworks have been proposed to develop FL-based applications, such as TensorFlow Federated [9], FedEval [10], FedML [11], PySyft [12], and Flower [13].

These frameworks are based on a network-centric communication paradigm that requires direct communication between clients and servers. The network-centric paradigm relies on a central server that is a single point of failure and cannot provide reliability for ubiquitous AI applications for 6G systems [1]. Therefore, clients cannot send or receive data to or from the server if the connection fails or the server is unavailable, especially for mobile or vehicular nodes with limited connectivity. In contrast, a communication paradigm such as Information-Centric Network (ICN) can ensure that information is exchanged between two endpoints (clients and server) even in the event of failures by caching and replicating data within the network. The expected benefits of this ICN paradigm are higher efficiency, better scalability in terms of bandwidth requirements, and higher robustness under challenging communication conditions, such as in vehicular scenarios. It should be noted that ICN still requires a TCP/IP connection for data transmission, although some solutions have been proposed based on other, more efficient protocols such as QUIC [14]. Therefore, ICN is generally considered an overlay network built on ossified network-centric protocols.

Exploiting the benefits of the ICN paradigm, we design a scalable communication architecture of FL for a vehicular scenario. This communication architecture aims to support mobile nodes where connectivity is intermittent by providing in-network caching capabilities. We develop this communication paradigm using Apache Kafka®[15], a distributed data streaming platform. In the FL communication paradigm, the Kafka entity called broker is configured at the edge as a cache for model parameters from clients or for global model parameters from MAS. The model parameters are managed at the broker as an event or message organization structure called *topics*. The Kafka broker, acting as an intermediate instance, receives data from publishers (that generate the model parameters) in topics and distributes this data to subscribers (who use the model parameters). Hence, we decouple the client nodes from the MAS and form a many-to-many communication model between publishers and subscribers. This decoupling makes distributed systems more flexible, scalable, and resilient to connection failures than the traditional client/server paradigm [16]. Indeed, conventional network-centric communication architectures are tightly coupled and require clients and servers to be active simultaneously, resulting in a fixed one-to-one architecture that is less scalable than the ICN architecture.

In the proposed ICN-based FL communication paradigm, the large number of local model parameters sent by clients to the broker to obtain an accurate global model can lead to broker congestion. Consequently, the data queues at the distributed edge and the resulting transmission delays to the MAS need to be considered. One of the solutions could be to limit the number of participating clients in the federation process. Many client selection techniques were proposed in the literature, such as in [17],[18], but these techniques are based on static clients. They do not adequately consider client selection in dynamic vehicle scenarios. For this reason, an effective client selection algorithm (FL) needs to be developed to support the application of FL in dynamic vehicle environments. In this context, we propose a mechanism to select clients participating in global model updating in massive scenarios, i.e., with many potentially active clients, to reduce broker congestion while maintaining model accuracy. This congestion control mechanism limits the number of client updates received to avoid queue overflow. An early design of this client selection mechanism, namely FedTCS, was presented in [19] using a time-based approach with exponentially distributed timers. In this paper, we extend the analysis of FedTCS by giving a complete account of the selection techniques for FL clients and testing different time distributions, such as the uniform or beta distribution. In addition, we consider clients with heterogeneous resources and training times.

Through extensive experiments, we demonstrate the numerous benefits of the proposed infrastructure: better scalability for FL scenarios, temporal and spatial decoupling between clients and MAS, reliability and availability of data while avoiding overflowing broker queues. Finally, we prove the effectiveness of our information-centric paradigm by comparing its performance with that of a network-centric paradigm, i.e., Flower [13]. We also present a performance analysis to demonstrate the scalability of the proposed client selection method, which uses an equal selection probability to accommodate stragglers during the FL process while optimally matching the expected number of clients to the queue size.

The remainder of this paper is organised as follows. Section 2 introduces recent advances in FL communication and Apache Kafka®, and the background of FL client selection techniques. Section 3 presents the reference scenario and the considered assumptions of the proposed architecture. Then, Section 4 describes the proposed ICN-based FL communication architecture. Section 5 presents the client selection procedure using the architecture described in Section 4. Then, Section 6 evaluates the performance of the proposed architecture along with the client selection mechanism. Finally, conclusions are drawn in Section 7, including possible directions for future work.

2. Related Work

This section briefly reviews the work related to the proposed communication architecture and client selection process. In this context, we discuss the background, related work, and communication perspective of Apache Kafka®, a publish/subscribe model and relevant existing work on client selection in FL.

2.1. Federated Learning

Federated Learning is a distributed learning system proposed by Google [7] that trains ML models on distributed cellular phones. The main idea was to protect user privacy by keeping all data on users' devices and sharing only the ML model parameters during the training process. While FL solves the privacy problem, numerous challenges arise when using FL in real-world scenarios [20]. These issues include the communication cost required to transfer the ML model parameters between MAS and the distributed client devices and the low computational power of the client devices. In this work, we use the publish/subscribe communication mode (pub/sub) to reduce the communication cost of FL. The communication cost of FL has been the subject of many studies. In [21], the NN quantization strategy for time series prediction reduces communication costs by minimizing the size of ML model parameters exchanged between clients and the server. In addition, a significant bottleneck in scaling distributed training also arises from the communication load on the server caused by updating multiple clients in the FL process. Several strategies have been proposed to solve this communication bottleneck, including compression [22], quantization [21], and efficient client selection [23]. The work in [24] meets the requirements of FL from the communication efficiency point of view by compressing both upstream and downstream communications with Sparse Ternary Compression (STC) and optimal Golomb encoding of ML model parameters. Their proposed method is also able to handle a large number of clients by using partial client participation in FL. However, we take a different approach by performing FL using the pub/sub communication model, while the other previously mentioned methods focus on a network-centric communication model. The main goal of the proposed approach is to develop a communication-efficient FL framework that benefits static and non-static users (mobile nodes).

2.2. Apache Kafka®

Apache Kafka®, a distributed messaging platform used for data stream processing [15]. In Apache Kafka®, messages are created by message producers (in our case, vehicles) and stored in queues called topics at the broker, where topic subscribers can then retrieve them. These topics are the queues where the messages are stored. Each topic can have numerous duplicate partitions stored on different Kafka brokers. The ability to use numerous broker instances, which gives the Kafka cluster fault-tolerance properties, is another factor that makes Kafka highly scalable and appropriate for distributed learning systems such as FL. There is comparatively little research on ML applications that use ICN and Kafka. Feraudo et al. have developed a pub/sub based selection mechanism for IoT devices to participate asynchronously in the FL process [25]. In [26], the researchers developed a method for training and inferring ML models that feed the continuous dataset directly into the ML model via the Kafka pipeline, as opposed to training ML models with static data. The authors in [27] proposed an IoT-based optimized cache setting in the ICN for mobile multimedia content using ML to reduce access time. They proposed a location prediction method and an intelligent network edge caching technique to improve the user experience. In [28], the authors developed a low-latency distributed messaging system to provide a data-centric perspective on the connected vehicles ecosystem.

In the literature, only a few works have dealt with the juxtaposition of the information-centric paradigm (pub/sub) and the network-centric paradigm. According to the authors of [29], pub/sub models are preferable for distributed real-time systems because they are inexpensive and easy to implement with polling techniques. The authors of [30] compared the request-response and pub/sub models based on many criteria, including mobility, timeliness, and adaptability. They also proposed a communication strategy that combines the two systems into a hybrid system that is network and information-centric. In their paper [31], Eugster et al. compared the typical request-response method with a pub/sub strategy in three dimensions, i.e., time, space, and synchronization decoupling.

2.3. Clients Selection

While the FL algorithm was originally based on a random and uniform selection of clients [7], this scheme proved to be biased and converged to a suboptimal minima of the convergence problem. For this reason, the notion of *unbiasedness* was introduced by the authors in [32]. Unbiased client sampling is used to optimize the original FL cost function while minimizing the number of active clients per FL round. Many other client selection schemes have also been proposed in the literature, such as in [17], where researchers attempt to solve the client selection problem with non-homogeneous resources. In their proposed framework, the Multi-Access Edge Computing (MEC) operator invites a randomly selected group of clients and asks them to report their available resources. The MEC operator selects only the clients with enough resources to complete the task in the FL process. In [18], the researchers described the client selection technique that selects only the clients that have suffered high losses in their local ML model to speed up convergence and significantly reduce communication overhead. In [33], the authors proposed FedMCCS, which considers clients' resources, including CPU, memory, time, and energy, to determine whether they are capable of participating in the FL task. FedMCCS selectively increases the number of participating clients in each FL round, taking into account the resources of each client and their ability to effectively train and deliver the required ML model updates. In this paper, we propose a client selection mechanism for FL that uses the pub/sub network model to select clients during each FL round, given constraints such as network and edge resources.

3. System Model

In this section, we provide a detailed description of the reference scenario and the assumptions we made for developing our communication infrastructure, which is suitable to achieve FL in vehicular scenarios while minimizing the communication overhead [34]. This work has two main objectives: to develop an efficient communication architecture for FL suitable for mobile nodes and to efficiently select clients using this communication paradigm to reduce broker congestion. The proposed infrastructure is based on edge computing to leverage the processing power of edge servers capable of handling distributed tasks such as FL with mobile nodes that require minimizing communication latency with the central server [35, 36].

We use the ICN communication paradigm because it provides in-network caching capabilities and decouples clients from the MAS, which increases robustness to connection failures. We assume each vehicle has a 5G radio and can communicate with the edge using a pub/sub protocol. We also assume that each vehicle has the computational resources necessary to train an NN model on local data. The MAS runs in the cloud, and the Kafka broker resides on edge. The communication flows from the vehicles (clients), which generate updated local model parameters, to the broker, which stores these models and provides them to the MAS, which aggregates them in the cloud, as shown in Figure 1. Once the aggregation of the models is complete, this global NN model is stored at the edge broker, where the vehicle clients can download it for further training.

Our infrastructure uses the Apache Kafka®broker[1] to decouple the clients and the MAS. Thus, if the MAS is not available when the vehicles send their updated local NN model parameters, the broker at the edge can store these model parameters and make them available to the MAS. Once the federated model is available, it can be sent to the broker at the edge for the vehicles to download as needed. Thus, the communication between MAS and the vehicles is done through the Kafka brokers at the edge, which forms a two-hop communication path, as shown in Figure 1. Since the broker at the edge collects model updates from clients, it can relieve MAS of much of the communication load and reduce the delay caused by clients uploading and downloading model parameters. In this distributed architecture, we defined the Kafka broker's topics as data queues. Topics serve as logical channels to separate messages from vehicles to the MAS server (uplink) and vice versa. The name of each topic is its key value, which clients use to log in to the Kafka broker. We have defined two topics on the broker: `clients_data`, to which the clients publish their model parameters, and `averaged_result`, where the MAS publishes the aggregated model. The entity (client or MAS) that wants to send a model to the topic on the broker is called a producer and uses the Kafka Producer API. In contrast, the entity that wants

[1] https://kafka.apache.org/

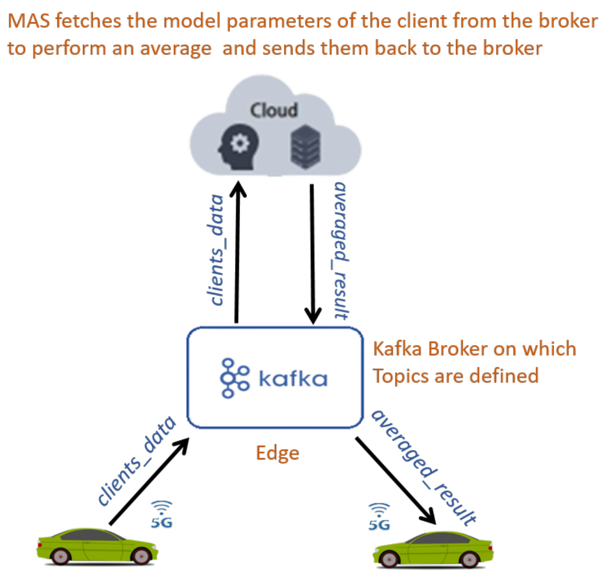

MAS fetches the model parameters of the client from the broker to perform an average and sends them back to the broker
Cloud
clients_data
averaged_result
kafka
Kafka Broker on which Topics are defined
Edge
clients_data
averaged_result
5G
5G

Figure 1: Overview of the proposed ICN-based FL communication architecture

to download the model from the broker uses the Kafka Consumer API. After developing the communication paradigm,

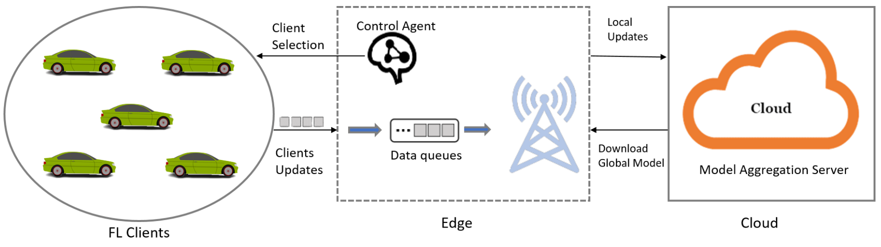


Figure 2: Overview of the client selection mechanism in FL with the control agent at the edge

we develop an efficient client selection mechanism for our infrastructure to avoid Kafka broker queue overflows and congestion caused by massive traffic of client model parameters. Even though the global ML model accuracy increases when many clients are involved, it is necessary to consider the edge resources on which the broker is deployed. Since the queue size is finite, it is important to reduce the queue overflow. The queue size depends on the arrival of model updates from clients and the departure of model updates, i.e., towards MAS. The overflow can be limited by making the departure rate higher than the arrival rate, i.e., the ratio between the average number of arrivals and the number of departures is less than one. In our proposed system, we assume that the departures are i.i.d., i.e., independent and identically distributed, and depend on the channel capacity between the edge and the MAS in the cloud. However, we can change the number of arrivals by limiting the number of participating clients in the federation process. Therefore, client selection is crucial to avoid congestion. We solve client selection by introducing a control agent (CA) at each edge that serves different regional clusters for the client selection process, as shown in Figure 2. CA uses ACKs (short messages as acknowledgements) to control the flow of model parameters to and from clients. In our architecture, we propose an activation mechanism for the CA that depends on timers, as described in the following sections. Once selected, each vehicle client trains its model over a set of local epochs and then serves as a producer that sends its local model parameters to the broker. Each client's model update is split into discrete data packets before being sent to the broker. The splitting facilitates the transmission of the models, which may even include millions of parameters in many cases. Note also that in the client selection algorithm, we refer to the topic defined in the Kafka broker as the "edge queue" because topics are stored in queues over the edge.

The steps for one round of the FL process using the proposed communication paradigm are shown in Figure 3 and are as follows:

1. The vehicular clients train their models by using local data;
2. Each client splits its updated model parameters into batches and runs the Kafka Producer API to send these batches to the broker at the edge to be stored in the specified topic named `clients_data`;
3. The MAS runs the Kafka Consumer API and retrieves all models from the topics `clients_data` and performs model averaging;
4. After averaging, the MAS splits the averaged model into batches and runs the Kafka Producer API to send these batches to the broker at the edge to be stored in the specified topic named `averaged_result`;
5. Each client runs the Kafka Consumer API and retrieves the updated model parameters from the topic `averaged_result`;
6. The steps from 1 to 5 are executed until convergence to the desired global model is achieved.

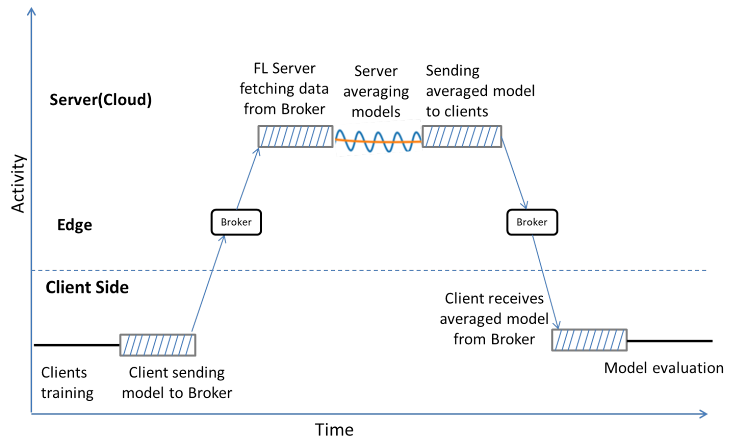

Server(Cloud)
FL Server fetching data from Broker
Server averaging models
Sending averaged model to clients
Activity
Edge
Broker
Broker
Client Side
Client receives averaged model from Broker
Clients training
Client sending model to Broker
Model evaluation
Time

Figure 3: Steps for a round of FL in ICN-based communication architecture [37]

4. Federated Learning Communication Architecture

In this section, we evaluate the effectiveness of the proposed communication architecture for the scenario with massively distributed vehicles in FL. For the performance analysis, we compare the method described in the previous section, which is based on an information-centric paradigm, with another method based on the network-centric paradigm, namely Flower [13]. Flower is a new FL framework for designing and developing FL-based solutions that provide higher level abstractions for designing network architectures. It is one of the few frameworks that supports execution on multiple ML frameworks, such as TensorFlow and PyTorch. The MAS in Flower interacts with clients via Remote Procedure Call (RPC) [38]. RPC is an interaction-based method that uses a request-response protocol for communication. When MAS selects the client for training in Flower, it sends the training and initial model parameters to all connected clients. The client receives these parameters, calls one of the client methods to use them for training with its local data, and sends back the updated local model parameters. All clients must be connected until the FL process is complete. We compare the network-centric and ICN-based architectures as fairly as possible by using the same client selection strategy and neural model parameters in both architectures during the FL process.

For our experimental setup, we use the Kafka framework version 3.0.0 running on a Jetson Xavier NX device with a 64 GB memory card. Our experimental setup includes a broker on the edge and the MAS in the cloud. Both the broker and the cloud run on a Jetson Xavier NX. The clients run on two PCs with the Debian operating system, the Kafka Python API, Python 3.8, and PyTorch installed. We use one PC as a dense server to emulate many FL clients and the other one to emulate only one FL client that examines the network parameters used to calculate the times for a FL round. Using two PCs, we avoid bias in the experimental results caused by the excessive computational overhead of emulating the dense group of clients and examining the parameters.

In the FL process, each client trains its model on its local dataset. The datasets for each client are heterogeneous to better represent the real use case where each client acquires its data randomly and independently. However, to provide an accurate and fair comparison of the performance of the FL method based on the ICN-based architecture and the network-based architecture, we need to guarantee that the client training time is the same on average in both cases. For this reason, we assume that each client has the same average number of data samples. However, this assumption does not hold for the client selection algorithm to account for the heterogeneity of the clients and their different training times. As a dataset, we use the MNIST[2] dataset, which contains a collection of 60k training samples and 10k samples for test data. Each data sample represents a handwritten digit image of 28 x 28 pixels.

To compare the performance between the proposed information-centric architecture and the network-centric architecture, we study the time required to complete a full round of FL for a 5G network scenario, assuming a network bandwidth of 100 MB /s and a delay of 1 ms for the communication links, and varying the number of clients. The 5G network conditions are emulated using the NetEm tool [39]. NetEm provides functions for testing protocols by emulating wide-area network characteristics such as packet delay, jitter, loss, and various bandwidth settings. In real-world environments, client network speeds vary widely by region. However, for our experiments, we assume a homogeneous upload speed based on the 5G use case in the automotive industry. The federation is achieved through an NN model consisting of three layers: an input, a hidden, and an output layer. For all experiments, the parameters of the model are initialized randomly. Each experiment is run five times, and performance is evaluated as an average over these runs. Each client runs five local epochs during training using the Adam optimizer [40] with a batch size of 128 samples and a 0.001 learning rate. The number of clients involved in the federation process varies in the set $\{2, 4, 8, 16, 24, 32, 48, 56, 64\}$.

The results on the time required per FL round with the proposed architecture and the one based on Flower are shown in Figure 4. The results show that Flower is performing better when only a small number of clients are

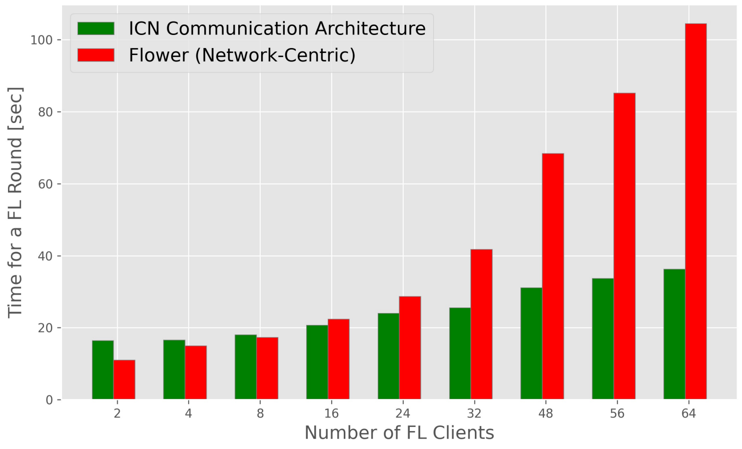


Figure 4: Time per round of FL with different number of clients for 5G use-case for Flower and ICN-based architecture

involved in the FL procedure, while with the increasing number of clients, the time for a FL round also increases. This performance degradation is due to the flash crowd situation that occurs when clients send their models to the MAS simultaneously, which increases the requests on the server. The results of the proposed ICN-based architecture show comparable performance to the Flower-based architecture for a small number of clients; instead, a remarkable performance improvement is achieved in the case of a large number of clients. With only 64 clients, we achieved almost 40% less time to complete a round FL compared to the network-centric architecture. In our architecture, where clients

[2]http://yann.lecun.com/exdb/mnist/

are decoupled from the server through the use of a middleware entity, i.e., a broker at the edge, we minimize the flash crowd effect by allowing clients to send their models regardless of MAS availability.

We also compare the characteristics of the proposed framework with the baseline framework, Flower. It is clear that we achieve the same properties as Flower, such as scalability and ML framework agnosticism since users can perform their FL tasks with any ML framework. It also provides client heterogeneity, as decoupling clients from MAS via a broker supports heterogeneity, as MAS does not know anything about the type of clients connected or their origin. Additionally, this decoupling adds another layer of security in FL. This is because performing all computations on the device, e.g., in FL, is not sufficient to ensure privacy since the ML models transmitted to the server can be exploited to retrieve sensitive information for privacy intrusions, and systems based on FL are obviously not compliant with the GDPR [41]. Therefore, it is necessary to protect clients' information. In our proposed architecture, a broker in the middle is also helpful to ensure the anonymization of clients' information, such as the name and location of each client [42]. Thus, our proposed algorithm not only improves client privacy but also scales up for massive client scenarios in FL and is suitable for both static and non-static users.

5. Federated Learning Client Selection

In this section, we analyze the proposed client selection algorithm and its performance regarding the average number of selected clients using different timer distributions for the vehicular communication scenario considered in this work. Then, we present the performance evaluation of the FL process achieved by the proposed ICN-based architecture using the proposed client selection algorithm.

Client selection is critical in the FL process. While a large number of clients allows for a more accurate construction of the global model, many clients lead to overflows when edge resources are limited. In addition, sending many model updates to the broker leads to excessive latency, network and broker congestion. These considerations become even more realistic with mobile client connections in a vehicular scenario. For these reasons, in this paper, we present a client selection algorithm that aims to avoid broker congestion at the edge and improve the performance of the proposed ICN-based communication architecture. The proposed algorithm moderates the number of randomly selected clients publishing their models on a broker for the FL process, thus reducing broker congestion, latency, queue length, and computational overhead. It should be noted that the implementation of the client selection algorithm requires a change to the communication architecture presented in the previous section. More specifically, we introduce a new topic on the Kafka broker at the edge called `control`. The previously defined CA uses this control topic to perform client selection in FL.

The proposed algorithm selects the clients (vehicles) involved in the FL procedure, while controlling the broker queue size at the edge. When selecting clients, the algorithm considers the heterogeneity of resources and the training time required for each client. In this way, our algorithm selects the clients that are able to complete their training within a certain threshold and with low predefined latency. Moreover, our proposed approach also selects the stragglers with equal probability, since the stragglers are not always slow devices, but may have a large subset of the data for training that provides a larger value for the convergence of the FL model. Therefore, we select them with equal probability. Also, in the client selection, we can remove the assumption made in the previous section about the client's homogeneous local dataset.

We now introduce the basic concepts underlying the proposed timer-based client selection algorithm. We assume that the set of all C clients is available for each round of the FL procedure. Before the start of each round of FL, MAS sends the global model parameters and a configuration message to all clients. The configuration message contains the following information: the current round R of FL, the time interval $\mathcal{T}$ from which we extract the timer values, and the parameters that characterize the distributions of the timers, such as those of the exponential or beta distribution. After receiving the configuration message from MAS, each client runs a backoff timer for a time t_i in seconds, drawn from a distribution with support in $[0, \mathcal{T}]$. When the timer expires, each client trains its NN model on its locally available data and sends the updated model parameters to the broker. The client i, along with its model update, also sends to the broker the information about the current FL -round R to which the update refers and its timer value t_i in seconds. Using these factors, the server can determine whether it needs to change the $\mathcal{T}$ value for the next round based on the number of updates received. It also uses the R parameter to check if a received model update is part of the current FL round.

After receiving model updates from the first client that has the (*minimum_timer*) value, CA is activated at the edge. Then CA sends an ACK back to MAS and the clients that have subscribed to the `control` topic or are involved in the FL process. Upon receiving an ACK, MAS reads the client model parameters from the topic `clients_data` and performs aggregation using the FedAvg algorithm to create the global model [7]. The MAS sends the global model into the topic `averaged_result`. However, the clients receiving the ACK suppress their training and do not send their model parameters if it has not yet completed or if their timers have not yet expired to save computational resources. While the MAS sends the $\mathcal{T}$ and (μ, α) for the next FL round after performing aggregation based on the expected number of clients and the maximum possible round latency. A full round of FL consists of the above procedure. This process is repeated until the accuracy of the overall model is achieved.

We now formalize the evaluation of the average number of selected clients $E(X)$ assuming a given probability distribution for the variable $\mathcal{T}_i$ using the results in [43] for groups of users. In this analysis, we define D_i as the one-way delay between each client and the MAS and $D_{i,e}$ as the one-way delay between each client and its respective edge. For simplicity, we consider the homogeneous delays between the clients and the edge, and the edge to the MAS (cloud).

Let $f_{\mathcal{T}_i}(t_i)$ and $f_{D_i}(d_i)$, with $i = 1, \dots, C$, be the probability density functions (PDFs) of the statistically independent random variables of the time interval $\mathcal{T}_i$ and the client-broker communication delays D_i, respectively. The random variables $G_i = D_i + \mathcal{T}_i$, for $i = 1, \dots, C$, denote the time between the generation of the configuration message by the MAS and the expiration of the back-off time on the i-th client. Assuming independence of $\mathcal{T}_i$ and D_i, the PDF $f_{G_i}(g_i)$ of the random variable G_i is calculated as the convolution of $f_{\mathcal{T}_i}(t_i)$ and $f_{D_i}(d_i)$ [44]:

$$f_{G_i}(g_i) = \int_{-\infty}^{\infty} f_{\mathcal{T}_i}(t_i) f_{D_i}(g_i - t_i) dt_i. \tag{1}$$

In the same way, the PDF of the random variable $W_{i,e} = G_i + D_{i,e}$ can be calculated, which represents the time between sending the configuration message and receiving the ACK message from the MAS. So, the PDF of $W_{i,e}$ is given as:

$$f_{W_{i,e}}(w_{i,e}) = \int_{-\infty}^{\infty} f_{D_{i,e}}(d_{i,e})\, f_{G_i}(w_{i,e} - g_i)\, dg_i. \tag{2}$$

Furthermore, by assuming a constant delay d between the client and the edge broker and between the edge broker and the MAS such that $D_i = 2d$ and $D_{i,e} = d$, the PDFs $f_{D_i}(d_i)$ and $f_{D_{i,e}}(d_{i,e})$ are as follows:

$$f_{D_{i,e}}(d_{i,e}) = \delta\,(d_{i,e} - d),$$
$$f_{D_i}(d_i) = \delta\,(d_i - 2d).$$

Now we calculate the expected number of clients $E(X)$ using the results in [45]. To do this, we define the Bernoulli random variable X_i, which describes whether the client i sends a message with the model updates $X_i = 1$ or not $X_i = 0$. Thus, the expected number of received model updates can be calculated as follows:

$$E(X) = \sum_{i=1}^{C} E(X_i) = C P(X_i = 1), \tag{3}$$

where $P(X_i = 1)$ can be calculated considering that a client sends its model updates when it has not yet received ACK from CA. Thus, the probability that it does not receive ACK is given as follows:

$$P(X_i = 1) = \int_{0}^{\infty} f_{G_i}(g_i)\,(1 - F_{W_{i,e}}(g_i, d_{i,e}))\, dg_i, \tag{4}$$

where $F_{W_{i,e}}(g_i, d_{i,e})$ is the cumulative distribution function of $f_{W_{i,e}}(g_i, d_{i,e})$.

The proposed analysis is valid for any timer distribution. However, we run a simulation to find $E(X)$ for the proposed architecture using three different distributions for the back-off timer: the uniform, beta, and exponential distributions for clients $C = 1000$.

5.1. Uniform Distribution

To select a subset of clients from a large number of clients for FL, the first timer-based technique we used is based on a uniform distribution in which all timer values are uniformly distributed in an interval of $[0, \mathcal{T}]$. Thus, the density of t_i is given as:

$$f_{\mathcal{T}_i}(t_i) = \begin{cases} \frac{1}{\mathcal{T}} & \text{if } 0 \leq t_i \leq \mathcal{T} \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

The timer values extracted by each client from this uniform distribution are given by using the inverse transform sampling [44], which provides the following equation for the timer:

$$t_i = \mathcal{T}\, u, \tag{6}$$

where u is uniformly distributed over $[0, 1]$, and $\mathcal{T} = \{4d, 6d, 8d, 10d\}$. Thus, the expected number of clients $E(X)$ in the case of a uniform distribution of timers is given by Equation (3-4):

$$E(X) = \begin{cases} C & \text{if } \mathcal{T} \leq 2d \\ \frac{2d}{\mathcal{T}} C & \text{if } \mathcal{T} > 2d. \end{cases} \tag{7}$$

The result in Equation (7) shows that the average number of clients generally increases linearly with the number of clients with a slope of 1. However, if $\mathcal{T}$ is large so that $\mathcal{T} > 2d$, then $E(X)$ increases with a slope smaller than 1. Indeed, $\mathcal{T} \leq 2d$ implies that with high probability, each client sends its model before receiving ACK. As $\mathcal{T}$ increases, this probability decreases, as does the number of clients that can send their model updates. This is due to clients receiving the ACK before completing their timer or training. The larger the $\mathcal{T}$ is, the larger the timer values are as well, so only the clients with minimal timer and training time are able to send their model updates.

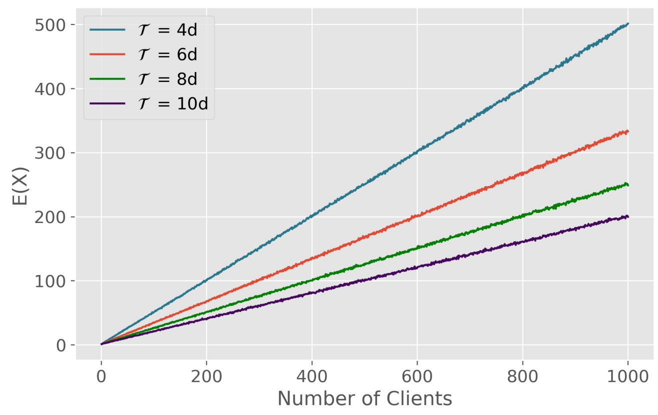
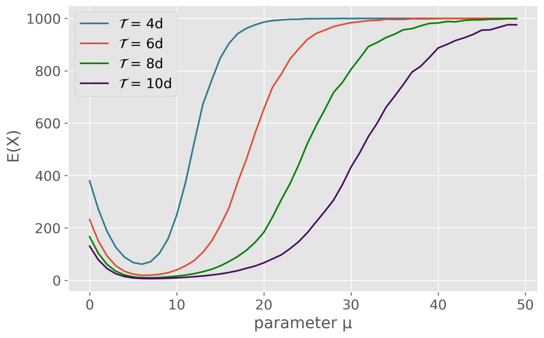
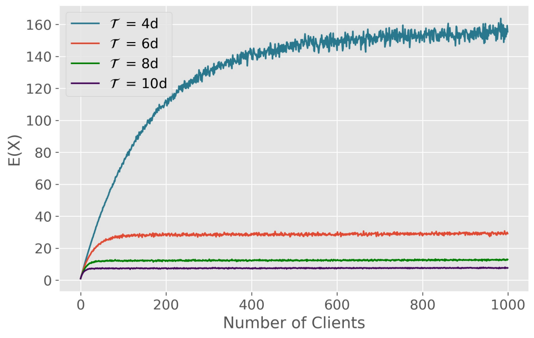


Figure 5: Expected number of clients $E(X)$ with different timer choices $\mathcal{T}$ with uniform distributed timers

Figure 5 shows the results of the average number of selected clients for a uniformly distributed timer. Each point on the curves is the average over 1000 simulations. The achieved simulation results are the same as those in Equation (7). The analysis of the simulations shows that the clients that set their timer within a time interval of $minimum\_timer + 2d$ can send the model updates to the edge, where $2d$ is the delay of receiving the ACK from CA after receiving the model updates from the first client with "$minimum\_timer$". The clients whose back-off timer is longer than this length will suppress the training. Moreover, $\mathcal{T}$ is the only parameter that can affect $E(X)$. This means that we can update the $\mathcal{T}$ parameter to control the queue load (broker) at the edge in FL. Using large values for $\mathcal{T}$ reduces broker congestion at the edge. Conversely, decreasing $\mathcal{T}$ increases traffic due to model updates and consequently increases queue load. Furthermore, $\mathcal{T}$ does not affect the FL rounds required to converge to the optimal global model, as shown in Section 6. Nevertheless, each round is delayed by a known factor $\mathcal{T}$. Thus, there is a tradeoff between using more clients to converge faster and using fewer clients to avoid congestion. Note that we use the terms "model parameters received at the edge" and "number of clients received at the edge" interchangeably, as they both refer to the same thing.



(a) Exponential-distributed timers vs. μ



(b) $E(X)$ with $\mu = 10$

Figure 6: Expected number of clients $E(X)$ with different timer choices $\mathcal{T}$ with exponentially distributed timers

5.2. Exponential Distribution

In this section, experiments are conducted to calculate the average number of selected clients using the exponential distribution with rate μ. More precisely, the truncated exponential distribution is used [46], since the support of the timer random variable is the finite interval $[0, \mathcal{T}]$, so the probability density is given as follows:

$$f_{\mathcal{T}_i}(t_i) = \begin{cases} \frac{\mu}{\mathcal{T}} \frac{1}{e^{\mu}-1} e^{\frac{\mu}{\mathcal{T}} t_i} & \text{if } 0 \leq t_i \leq \mathcal{T} \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

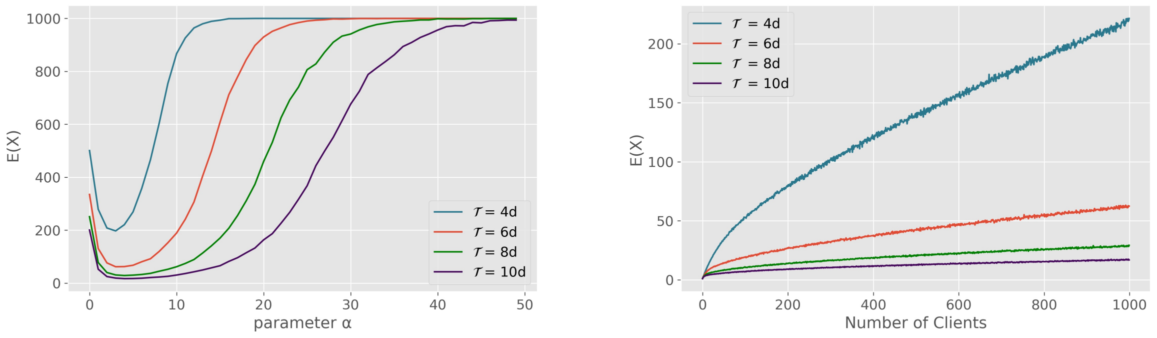
Using inverse transform sampling, the samples for the timer t_i can be generated by the equation:

$$t_i = \frac{\mathcal{T}}{\mu} log[(e^{\mu} - 1)u + 1] \tag{9}$$

where, u is uniformly distributed in $[0, 1]$ as described before. So the expected number of clients $E(X)$ for the truncated exponential distribution is given as:

$$E(X) = \begin{cases} C & \text{if } \mathcal{T} \leq 2d \\ \frac{e^{\frac{2d}{\mathcal{T}}\mu}-1}{e^{\mu}-1} C - e^{\frac{2d}{\mathcal{T}}\mu} \left(\left(\frac{1-e^{-\frac{2d}{\mathcal{T}}\mu}}{1-e^{-\mu}} \right)^{C} - 1 \right) & \text{if } \mathcal{T} > 2d. \end{cases} \tag{10}$$

The results in Equation (10) show that if $\mathcal{T} \leq 2d$, all clients send their model updates. However, if we fix the time interval $\mathcal{T} > 2d$, the average number of clients is determined by the Equation (10). This Equation is composed of two terms: The first grows linearly, while the second is negatively exponential, which reduces the first so that the average number of clients is asymptotic as the number of clients increases. In this case, MAS must adjust the values of μ and $\mathcal{T}$ to reduce the expected number of clients so that the queue does not overflow at the edge. For the truncated exponential density, we analyse $E(X)$ for $C = 1000$ and by varying both $\mathcal{T}$ and μ considering the uniform training times; the results are shown in Figure 6(a). The Figure shows that for $\mu \in [5, 10]$ we have an optimal number of clients, which means that the load offered to the queue at the edge is small. However, for $\mu > 10$, the weight of the timer density shifts towards $\mathcal{T}$, which allows for a large number of model updates. For $\mu < 5$, on the other hand, the timer samples are again in a small interval, i.e., closer to 0, which allows a large number of model parameters, but fewer than in the last case. The experiments are repeated by fixing the value $\mu = 10$ and varying $\mathcal{T}$ for a different number of clients; the results are shown in Figure 6(b). The expected number of clients $E(X)$ is inversely proportional to $\mathcal{T}$, as in the uniform distribution case. However, in the exponential distribution case, $E(X)$ remains almost constant even if we increase the value of $\mathcal{T}$ for a large number of clients. This is because, for larger values of $\mathcal{T}$, clients receive ACK before their timer expires, causing many clients to suppress their model updates. The exponential distribution parameter μ affects the



(a) $E(X)$ vs. α for different values of $\mathcal{T}$ and number of clients $C = 1000$ (b) $E(X)$ vs. number of clients C for $\alpha \sim 5$, and $\mathcal{T} = \{4d, 6d, 8d, 10d\}$

Figure 7: $E(X)$ of selected clients for model updates given beta-distributed timers

probability of timers with values close to $\mathcal{T}$, which means that the number of clients suppressing their model increases when the considered values of $\mathcal{T}$ are all larger than $2d$. So we can change the edge queue utilisation by adjusting the μ parameter even if the number of clients increases.

5.3. Beta Distribution

The next distribution for which we studied the client selection is the beta distribution. The beta distribution has two parameters, α and β [47]. Given a parameter $\beta = 1$, $\alpha \geq 1$ and a support of the interval $[0, \mathcal{T}]$, the probability density function is given as:

$$f_{\mathcal{T}_i}(t_i) = \begin{cases} \frac{\alpha}{\mathcal{T}} \left(\frac{t_i}{\mathcal{T}}\right)^{\alpha-1} & \text{if } 0 \leq t_i \leq \mathcal{T} \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

With the help of the inverse transform sampling, the samples for the timer t_i are calculated by the equation:

$$t_i = \mathcal{T} u^{\frac{1}{\alpha}}, \tag{12}$$

where, u is uniformly distributed in $[0, 1]$. The expected number of clients $E(X)$ for the beta distribution is given as:

$$E(X) = \begin{cases} C & \text{if } \mathcal{T} \leq 2d \\ \left(\frac{2d}{\mathcal{T}}\right)^{\alpha} C + \alpha\, C \int_0^1 \frac{2d}{\mathcal{T}} t_i^{\alpha-1} \left(1 - \left(t_i - \frac{2d}{\mathcal{T}}\right)^{\alpha}\right)^{C-1} dt_i & \text{if } \mathcal{T} > 2d\ . \end{cases} \tag{13}$$

Analyzing the result in equation (13) again, we find, as in the previous cases, that for time intervals less than or equal to $2d$ all clients can send updates. Instead, for time intervals $\mathcal{T}$ greater than $2d$, $E(X)$ is provided by a function where MAS can adjust the two parameters α and $\mathcal{T}$ to change the number of expected clients. Figure 7(a) shows the simulation results of the expected number of selected clients when α varies. Figure 7(a) shows an optimal number of $E(X)$ when $\alpha = 5$; this means that for the given value of $\alpha = 5$, a large number of clients suppressed their model updates. Changing the value of $\mathcal{T}$ has similar effects on the number of clients selected as in the previous distributions. From the analysis of the equation (13) along with the simulation results, it can be seen that for values of $\alpha > 5$, the timer samples in the interval are closer to $\mathcal{T}$, which can lead to a large number of models. For $\alpha < 5$, the timer sample values are closer to 0 in a small interval, which again increases the number of clients. In Figure 7(b), we set the optimal value of $\alpha = 5$ to examine the effects of different $\mathcal{T}$ intervals. It is clear that a smaller $\mathcal{T}$ leads to a large number of clients while increasing $\mathcal{T}$ beyond a certain value does not have much impact on the number of clients.

The results obtained in Section 5 for suppressing client updates for the considered distribution scenario indicate that by using a parametric distribution for timer selection and keeping the interval size $\mathcal{T}$, it is possible to prevent edge queue overflow for a large number of clients by using probabilistic timers that add minimal latency to the FL round. In addition, the results obtained with exponential and beta distributions provide a more accurate estimate of the number of clients than the uniform distribution. As the expected number of clients is convex when the parameters μ and α vary for the exponential and beta distributions. Moreover, the exponential and beta distributions provide another parameter that MAS can tune by considering the tradeoff between the load offered to the queue at the edge and the accuracy of the global model. From the results, we also conclude that the exponential distribution is an appropriate choice for sampling the timers. Indeed, the average number of clients for this distribution shows the best asymptotic behaviour as the number of clients increases.

6. Federated Learning Performance

In this section, we evaluate the performance of the FL process for the proposed communication architecture and client selection algorithm. As described earlier, each client performs local training and sends model parameters to the broker. These model updates are retrieved by the MAS via broker and combined into a global NN model in each communication round, as shown in Figure 3. However, the cost and communication overhead between participating clients and the broker at the edge can be a major bottleneck when multiple clients are involved. In addition, storing model updates in edge queues, as described previously, can lead to congestion in the edge infrastructure. Therefore, there is a need to reduce the overflow of edge queues due to enormous clients. In this work, we propose to select a small fraction of clients and send the model parameters of these selected clients to the server using a time-based backoff method.

Experiments are performed with a set of clients $C = \{5, 10, 15, 20, 25, 30, 35, 40, 50\}$, a broker at the edge, and a MAS in the cloud for simulation. We used the MNIST dataset to run our simulations. The learning model is based on a three-layer NN with 200 neurons in the hidden layer, an input and output layer, a batch size of 32 samples, a learning rate of 0.01, and 10 local epochs performed by each client before model updates are sent to the broker. In addition, the weight assigned to each client for aggregation by MAS corresponds to the size of the dataset used by each client for training. Also, we terminate the FL process after achieving a classification accuracy of 97% of the global model. For client selection in FL, we chose timers with exponential distributions that perform better than uniform and beta distributions in terms of model suppression, as evaluated in Section 5. The exponentially distributed timer has the following parameters: $\mathcal{T} = \{4, 6, 8, 10\}$ and $\mu = 10$. We repeat each experiment 10 times to calculate the average FL round count for the selected clients with a confidence interval of 95%.

Using the above parameters, we examine the impact of the proposed client selection algorithm on the global model accuracy and the communication overhead and broker load. We find that limiting the number of clients that are sending their model parameters in the FL process reduces communication costs and frees up the computational resources of the remaining clients, allowing faster convergence in a smaller number of FL rounds.

In Figure 8 we show the variation in the number of FL rounds for different numbers of clients. Figure 8 shows the FL rounds required to achieve 97% accuracy for different values of the $\mathcal{T}$ parameter and with selected FL clients. It can be seen that the number of rounds drops rapidly from a maximum of 20 with 5 clients to 13 rounds with 50 clients to obtain the federated model that provides the prescribed accuracy. Note that we set the desired accuracy to 97% to study the impact of selecting a different number of clients with the proposed timer-based client selection mechanism. Also, we considered sending the updated global model to all C clients, not just those selected during the client selection process. The reduced number of rounds required for convergence compensates the increased communication overhead of sending the global model to all clients. The empirical results show that our strategy increases learning effectiveness and ensures a more consistent and fair performance for all clients. In fact, with the proposed algorithm, we achieve convergence with only a few clients and a smaller number of FL rounds.

Clients Heterogeneity in Federated Learning: Now we analyze the impact of client heterogeneity on the client selection algorithm. Client heterogeneity, which includes heterogeneity of systems and data, is one of the main problems of FL. Differences in client performance, such as CPU frequency, data size, and transmission performance, are referred to as system heterogeneity, while data heterogeneity refers to differences in the distribution of data across clients, i.e., non-iid (non-independent and identical) data distribution [48]. Each of these conditions directly leads to differences in training latency across clients. Besides the heterogeneity of data and devices, the other main problem

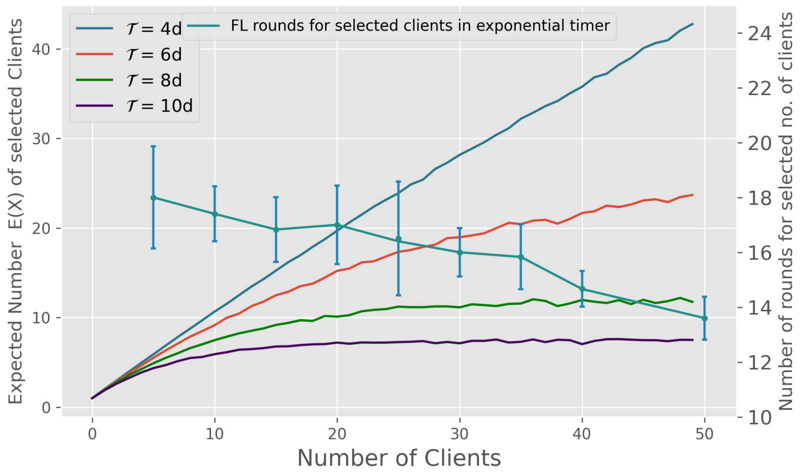
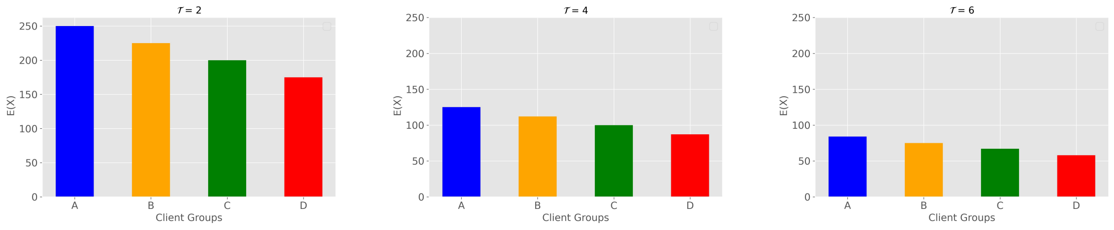


Figure 8: Expected number $E(X)$ of clients selected using a timer with exponential distribution when $\mu = 10$ and with different time intervals $\mathcal{T}$, and the number of FL rounds for a given number of clients

of FL is the dynamic environment in which the communication network can fail and devices can lose connectivity, such as mobile nodes (vehicles). This is due to the fact that many of the clients involved in the system are mobile, and this problem is exacerbated in a vehicle scenario. To overcome these problems, the communication architecture is developed in the above section using the pub/sub paradigm. In the following, we show the numerical results that demonstrate the resilience of the method based on the proposed FL architecture to client heterogeneity. We account for the heterogeneity of clients by dividing them into four classes: A, B, C, and D, ordered from the fastest class A to the slowest class D. They can be slow for two reasons: either they have a large subset of data to train the NN model, or they consist of simpler and slower hardware. The simulations are performed considering the available resources of the clients. In this way, the FL procedure can avoid selecting clients that are not able to complete the training and that could interfere with the current FL rounds or lead to delayed responses. By specifying the values of $\mathcal{T}$ in the proposed algorithm, we introduce a known delay in each FL round, i.e., equal to $\mathcal{T}$. However, in previous scenarios, the delay in FL was based on the slowest device in the system, which is not known in advance. Experiments are performed separately for each of the time distributions considered in this work: uniform, exponential, and beta. The results show that suppressing client updates while maintaining the required classification accuracy, i.e. 97%, is best achieved with the exponential distribution.

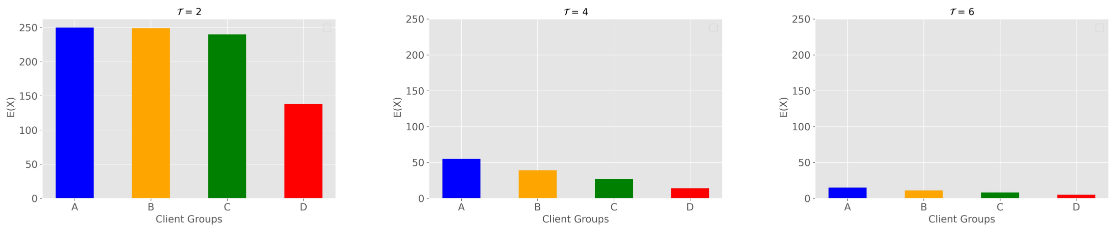
Figures 9(a)-9(c) show the simulation results for the client selection algorithm in the case of uniformly distributed timers. As mentioned earlier, when the client training times are homogeneous, we obtain the updates from all clients at time $\mathcal{T} = 2$, as shown in Equation (7). However, we now consider the heterogeneous client resources, which include fast and slow devices. For this reason, it is not possible to get the updates from all clients, since the training suppression by the slowest clients is combined with the training suppression by the timers. Moreover, the number of clients continues to decrease as $\mathcal{T}$ increases, as can be seen in Figures 9(b), 9(c). This is because the clients of all classes extract larger timer values and therefore get ACK. The ACK is generated by CA when the model parameters are received from the client with the fastest class and a smaller timer value. This behaviour that the number of clients decreases while $\mathcal{T}$ increases, holds for all timers mentioned, both for homogeneous and heterogeneous clients.

From Figures 9(a)-9(c), it can also be seen that the proposed client selection algorithm selects clients from all four groups, which shows that the proposed selection algorithm is not affected by the heterogeneity of clients, but only by the parameter $\mathcal{T}$ computed by MAS. Moreover, each timer value for each client is independent and random. So there is a possibility that clients in classes A and B will receive a larger timer value than clients in categories C and D, and that they will receive the ACK before their timer or training is complete. This behaviour of independent selection from each class makes the algorithm more dynamic. Therefore, the algorithm also selects clients that train their model on a



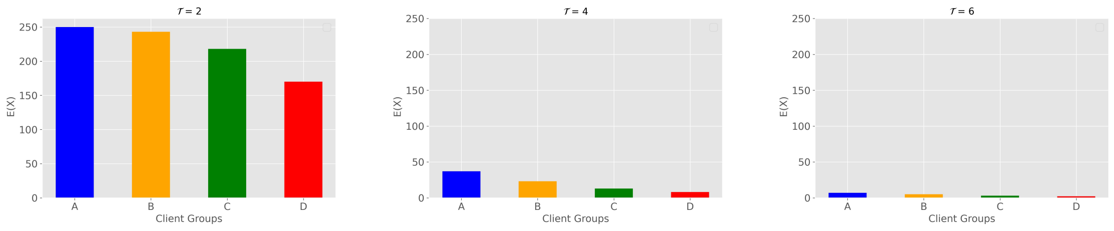
(a) Client selection when $\mathcal{T} = 2$ (b) Client selection when $\mathcal{T} = 4$ (c) Client selection when $\mathcal{T} = 6$

Figure 9: Selection of clients using heterogeneous resources from four classes with uniform-distributed timers



(a) Client selection for time interval $\mathcal{T} = 2$ (b) Client selection for time interval $\mathcal{T} = 4$ (c) Client selection for time interval $\mathcal{T} = 6$

Figure 10: Clients selection using heterogeneous resources from four classes with beta-distributed timers by setting $\alpha = 5$



(a) Client selection for time interval $\mathcal{T} = 2$ (b) Client selection for time interval $\mathcal{T} = 4$ (c) Client selection for time interval $\mathcal{T} = 6$

Figure 11: Clients selection using heterogeneous resources from four classes with exponential timers by setting $\mu = 10$

larger dataset, i.e., the clients belonging to the slower classes that place more emphasis on global model convergence and to the clients that can provide their updates quickly, which contributes to faster convergence.

Figures 10(a)-10(c) show the simulation results for the client selection algorithm in the case of beta-distributed timers with $\alpha = 5$. The results again show that MAS selects clients from all groups along with CA. However, in this case, clients of classes A, B and C are preferred over class D. This is because we chose the optimal value of $\alpha = 5$, which allows for better suppression of clients since all timers are nearly distributed. As a result, client selection increasingly depends on the training times of the clients. In the case of $\mathcal{T} = 6$, client selection is again more evenly distributed between groups as the density of timers move towards $\mathcal{T}$. Nevertheless, the stragglers are not completely neglected.

Figures 11(a)-11(c) show the simulation results for the client selection algorithm in the case of exponentially distributed timers with $\mu = 10$. The results show that MAS distributes the clients more evenly among the groups in this case, and that we do not select only clients from one category or the fastest clients, but from all four classes. Also, fewer clients are selected compared to the uniform and beta-distributed timers, especially for $\mathcal{T} = 4$ and $\mathcal{T} = 6$. The simulation results show that the exponentially distributed timers allow the FL method to load the queue less at the edge than in the case of the uniform and beta distributions. Thus, the properties we obtained for homogeneous training timers with different timers also hold for the heterogeneous clients.

Table 1
Jain fairness index for different probabilistic timers distribution and different values of $\mathcal{T}$

Probability	$\mathcal{T}$ values				
Distribution	2	4	6	8	10
Uniform	0.9831	0.9828	0.9825	0.9841	0.9835
Beta	0.9564	0.8365	0.8764	0.9047	0.9133
Exponential	0.9803	0.7777	0.8791	0.9304	0.9554

We need to select clients from four different classes and to perform a fair selection among all of them in each round of FL, such that the slowest clients can also participate in the FL process. To this end, we measure the fairness of each group in each round of FL using Jain's fairness index [49] as given in Equation 14:

$$J = \frac{(\sum_{i=1}^{n} x_i)^2}{n\sum_{i=1}^{n} {x_i}^2} \tag{14}$$

The Jain index is a fairness criterion that takes into account all users of the system. Because of its simplicity and ease of understanding, the Jain index is still one of the most popular measures for comparing the fairness of allocations in computer networks, but we used it to justify selection from different groups of clients in the FL process. The value of $J = 1$ corresponds to the fairest allocation, where all users have the same benefit. In the case of FL, the value of $J = 1$ corresponds to the same number of clients from each category. Therefore, the FL process must consider the tradeoff between model accuracy and index maximization to avoid overloading the broker during client selection. We calculated the fairness of client selection among all the proposed probabilistic timers to determine which of them is the fairer in the client selection. Table 1 shows the results of the Jain index evaluated for the uniform, exponential and beta-distributed timers. The results in this Table show that the uniform distributed timers provide a higher value for J. The results also show that the fairness of the uniform distributed timers remains almost the same across different time intervals. However, uniform distributed timers suppress fewer client updates, resulting in many model updates being transmitted to the edge queue. The results for the exponentially distributed timers are comparable to the values for the uniformly distributed case for $\mathcal{T} = 2$. In fact, in the exponentially distributed case for $\mathcal{T} = 2$, a large number of clients from all four classes are involved in the FL process due to the backoff timer values. But as mentioned earlier, the exponentially distributed timers provide better load balancing in the queue at the edge than the other distributions. Moreover, for all timers, fairness decreases at $\mathcal{T} = 4$ and then increases again at higher $\mathcal{T}$ values. In summary, our proposed client selection algorithm gives us the best performance in terms of fairness and load offered to the queue at the edge with the exponentially distributed timers.

7. Conclusion and Future Work

In this study, we implemented an edge-based architecture for FL adapted to communication and computation costs. We decompose the global problem of an efficient FL system for mobile and resource-constrained devices into two subproblems. In our solution process, we focus on a communication-efficient framework for FL, which is our first research question, and the second is client selection in FL considering the resource-constrained edge infrastructure. The ICN-based framework for communication-efficient FL can be used for mobile vehicle nodes because it enables asynchronous and scalable communication. According to the simulation results, the proposed framework can provide better FL results than a network-centric architecture and significantly reduce the time required to perform a FL round. Another advantage of this architecture is that it reduces the load on MAS since in the ICN paradigm, a MAS serves only a small number of brokers, which in turn distribute model updates to all vehicles in the environment. This architecture also helps reduce latency while solving the problem of intermittent connectivity for mobile users. In addition, decoupling clients from MAS via brokers helps address privacy challenges in FL. Therefore, we show that the pub/sub paradigm is the most promising communication model for FL.

After defining the communication architecture, we investigated a client selection mechanism for the FL process by considering the load on the edge queues where the edge broker caches model updates uploaded by connected clients. The proposed client selection mechanism uses different probabilistic timers and determines the effective distribution in case of device heterogeneity and different number of clients. This time-based client selection mechanism requires very little state on each client device, requires only network data transmission support, and adapts to massive client scenarios. We have studied probabilistic feedback timers for up to 1000 clients through experiments and analysis. Our main results are: exponentially distributed timers provide better client update suppression than systems based on uniform and beta-distributed timers, and reduce queue overflows. The algorithm works for both heterogeneous and homogeneous clients and can therefore be used in almost any system, such as satellite-based networks. It also enables to control the bandwidth for uploading model parameters by tweaking various parameters depending on the tradeoff between the latency for each FL round and the expected number of clients. The proposed system, which dynamically adjusts the selection frequency, reduces client participation to avoid queue overflow and improves the robustness of the system. We also investigated the fairness of the proposed client selection algorithm based on different client categories by considering their heterogeneity. In this way, we prove an optimal tradeoff between efficiency and fairness from the Jain index point of view and show that our proposed algorithm ensures fairness in selecting clients from different classes.

In the future, we will extend our test environment to include a cluster of brokers and orchestrate them at the edge, using the Kafka APIs. We will refine the methods and details of the client selection technique by also considering the heterogeneous delays between edge and client devices. To exploit data from unmanned aerial vehicles or remote areas, we would also investigate how to move the nodes at the edge to a low-orbit satellite constellation to create an orbital edge infrastructure.

Acknowledgements

This work has been supported by the H2020 TEACHING project (GA#871385).

References

[1] Y. Liu, X. Yuan, Z. Xiong, J. Kang, X. Wang, D. Niyato, Federated learning for 6g communications: Challenges, methods, and future directions, China Communications 17 (2020) 105–118.

[2] Y. Xiao, G. Shi, M. Krunz, Towards ubiquitous ai in 6g with federated learning, arXiv preprint arXiv:2004.13563 (2020).

[3] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, Y.-J. A. Zhang, The roadmap to 6g: Ai empowered wireless networks, IEEE Communications Magazine 57 (2019) 84–90.

[4] Y. Shi, K. Yang, T. Jiang, J. Zhang, K. B. Letaief, Communication-efficient edge ai: Algorithms and systems, IEEE Communications Surveys & Tutorials 22 (2020) 2167–2191.

[5] Z. Yang, M. Chen, K.-K. Wong, H. V. Poor, S. Cui, Federated learning for 6g: Applications, challenges, and opportunities, Engineering 8 (2022) 33–41.

[6] K. B. Letaief, Y. Shi, J. Lu, J. Lu, Edge artificial intelligence for 6g: Vision, enabling technologies, and applications, IEEE Journal on Selected Areas in Communications 40 (2021) 5–36.

[7] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. y Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial Intelligence and Statistics, PMLR, 2017, pp. 1273–1282.

[8] K. B. Letaief, W. Chen, Y. Shi, J. Zhang, Y.-J. A. Zhang, The roadmap to 6g: Ai empowered wireless networks, IEEE communications magazine 57 (2019) 84–90.

[9] Tensorflow federated, ???? URL: https://www.tensorflow.org/federated.

[10] D. Chai, L. Wang, K. Chen, Q. Yang, Fedeval: A benchmark system with a comprehensive evaluation model for federated learning, CoRR abs/2011.09655 (2020).

[11] C. He, S. Li, J. So, X. Zeng, M. Zhang, H. Wang, X. Wang, P. Vepakomma, A. Singh, H. Qiu, X. Zhu, J. Wang, L. Shen, P. Zhao, Y. Kang, Y. Liu, R. Raskar, Q. Yang, M. Annavaram, S. Avestimehr, Fedml: A research library and benchmark for federated machine learning, 2020. arXiv:2007.13518.

[12] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, J. Passerat-Palmbach, A generic framework for privacy preserving deep learning, 2018. arXiv:1811.04017.

[13] D. J. Beutel, T. Topal, A. Mathur, X. Qiu, T. Parcollet, P. P. B. de Gusmão, N. D. Lane, Flower: A friendly federated learning research framework, 2021. arXiv:2007.14390.

[14] P. Kumar, B. Dezfouli, Implementation and analysis of quic for mqtt, Computer Networks 150 (2019) 28–45.

[15] J. Kreps, N. Narkhede, J. Rao, et al., Kafka: A distributed messaging system for log processing, in: Proceedings of the NetDB, volume 11, 2011, pp. 1–7.

[16] B. Al-Madani, A. Al-Roubaiey, Z. A. Baig, Real-time qos-aware video streaming: a comparative and experimental study, Advances in Multimedia 2014 (2014).

[17] T. Nishio, R. Yonetani, Client selection for federated learning with heterogeneous resources in mobile edge, in: ICC 2019-2019 IEEE international conference on communications (ICC), IEEE, 2019, pp. 1–7.

[18] Y. J. Cho, S. Gupta, G. Joshi, O. Yağan, Bandit-based communication-efficient client selection strategies for federated learning, in: 2020 54th Asilomar Conference on Signals, Systems, and Computers, IEEE, 2020, pp. 1066–1069.

[19] S. Bano, N. Tonellotto, P. Cassarà, A. Gotta, Fedtcs: Federated learning with time-based client selection to optimize edge resources (2022).

[20] Q. Yang, Y. Liu, T. Chen, Y. Tong, Federated machine learning: Concept and applications, ACM Transactions on Intelligent Systems and Technology (TIST) 10 (2019) 1–19.

[21] N. Tonellotto, A. Gotta, F. M. Nardini, D. Gadler, F. Silvestri, Neural network quantization in federated learning at the edge, Information Sciences 575 (2021) 417–436.

[22] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, D. Bacon, Federated learning: Strategies for improving communication efficiency, arXiv preprint arXiv:1610.05492 (2016).

[23] H. H. Yang, A. Arafa, T. Q. Quek, H. V. Poor, Age-based scheduling policy for federated learning in mobile edge networks, in: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2020, pp. 8743–8747.

[24] F. Sattler, S. Wiedemann, K.-R. Müller, W. Samek, Robust and communication-efficient federated learning from non-iid data, IEEE transactions on neural networks and learning systems 31 (2019) 3400–3413.

[25] A. Feraudo, P. Yadav, V. Safronov, D. A. Popescu, R. Mortier, S. Wang, P. Bellavista, J. Crowcroft, Colearn: Enabling federated learning in mud-compliant iot edge networks, in: Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking, 2020, pp. 25–30.

[26] C. Martín, P. Langendoerfer, P. S. Zarrin, M. Díaz, B. Rubio, Kafka-ml: connecting the data stream with ml/ai frameworks, arXiv preprint arXiv:2006.04105 (2020).

[27] Y. Tang, K. Guo, J. Ma, Y. Shen, T. Chi, A smart caching mechanism for mobile multimedia in information centric networking with edge computing, Future Generation Computer Systems 91 (2019) 590–600.

[28] Y. Du, M. Chowdhury, M. Rahman, K. Dey, A. Apon, A. Luckow, L. B. Ngo, A distributed message delivery infrastructure for connected vehicle technology applications, IEEE Transactions on Intelligent Transportation Systems 19 (2017) 787–801.

[29] S. Oh, J.-H. Kim, G. Fox, Real-time performance analysis for publish/subscribe systems, Future Generation Computer Systems 26 (2010) 318–323.

[30] C. Rodríguez-Domínguez, K. Benghazi, M. Noguera, J. L. Garrido, M. L. Rodríguez, T. Ruiz-López, A communication model to integrate the request-response and the publish-subscribe paradigms into ubiquitous systems, Sensors 12 (2012) 7648–7668.

[31] P. T. Eugster, P. A. Felber, R. Guerraoui, A.-M. Kermarrec, The many faces of publish/subscribe, ACM computing surveys (CSUR) 35 (2003) 114–131.

[32] X. Li, K. Huang, W. Yang, S. Wang, Z. Zhang, On the convergence of fedavg on non-iid data, arXiv preprint arXiv:1907.02189 (2019).

[33] S. AbdulRahman, H. Tout, A. Mourad, C. Talhi, Fedmccs: Multicriteria client selection model for optimal iot federated learning, IEEE Internet of Things Journal 8 (2020) 4723–4735.

[34] S. Bano, Phd forum abstract: Efficient computing and communication paradigms for federated learning data streams, in: 2021 IEEE International Conference on Smart Computing (SMARTCOMP), 2021, pp. 410–411. doi:10.1109/SMARTCOMP52413.2021.00086.

[35] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, A. Ahmed, Edge computing: A survey, Future Generation Computer Systems 97 (2019) 219–235.

[36] S. Bano, E. Carlini, P. Cassara', M. Coppola, P. Dazzi, A. Gotta, A novel approach to distributed model aggregation using apache kafka, in: Proceedings of the 2nd Workshop on Flexible Resource and Application Management on the Edge, FRAME '22, Association for Computing Machinery, New York, NY, USA, 2022, p. 3336. URL: https://doi.org/10.1145/3526059.3533621. doi:10.1145/3526059.3533621.

[37] S. Bano, N. Tonellotto, P. Cassarà, A. Gotta, Kafkafed: Two-tier federated learning communication architecture for internet of vehicles, in: 2022 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), 2022, pp. 515–520. doi:10.1109/PerComWorkshops53856.2022.9767510.

[38] B. J. Nelson, Remote procedure call, Carnegie Mellon University, 1981.

[39] S. Hemminger, et al., Network emulation with netem, in: Linux conf au, volume 5, Citeseer, 2005, p. 2005.

[40] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014. URL: https://arxiv.org/abs/1412.6980. doi:10.48550/ARXIV.1412.6980.

[41] N. Truong, K. Sun, S. Wang, F. Guitton, Y. Guo, Privacy preservation in federated learning: An insightful survey from the gdpr perspective, Computers & Security 110 (2021) 102402.

[42] S. Arshad, M. A. Azam, M. H. Rehmani, J. Loo, Recent advances in information-centric networking-based internet of things (icn-iot), IEEE Internet of Things Journal 6 (2018) 2128–2158.

[43] J. Nonnenmacher, E. W. Biersack, Optimal multicast feedback, in: Proceedings. IEEE INFOCOM'98, the Conference on Computer Communications. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Gateway to the 21st Century (Cat. No. 98, volume 3, IEEE, 1998, pp. 964–971.

[44] A. Popoulis, S. U. Pillai, Probability, random variables, and stochastic processes, Boston, MA, McGraw-Hill (1991).

[45] E. Castillo, Extreme value theory in engineering, New York: Academic, (1988).

[46] M. F. M. Al-Athari, Estimation of the mean of truncated exponential distribution, Journal of Mathematics and Statistics 4 (2008) 284.

[47] P. E. Pfeiffer, Probability for applications, Springer Science & Business Media, 2012.

[48] X. Liu, H. Li, X. Lu, T. Xie, Q. Mei, F. Feng, H. Mei, Understanding diverse usage patterns from large-scale appstore-service profiles, IEEE Transactions on Software Engineering 44 (2017) 384–411.

[49] R. Jain, A. Durresi, G. Babic, Throughput fairness index: An explanation, in: ATM Forum contribution, volume 99, 1999.

Saira Bano received her M.Sc. in Telecommunications Engineering from Politecnico di Torino, Italy, in 2020. She is currently pursuing her PhD in Information Engineering at the University of Pisa, Italy, and conducting research at the Institute of Information Sciences and Technologies (ISTI) of the National Research Council (CNR), Pisa, Italy. Her research interests include information-centric networks, mobile and pervasive computing, cyber-physical systems, federated learning, machine learning, and data science.

Nicola Tonellotto received his M.Sc and Ph.D in 2002 and 2008, respectively. He is associate professor at the Information Engineering Department of the University of Pisa since 2022 and honorary research fellow in the College of Science & Engineering of the School of Computing Science of the University of Glasgow since 2020. From 2002 to 2019 he was researcher at the Information Science and Technologies Institute "A. Faedo" of the National Research Council of Italy. His main research interests include Cloud Computing, Web Search, and Information Retrieval, with a particular focus on efficient data processing in distributed computing architectures. He co-authored more than 80 papers on these topics in peer reviewed international journals and conferences. He is co-recipient of the ACM SIGIR 2015 Best Paper Award.

Pietro Cassará received his M.Sc. degrees in Telecommunication and Electronic Engineering from University of Palermo in the 2005, and its Ph.D. degree in the 2010, jointly with the State University of New York. Nowadays, he is staff member of the Institute of Science and Information Technologies (ISTI), at the National Research Council (CNR), Pisa, Italy, and since 2017 he has been temporary staff member of the CMRE lab at the NATO of La Spezia. He is currently a member of the IEEE ComSoc and VTS Committees, his research interests include wireless sensor network and IoT communications. He has been participating in European, ESA, and National funded projects.

Alberto Gotta received his M.Sc and Ph.D in 2002 and 2007, respectively. He is a researcher of the Institute of Information Science and Technologies (ISTI) of the National Research Council (CNR), Italy. His research interests include traffic engineering, satellite and non-terrestrial networks, aerial networks, IoT, sensor networks, and machine learning for communications. He has been principal investigator of several EU, national, regional and ESA funded R&D projects. He co-authored more than 100 papers among which the most cited article in 2012 and 2013 on Elsevier Computer Communications and the second most cited article from 2019 on Elsevier Array. He serves the TPCs of flagship ComSoc conferences and symposia and also the editorial boards of MDPI Sensors, MDPI Network, Frontiers, the International Journal of Informatics and Communication Technology (IJ-ICT), the International Journal of Power Electronics and Drive Systems (IJPEDS), and the Journal of Computer Networks and Communications (CNC).