P4FL: An Architecture for Federating Learning with In-Network Processing

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 5 September 2023, accepted 15 September 2023, date of publication 22 September 2023,
date of current version 27 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3318109

RESEARCH ARTICLE

P4FL: An Architecture for Federating Learning With In-Network Processing

ALESSIO SACCO[1], **(Member, IEEE)**, **ANTONINO ANGI**[1], **(Student Member, IEEE)**,
GUIDO MARCHETTO[1], **(Senior Member, IEEE)**, **AND FLAVIO ESPOSITO**[2], **(Member, IEEE)**
[1]Department of Control and Computer Engineering, Politecnico di Torino, 10129 Turin, Italy
[2]Department of Computer Science, Saint Louis University, Saint Louis, MO 63103, USA

Corresponding author: Alessio Sacco (alessio_sacco@polito.it)

This work was supported in part by Comcast, and in part by the National Science Foundation (NSF) under Award 1908574 and Award 2201536.

ABSTRACT The unceasing development of Artificial Intelligence (AI) and Machine Learning (ML) techniques is growing with privacy problems related to the training data. A relatively recent approach to partially cope with such concerns is Federated Learning (FL), a technique in which only the parameters of the trained neural network models are transferred rather than data. Despite the benefits that FL may provide, such an approach can lead to synchronization issues (especially when applied in the context of numerous IoT devices), the network and the server may turn into bottlenecks, and the load may become unsustainable for some nodes. To solve this issue and reduce the traffic on the network, in this paper, we propose *P4FL*, a novel FL architecture that uses the paradigm of network programmability to program P4 switches to compute intermediate aggregations. In particular, we defined a custom in-band protocol based on MPLS to carry the model parameters and adapted the P4 switch behavior to aggregate model gradients. We then evaluated *P4FL* in Mininet and verified that using network nodes for in-network model caching and gradient aggregating has two advantages: first, it alleviates the bottleneck effect of the central FL server; second, it further accelerates the entire training progress.

INDEX TERMS Data plane programmability, federated learning, machine learning, network processing, p4.

I. INTRODUCTION

The deployment of Federated Learning (FL) architectures is increasing, involving several applications. For example, smart cities, precision agriculture, healthcare, smart homes, and wearable devices, to name a few. Distributed approaches were introduced with the aim of building Machine Learning (ML) models able to account for security and data privacy concerns. Instead of transferring the raw privacy-sensitive data among devices, only a few parameters during the ML training model would be exchanged, preserving at least some level of privacy [1], [2], [3], [4].

The typical and most common design of an FL system comprises a central server that aggregates the weights of a neural network during a training phase. Each training node of the federation works locally on the available data and submits its partially trained model to such a central server; subsequently, it receives the global model returned by the server and continues the training process [5]. Despite the scalability, privacy, and security benefits that FL provides, researchers have shown that FL may suffer from a few issues (*e.g.*, neural network synchronization among servers and clients) and network or server bottlenecks, as in many distributed systems. The model synchronization can be especially problematic for neural network model convergence time over wide-area networks [4], [6]. These problems are aggravated by the complexity of Deep Neural Networks (DNNs), since billions of parameters need to be transmitted thousands of times between clients and servers, becoming a considerable burden on the communication networks connecting them [7]. Although recent approaches aim to speed up ML training using intermediate servers [8], [9] or network elements [10], [11], these solutions are either tailored to specific network scenarios (*e.g.*, datacenter), or to hardware features supported

The associate editor coordinating the review of this manuscript and approving it for publication was Rentao Gu.

by specific vendors, or require a logically centralized entity (*e.g.*, a Software-Defined Networking (SDN) controller) for part of the computation.

A. OUR CONTRIBUTIONS

To alleviate some of these problems, we present the design and prototype implementation of *P4FL*, our proposed FL architecture that takes advantage of network programmability in general, and P4 [12] programmable switches in particular, to perform intermediate aggregation of model parameters, reducing significantly the model parameter exchange between client and the averaging server. In our design, the model parameters are aggregated locally rather than on the global server, which can thus maintain active connections only for the switches. Such aggregation, in turn, can also mitigate the machine learning model synchronization issues caused by excessive network latency. To speed up the aggregation on switches, we design an in-band FL model exchange protocol based on MPLS that can be interpreted by programmable switches. Since the switches are transparent in the client-server communication, our switch processing introduces a limited overhead in the process, and the final architecture is highly portable.

Designing a federated learning aggregation model with P4 is a challenge, since the architecture does not support the arithmetical division operation and floating-point values, which is fundamental in model aggregation for FL. To overcome the division limitation, we modified the traditional P4-16 compiler and the version of BMV2, *i.e.*, the P4 standard architecture model, in order to support external functions written in C++.

We conducted experiments on an emulated network topology, verifying that the intermediate aggregation limits the network traffic and reduces training time in the case of challenging networks. In our evaluation, we found that P4FL leads to better training process performance in terms of accuracy, loss, and network overhead.

B. PAPER OUTLINE

The rest of the paper is structured as follows. In Section II, we introduce the FL concepts and present some limitations that motivated this work; in Section III, we describe our solution design. Section IV presents the experimental results, and Section V concludes the paper.

II. BACKGROUND ON FEDERATED LEARNING AND RELATED WORK

Over the past few years, we have observed that the number of connected IoT devices has grown exponentially, as they underpin heterogeneous architectures and applications, *e.g.*, smart industry, healthcare, smart homes, and wearables [13], [14]. At its core, Federated Learning (FL) enables end devices to train models cooperatively, whereas Machine Learning (ML) models are trained on such devices without the need to share their local datasets.

A. FEDERATED LEARNING CONCEPT AND ARCHITECTURES

By sending update parameters instead of raw data, *i.e.*, reducing communication data size, FL solutions free up network bandwidth that can be used by delay-sensitive applications, such as augmented and virtual reality, event detection, and autonomous vehicle network systems. Moreover, because data never leave client devices, FL represents an excellent solution in all applications where data privacy and security are essential. In the healthcare industry, for example, where the usage of patient data is regulated by specific laws, a prediction algorithm for clinical purposes, *e.g.*, find clinically similar patients or hospitalization prediction due to cardiac events, can be trained using vast and varied datasets collected in different hospitals, *i.e.*, sites [4].

Similarly, in the insurance sector, FL could be used to detect individuals or businesses accountable for fraud and illicit activity, for example, by training a model that uses varied data ranging from health insurance to cars to mobile to business assets. Or in the IoT field, FL can help to achieve the personalization of user experience and increase the performance of devices, as in text prediction in Google's Android Keyboard and Siri's voice recognition for Apple [6].

The FL process that we consider in this paper can be divided into several iterations. At first, the FL server first determines an ML model to be trained on the client's local database. Then, each participating client downloads the global model from the server and takes local training on its own data. Each client sends the updated model parameters to the server for global aggregation, typically weighted average on parameters. Finally, the aggregated global model is sent to each device for the next round of training. The entire process is repeated until either a target accuracy or a certain number of iterations are achieved. However, there may still be issues mainly caused by the limited resources of edge nodes that have to manage an increasing amount of data and the communication between the server and clients.

B. STATE-OF-THE-ART AND LIMITATIONS

After the concept of Federated Learning (FL) was proposed by Google [5], several modifications have been proposed to solve the synchronization issues that may arise in IoT domain [4], [15], [16], [17]. Despite the fact that data generated on each device stay local, federated networks of IoT can comprise a massive number of agents, and the model communication can be a critical bottleneck due to limited resources, such as bandwidth, energy, and power [1], [18]. In some proposed solutions, the FL approach has been combined with the edge computing paradigm, and the central cloud server is replaced by different edge servers collaborating to obtain the final global model [19], [20], [21]. With this method, both the central server and edge nodes perform aggregation and update of parameters, reducing the traffic on the backbone. Many solutions rely on D2D (device-to-device) communication to speed up the training process,

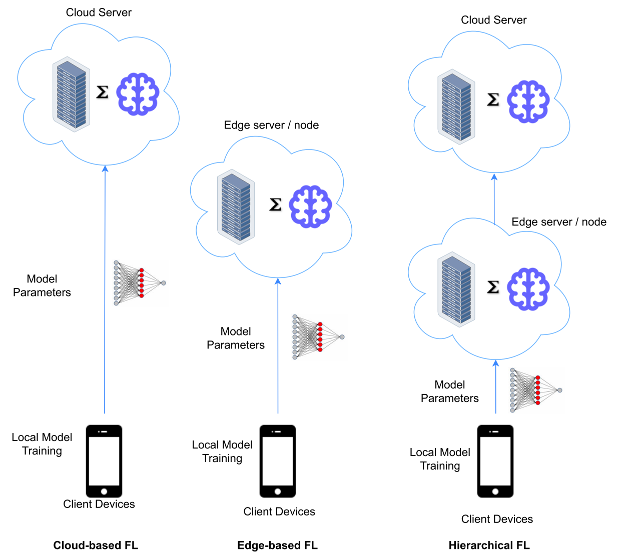


FIGURE 1. Federated Learning architectures, client-based, edge-based and hierarchical FL.

as in [22], which combines the traditional paradigm of device-to-server communication for federated learning with device-to-device communication for model training. Clients are divided into clusters, and, in each round, the central server elects one device from each cluster responsible for aggregating local training parameters and sending the final results to the server.

Intermediate aggregation of model parameters performed by edge servers or, in general, edge nodes, is appearing as a viable trend [8], [9], [23]. These solutions propose intermediate edge servers to perform partial aggregation of the model in a *hierarchical FL system*, which has one cloud server, L edge servers, to which clients with distributed datasets are connected. After each local update, each edge server aggregates its clients' models; then, after every edge model aggregation (less frequent than the local update), the cloud server aggregates all the edge servers' models. This approach lowers the interaction with the cloud compared to the traditional cloud FL. We summarize in Fig. 1 the different approaches, highlighting the differences between *cloud-enabled FL* and *edge-enabled FL* with the new perspective of a *hierarchical FL*.

To eliminate part of the overhead and abundance in the entities involved (typically edge servers run inside VMs and require complicated synchronization protocols), we decided to introduce in-network model processing. As demonstrated in [24], aggregating information along the path rather than on dedicated edge servers can efficiently increase application throughput and reduce latency. Based on the same concept of in-network aggregation for machine learning models, the work of Chen et al. [11] aims to reduce the bottleneck in the network during model synchronization, with special attention to the security of the transmitted data. The model parameters, enciphered using homomorphic encryption, are extracted by the switch and sent to the SDN controller for deciphering and aggregating. The global updated model is sent back to the clients so they can proceed to the next round of training.

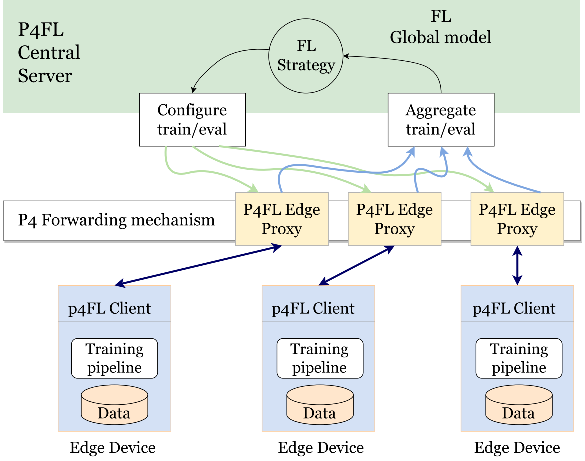


FIGURE 2. *P4FL* system: edge devices train the model and send results to the intermediate node (edge proxy), which aggregates them and responds to clients while forwarding aggregation to the server for final computation.

With the advent of programmable switches, moving these computations inside network devices is becoming a possibility. For example, SwitchML [10] uses computations on the P4 programmable switches to aggregate model updates and to reduce the volume of exchanged data during ML training of model updates from multiple workers in the network. This involves minimal communication so that each worker sends its update model vector and receives the aggregated updates back. However, this approach is designed for a specific network scenario, *i.e.*, datacenter; to overcome some of the P4 language limitations and perform in-network aggregation at line rate, the authors use hardware and vendor-specific features. Similarly, GRID [25] is a recent solution to gradient routing for in-network aggregation, consisting of the control plane and data plane, to mitigate the communication bottleneck and speed up distributed training tasks. In this paper, instead, we aim to provide a general architecture where P4 programs can run on any P4-compatible switch.

III. SYSTEM DESIGN

We summarize in Fig. 2 the main components of P4FL. As it can be seen, our hierarchical architecture is composed of: *(i)* edge devices, also referred to as P4FL clients since they hold training data and start the training process, *(ii)* the intermediate nodes, which in our case are modified instances of P4-enabled switches performing intermediate gradients aggregation, *(iii)* central server, for central aggregation and for driving the entire process by sending instructions to the edge devices. In this scenario, after the local training round on the client ends, the end host creates and sends the packet containing the parameters of the model. This packet is received by the intermediate switches that perform the aggregation and send the intermediate results back to the

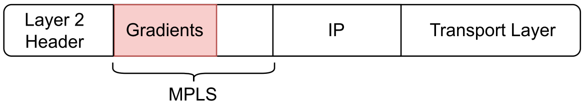


FIGURE 3. MPLS-compliant version of P4FL. Model's gradients are inserted directly into the packet (in red), with little impact on the switch-forwarding process.

clients while forwarding the same result to the server, which can aggregate all results of switches. Because the switch is the equivalent of an edge server in an edge-computing architecture, we also refer to this node in general as *P4FL Edge Proxy* of the FL architecture. In what follows, we detail the role and functionalities of these devices in the FL process.

A. NETWORK NODES ASSISTANCE

Programmable switches are pivotal in our architecture as they aggregate model parameters and constitute the intermediate server of the model exchange communication. The behavior of these switches has been expressed using P4, a high-level programming language for packet processing [12], which provides some clear advantages compared to the traditional way of expressing traffic management rules [26], [27]. P4 defines a device-independent way of expressing how programmable forwarding elements, such as a programmable switch, should manage packets independently of the hardware architecture and platforms, *e.g.*, NetFPGA, and Intel Tofino. This introduces much flexibility in the system compared with a traditional one because the data plane can be changed in a programmatically way; particularly, it is possible to parse the packet header, extract information, and use it to process the packet.

In order to speed up the aggregation on switches, the model parameters were encapsulated into the packets using an in-band protocol built upon MPLS. While most existing FL frameworks transfer ML parameters in the application layer of the messages, we design the MPLS protocol to encode the parameters in order to simplify programming the devices, which can easily parse intermediate layer protocols and work with multiple higher level protocols, *e.g.*, TCP/UDP/QUIC. We show in Fig. 3 the usage of MPLS as a packet header to carry the model's gradients. This MPLS-based approach thus provides remarkable flexibility and is compatible with most existing network devices. MPLS works by prefixing packets with an MPLS header containing one or more labels and forming a label stack. Each entry in the label stack contains four fields, where the main part is a 20-bits used for the *label* value. Although this protocol is typically used to select the forwarding path, in P4FL we use it only to carry the information about the model weights in the label field, removing the Label Distribution Protocol (LDP). The popularity of the MPLS protocol would favor considerable interoperability among both switches and hosts. Since a model is characterized by hundreds of weights, we encode the entire list of parameters into multiple MPLS label fields. In addition to these parameters, we need to encapsulate some additional configurations to make the FL system work. For example, a preliminary handshake phase is required to exchange communication parameters, and the server sends actions to clients specifying when to start training and the evaluation phase. Along with the important abstraction, however, coding using P4 brings many limitations and challenges. First of all, "for" loops are not available, and the programmer can write "if-else" blocks only in selected areas. Then, the P4 language supports only integer and bit operations, so it is impossible to perform aggregation directly on model parameters, which are usually floating-point values. To solve this limitation, we apply a quantization technique that scales the floating-point value to an 8-bits integer [28]. This approach is commonly used in deep learning models as it maps the real values to integers representable by the bit-width of the quantized representation and then rounds each mapped real value to the closest integer value.

On top of these limitations, the P4 language does not support arithmetical division operations, so the average calculation became challenging for any chosen aggregation method. As explained in Section III-B, the method used for aggregation is a variation of the average operation. To address this issue, we modified the default P4 frontend and backend compiler to enable *extern* functions [29]. The additional constructs are functions implemented in C/C++ that are called directly in the P4 program as new P4 constructs. Because this construct considers the final platform and its possible limitations during the compilation, this approach is considered portable over all P4-enabled switches. Thus, the use of externs results in a more elegant code solution since they are implemented outside the switch core, reducing side effects and risks that can be caused by an alternative approach based on modification in the switch pipeline implementation.

Lastly, the forwarding behavior of the switch is modified so that only the aggregated value is sent to the server, which value can be obtained only when all packets from connected hosts are received.

B. FEDERATED LEARNING FRAMEWORK IN P4FL

On clients and the central server runs the FL framework, whose training process relies on exchanging request-response messages. We defined a new communication pattern based on a hierarchical FL setting composed of the central server, edge switches (also acting as proxies in the FL framework), and the clients involved in such a process.

Despite the large number of existing FL frameworks and optimizations on ML model training, the main operations of any architecture are the exchange of model parameters. In this paper, we consider a centralized version (since the most common one) where both the clients and the server include model parameters in the messages via MPLS. Because the server controls the overall procedure, its messages contain instructions on the following actions to be performed by the clients, along with the model parameters. Compared to the vanilla FL schema, where the server establishes

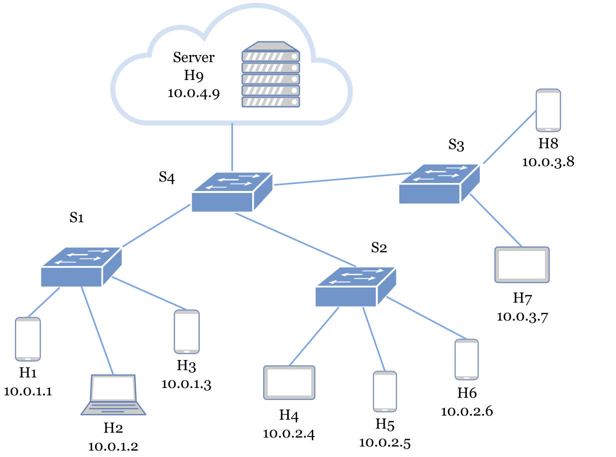


FIGURE 4. Network topology used in the evaluation of the FL architecture.

connections with each client that joins the training process, in our version, since it receives aggregated values from the switches, it establishes connections only with special hosts representative of the entire subnetwork.

The central server handles the overall FL process by sending messages to the clients with instructions to train or evaluate the model locally. In particular, the central server is in charge of starting the FL process, initializing the strategy, delimiting the round, sending requests to the clients, and aggregating the responses. As soon as the server is started, it manages M parallel connections, where M is the number of different sites. For each site, a leader is elected to maintain active connections with keepalive messages. Then, the server requests the initial communication parameters to these representatives and, after receiving the responses, asks all clients to start the training evaluation round until convergence. A specific process on the server manages these message exchanges, parses the packets, and extracts the data, while another process manages the FL aggregation process. In this paper, we use the most common strategy, the same of FedAvg [5], which performs the average of the collected neural network weights. However, since we focus on the network architecture, this aggregation strategy is just a policy of P4FL. Our architecture comes with other strategies and optimization methods, such as FedAdaGrad, FedYogi, and FedAdam [30], which are adaptive and can be used for different scenarios, *e.g.*, with heterogeneous data.

Moreover, the neural network model can undergo several other optimizations, such as 1-bit neuron [31], to further reduce model complexity and overhead. However, our main contribution focused mainly on how network devices can assist the FL centralized training process, resulting in an architecture that is general enough and can be applied to other model variations.

IV. EVALUATION RESULTS

In this section, we present experimental results obtained that confirm the validity of our architecture. We start describing the testbed and settings used during the experimental campaign, then we quantify the advantages in the model accuracy and the reduced additional traffic, concluding with evaluating the feasibility of the overall solution.

TABLE 1. Performance comparison between the traditional FedAvg and our solution. P4FL drastically reduces the number of bytes exchanged while increasing the global accuracy.

Metric	FedAvg	FL-hierarchical	P4FL
Network traffic [kB]	423.07	301.44	163.98
Global Loss	0.4376	0.3781	0.3605
Global Accuracy [%]	77.97	81.63	82.35

A. EXPERIMENTAL SETTINGS

To validate P4FL's benefits, we deploy it over Mininet, a network emulator that allows reproducing arbitrary virtual networks for fast simulations [32]. Since it is designed explicitly for Software-Defined Networking (SDN), it can also support P4-compatible switches via Behavioral Model Version 2 (BMV2), which allows compiling a P4 program into packet-processing actions of C++11 software switches. Our modified instance of the P4 compiler is built upon standard version *p4-16*. As the P4 behavioral model we use is a soft switch, "externs" are typically defined in a separate file, often in a C or C++ programming language, and linked to the P4 program during compilation. This allows the P4 program to use the functionality provided by the extern object without having to know the implementation details. In P4FL, our extern object is implemented in C++, extending the *ExternType* class used to interact with the extern instances in our P4 code.

The task of the ML model is to predict whether or not a patient has diabetes by using a list of diagnostic measurements contained in the Pima Indians Diabet dataset [33], a dataset widely used for benchmarking, originally compiled by the National Institute of Diabetes and Digestive and Kidney Diseases. The dataset consists of 1596 samples, where each item contains eight features as input: number of times pregnant, plasma glucose concentration at 2 hours in an oral glucose tolerance test (GTIT), diastolic blood pressure, triceps skin fold thickness, 2-hour serum insulin, Body Mass Index, diabetes pedigree function, age. The dataset is first split into training set (80%) and test set (20%), then the training set is split into equal-sized subsets then assigned to each client. The neural network we trained comprises two hidden layers of 64 and 32 neurons, respectively. These hyper-parameters are selected after cross-validation in a single client setting. The machine used for tests presented 20 GB of RAM and four processors. We consider the default transport protocol of the FL communications to be UDP to make the packet parsing smoother and to manage (both in the application and switches) efficiently the multicast transmission of packets from the server to all clients, *e.g.*, for model configuration and rounds definition.

TABLE 2. Training time (TR) and evaluation time (EV) evolution over 3 rounds for all clients in the network.

Host	TR - 1 (s)	EV - 1 (s)	TR - 2 (s)	EV - 2 (s)	TR - 3 (s)	EV - 3 (s)
H1	50.07	1.13	39.80	2.18	42.52	4.43
H2	49.85	1.36	41.52	0.47	38.41	4.56
H3	50.47	0.74	38.62	3.33	42.73	4.22
H4	44.15	1.46	41.12	0.94	42.19	4.66
H5	53.33	1.78	41.13	0.83	82.50	0.47
H6	53.67	1.09	39.93	1.97	40.29	4.76
H7	44.45	0.74	41.68	0.38	41.35	1.52
H8	55.18	0.65	34.78	7.21	40.40	2.33

We replicate the hierarchical topology with eight clients connected to a local network node and the external server residing in a different network, as shown in Fig. 4. This scenario is inspired by the healthcare industry, where the sites have modest computing resources and participate in the process of training a global model without exchanging private and sensitive medical data [4]. In this case, we consider 3 different sites, where every link has 100 Mbps bandwidth, but the link between $S4$ and the server is of 50 Mbps bandwidth and the delay of 20ms. Each client sends the neural network's weights to the switch using the MPLS header.

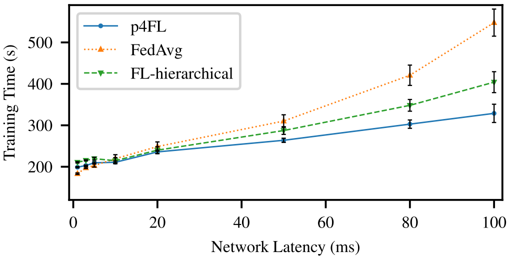


FIGURE 5. Training time for increasing network latency. P4FL can keep the training process limited in time.

B. MODEL PERFORMANCE AND NETWORK TRAFFIC ANALYSIS

We start comparing the critical parameters of P4FL against the traditional *FedAvg*, which is the widely used solution in FL-based systems, and *FL-hierarchical*, a hierarchical FL setting in which intermediate aggregation is performed by edge computers instead of network devices. The difference between P4FL and *FL-hierarchical* also lies in the protocol used for gradient exchange: gRPC for *FL-hierarchical* and MPLS for P4FL. Table 1 summarizes the obtained results in terms of training time, network traffic, and final global accuracy and loss after 10 rounds. In the FedAvg case, each client trains the model individually and sends the learned parameters to the server at the end of each round.

The main result brought by P4FL is the reduced amount of network traffic that is reduced by a factor of more than 60%, as observed in the table. In general, our architecture allows reducing the traffic by a factor of S/N where S is the number of switches, and N is the number of training nodes in the process. Also, FL-hierarchical, based on application layers for sending model gradients to the edge and cloud, results in additional bytes in the network. This result is extremely important in cross-silo scenarios, where the central server may be far from sites, *e.g.*, hospitals, and network latency represents an issue originating synchronization problems [6], [34].

Another relevant obtained result regards the accuracy of the global model. First, we can note how the simplicity of the neural network model hinders an optimal prediction, but in the paper, we mainly focused on the network architecture rather than on the design of a specialized ML model. However, we can observe how P4FL leads to higher accuracy and lower loss compared to the alternative approach of FedAvg. Even if the same number of clients takes part in the training process, the preliminary aggregation performed by the switches enables the server to perform a weight adjustment over a more stable version of the received models. In fact, the loss and accuracy are similar to FL-hierarchical despite being slightly higher.

We then compare the training time of these three versions at varying latency between the clients and server (Fig. 5). When the server is located near the sites, the training processes of the approaches last quite similarly, but when latency increases, we can observe significant improvements. Having the switches to provide a first response to the clients, the next phase, *i.e.*, the evaluation phase, of clients, can start without waiting for final aggregation on the server. Notably, the in-network computing offered by P4FL reduces the training time even compared to a more traditional hierarchical environment. This result is extremely important for many cloud-centric solutions where the server resides in a different city or infrastructure, *e.g.*, a hospital. We expect, however, that in IoT scenarios, as explained in [35] and [36], the benefits in the required time would be even greater, and P4FL can positively tolerate the presence of stragglers, *i.e.*, clients that take much longer to report their output than other nodes.

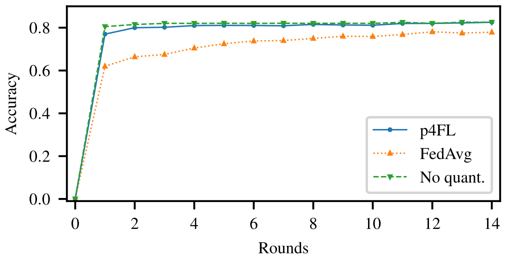


FIGURE 6. Aggregated accuracy comparison on the central server in 15 rounds.

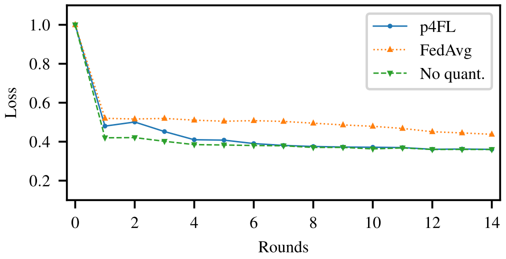


FIGURE 7. Loss trend comparison on the central server in 15 rounds.

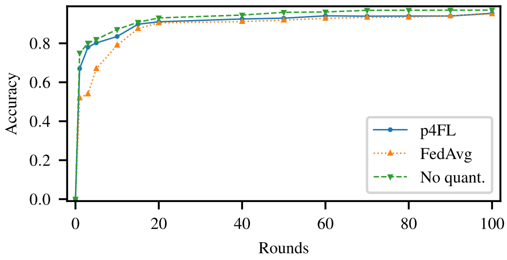


FIGURE 8. Aggregated accuracy comparison on the central server in 100 rounds for clients training a ResNet-56 model over the CIFAR-10 dataset.

TABLE 3. Resource consumption per intermediate node for the three possible alternatives. The overhead introduced with P4FL is minimal and demonstrates its feasibility.

Solution	RAM per int. node [%]	CPU per int. node [%]
FedAvg	0.19	0.02
FL-hierarchical	10.51	24.27
P4FL	2.63	3.08

To better understand improvements in our approach and study its behavior, we report the time spent for each client over 3 rounds in Table 2. In particular, we show both training time, *i.e.*, the time elapsed between the reception of one server fit request message and the transmission of client response, and evaluation time, *i.e.*, the time elapsed between the reception of one server evaluate request message and the transmission of client response. The alternate steps in each round are canonical in FL settings based on neural network training [5]. Clearly, the evaluation of the model at the end of each round is much faster than training, but we can observe a considerable variance in the training time of hosts. To this end, we argue how having a network switch that preliminary aggregates the parameters limits such variances and uniforms the duration of each round (training + evaluation). This phenomenon is one of the reasons behind the improvements compared to a traditional approach.

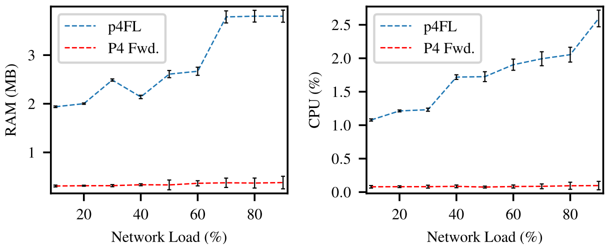
Later, we consider the accuracy and loss at the server side after 15 rounds of FL (Figure 6 and Figure 7). We did not consider the hierarchical setting, as it can be very similar in model performance to our version based on in-network computing. However, to evaluate the impact of quantization, we consider the centralized FL version where no quantization is introduced. From the graphs, we can observe that after almost three rounds, both accuracy and loss are stable, and only minor improvements take place before the run stops at round 15. With respect to the quantization method, instead, we can also conclude that the chosen algorithm allows an efficient training of the model FL while maintaining the final accuracy. The aggregation performed by switches is effective in reaching such stability, and our solution attains an accuracy and a loss that are comparable to the model with no quantization. Aside from the greater accuracy in our approach, we can observe how the noise in such evolution is limited in P4FL, where FedAvg is then slower to converge in both accuracy and loss metrics.

Furthermore, we measure the performance when training a Convolutional neural network (CNN), one of the most popular deep learning architectures, on the CIFAR-10 dataset, consisting of color images (32 × 32 RGB with three channels) classified into 10 classes, e.g., ships, cats, and dogs, and partitioned into 50, 000 training samples and 10, 000 test samples [37]. Such dataset is divided into five training batches and one test batch, where each batch has 10000 images and the test batch includes 1000 randomly-selected images from each class. The CNN architecture is the ResNet-56, a pre-trained deep learning model having 56 layers [38]. To avoid overfitting to the training set, the identity mapping used in ResNet allows the model to bypass a CNN weight layer if the current layer is not necessary. In detail, the convolutional layers have 56 weighted layers in total, mostly 3 × 3 filters, which ends with a global average pooling layer and a 1000-way fully-connected layer with softmax.

Fig. 8 shows the corresponding accuracy during training such a model. We can first observe that the FL model with no quantization leads to slightly higher accuracy, but the difference with our solution is negligible. Moreover, we can note that, although the attained accuracy is similar between FedAvg and P4FL, the partial aggregation in our solution can speed up the global model convergence. This outcome is particularly important to demonstrate how our protocol can scale to deeper neural networks, consisting of multiple hidden layers, and a larger dataset.

C. SWITCH OVERHEAD

We then measure the CPU consumption and RAM usage of the intermediate node to understand the impact on the



(a) (b)

FIGURE 9. (a) RAM and (b) CPU consumption of the considered approach in a single switch.

P4 switch. We highlight that the intermediate node in FedAvg and P4FL refers to the switch, which in FedAvg only forwards packets, while in our solution it also aggregates the gradients. In FL-hierarchical, instead, it refers to the intermediate edge server. We report in Table 3 the average resource usage for all the intermediate nodes of the network. It can be observed that FedAvg represents the lowest consumption since the nodes only forward packets, while FL-hierarchical consumes significant memory as well as processing capacity. With P4FL, on the other hand, the overhead introduced by the *extern* and the gradients aggregation is only minimal. This positive outcome is the result of an efficient program design, where the majority of operations are possible in P4 (grouping, packet offset detection, gradients dispatching) and are thus effective and fast, while only the average operation occurs in the external space. This result is therefore important to validate the efficient program design and the feasibility of P4FL in a real switching pipeline.

Finally, we consider the impact of our solution in terms of processing efficiency when background traffic is present in the network. While P4FL achieves valuable performance, it is crucial to evaluate whether its implementation requires a set of specific hardware components. For this reason, we computed the RAM and CPU consumption of P4FL in a machine with 4*GB* RAM and 1-core CPU and compared it to a P4 program that only performs forwarding (Figure 9). Furthermore, to study how P4FL behaves at different network loads, we used the *iperf3* tool and changed the bandwidth according to the network load we wanted to achieve. As visible from Figure 9a, at low network loads (10%-50%) P4FL uses around 2*MB* of memory, while it increases its consumption only at a higher network load (from 70%). The same considerations are valid when evaluating the CPU (Figure 9b). As visible from the figure, P4FL requires around $1\% - 1.5\%$ of CPU usage when the network load is low (from 10%-40%), while it only requires up to 2.5% when the network is fully congested (90%). This result is key in assessing that the amount of RAM and CPU is easily accessible in real implementations. For example, the Intel Tofino with the Data Processing Unit (DPU) Module integrated, can support up to 96 GB of RAM and 26 cores. Thus, the required memory and processing capabilities are of little impact, meaning that P4FL can find great applicability over SmartNICs or Tofinos.

V. CONCLUSION

In this paper, we presented P4FL, an FL architecture that uses P4 switches to compute intermediate aggregations of the model parameters. This approach allows reducing network traffic, alleviating the bottleneck effect on the central FL Server, and further accelerating the entire training process. We augmented the standard behavior of network switches in order to support gradient aggregation, providing external function support and an in-band parameters transmission protocol that facilitates the overall operations. Results obtained over an emulated network topology composed of a central server, intermediate P4 switches performing aggregations, and different clients, are promising. The generated network traffic can be reduced drastically, allowing scaling over larger networks. The aggregation performed by switches also has a beneficial impact on the final learned model. We leave as an open question the investigation of the effect of such aggregation on the model accuracy, and the exploration of how our P4FL or other solutions may aggregate efficiently in more challenging network conditions.

REFERENCES

[1] K. Bonawitz, H. Eichner, W. Grieskamp, and D. Huba, "Towards federated learning at scale: System design," in *Proc. Mach. Learn. Syst.*, vol. 1, 2019, pp. 374–388.

[2] A. Sacco, F. Esposito, and G. Marchetto, "A federated learning approach to routing in challenged SDN-enabled edge networks," in *Proc. 6th IEEE Conf. Netw. Softwarization (NetSoft)*, Jun. 2020, pp. 150–154.

[3] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," 2016, *arXiv:1610.02527*.

[4] O. Aouedi, A. Sacco, K. Piamrat, and G. Marchetto, "Handling privacy-sensitive medical data with federated learning: Challenges and future directions," *IEEE J. Biomed. Health Informat.*, vol. 27, no. 2, pp. 790–803, Feb. 2023.

[5] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Artif. Intell. Statist.*, 2017, pp. 1273–1282.

[6] P. Kairouz, P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, and K. Bonawitz, "Advances and open problems in federated learning," *Found. Trends Mach. Learn.*, vol. 14, nos. 1–2, pp. 1–210, 2021.

[7] J. Mills, J. Hu, and G. Min, "Client-side optimization strategies for communication-efficient federated learning," *IEEE Commun. Mag.*, vol. 60, no. 7, pp. 60–66, Jul. 2022.

[8] R. Firouzi, R. Rahmani, and T. Kanter, "Federated learning for distributed reasoning on edge computing," *Proc. Comput. Sci.*, vol. 184, pp. 419–427, Jan. 2021.

[9] L. Liu, J. Zhang, S. H. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2020, pp. 1–6.

[10] A. Sapio, M. Canini, C.-Y. Ho, J. Nelson, P. Kalnis, C. Kim, A. Krishnamurthy, M. Moshref, D. Ports, and P. Richtarik, "Scaling distributed machine learning with in-network aggregation," in *Proc. 18th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2021, pp. 785–808.

[11] F. Chen, P. Li, and T. Miyazaki, "In-network aggregation for privacy-preserving federated learning," in *Proc. Int. Conf. Inf. Commun. Technol. Disaster Manage. (ICT-DM)*, Dec. 2021, pp. 49–56.

[12] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[13] A. V. Ventrella, F. Esposito, A. Sacco, M. Flocco, G. Marchetto, and S. Gururajan, "APRON: An architecture for adaptive task planning of Internet of Things in challenged edge networks," in *Proc. IEEE 8th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2019, pp. 1–6.

[14] A. Sacco, F. Esposito, and G. Marchetto, "Resource inference for task migration in challenged edge networks with RITMO," in *Proc. IEEE 9th Int. Conf. Cloud Netw. (CloudNet)*, Nov. 2020, pp. 1–7.

[15] Z. Zhang, Y. Yang, Z. Yao, Y. Yan, J. E. Gonzalez, and M. W. Mahoney, "Benchmarking semi-supervised federated learning," 2020, *arXiv:2008.11364*.

[16] M. Shafiq, Z. Tian, A. K. Bashir, X. Du, and M. Guizani, "CorrAUC: A malicious bot-IoT traffic detection method in IoT network using machine-learning techniques," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3242–3254, Mar. 2021.

[17] M. Shafiq, Z. Tian, Y. Sun, X. Du, and M. Guizani, "Selection of effective machine learning algorithm and bot-IoT attacks traffic identification for Internet of Things in smart city," *Future Gener. Comput. Syst.*, vol. 107, pp. 433–442, Jun. 2020.

[18] C. H. van Berkel, "Multi-core for mobile phones," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2009, pp. 1260–1265.

[19] J. S. Ng, W. Y. B. Lim, Z. Xiong, X. Cao, J. Jin, D. Niyato, C. Leung, and C. Miao, "Reputation-aware hedonic coalition formation for efficient serverless hierarchical federated learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2675–2686, Nov. 2022.

[20] H. Guo, W. Huang, J. Liu, and Y. Wang, "Inter-server collaborative federated learning for ultra-dense edge computing," *IEEE Trans. Wireless Commun.*, vol. 21, no. 7, pp. 5191–5203, Jul. 2022.

[21] D.-J. Han, M. Choi, J. Park, and J. Moon, "FedMes: Speeding up federated learning with multiple edge servers," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3870–3885, Dec. 2021.

[22] F. P. Lin, S. Hosseinalipour, S. S. Azam, C. G. Brinton, and N. Michelusi, "Semi-decentralized federated learning with cooperative D2D local model aggregations," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3851–3869, Dec. 2021.

[23] W. Y. B. Lim, J. S. Ng, Z. Xiong, J. Jin, Y. Zhang, D. Niyato, C. Leung, and C. Miao, "Decentralized edge intelligence: A dynamic resource allocation framework for hierarchical federated learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 3, pp. 536–550, Mar. 2022.

[24] L. Mai, L. Rupprecht, A. Alim, P. Costa, M. Migliavacca, P. Pietzuch, and A. L. Wolf, "NetAgg: Using middleboxes for application-specific on-path aggregation in data centres," in *Proc. 10th ACM Int. Conf. Emerg. Netw. Experiments Technol.*, Dec. 2014, pp. 249–262.

[25] J. Fang, G. Zhao, H. Xu, C. Wu, and Z. Yu, "GRID: Gradient routing with in-network aggregation for distributed training," *IEEE/ACM Trans. Netw.*, early access, Feb. 22, 2023, doi: 10.1109/TNET.2023.3244794.

[26] M. Baldi, G. Marchetto, and Y. Ofek, "A scalable solution for engineering streaming traffic in the future internet," *Comput. Netw.*, vol. 51, no. 14, pp. 4092–4111, Oct. 2007.

[27] M. Baldi, M. Corrà, G. Fontana, G. Marchetto, Y. Ofek, D. Severina, and O. Zadedyurina, "Scalable fractional lambda switching: A testbed," *J. Opt. Commun. Netw.*, vol. 3, no. 5, pp. 447–457, May 2011.

[28] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," 2020, *arXiv:2004.09602*.

[29] *The P4 Language Specification*. Accessed: Mar. 13, 2023. [Online]. Available: https://p4.org/p4-spec/docs/P4-16-v1.0.0-spec.html

[30] S. Reddi, Z. Charles, M. Zaheer, Z. Garrett, K. Rush, J. Konečný, S. Kumar, and H. B. McMahan, "Adaptive federated optimization," 2020, *arXiv:2003.00295*.

[31] C. S. Güntürk and W. Li, "Approximation of functions with one-bit neural networks," 2021, *arXiv:2112.09181*.

[32] R. L. S. de Oliveira, C. M. Schweitzer, A. A. Shinoda, and L. R. Prete, "Using mininet for emulation and prototyping software-defined networks," in *Proc. IEEE Colombian Conf. Commun. Comput. (COLCOM)*, Jun. 2014, pp. 1–6.

[33] P. Bennett, T. Burch, and M. Miller, "Diabetes mellitus in American (PIMA) Indians," *Lancet*, vol. 298, no. 7716, pp. 125–128, Jul. 1971.

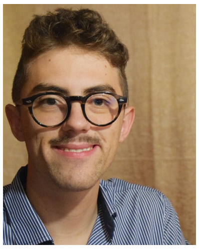
[34] S. Silva, B. A. Gutman, E. Romero, P. M. Thompson, A. Altmann, and M. Lorenzi, "Federated learning in distributed medical databases: Meta-analysis of large-scale subcortical brain data," in *Proc. IEEE 16th Int. Symp. Biomed. Imag. (ISBI)*, Apr. 2019, pp. 270–274.

[35] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar, "Slow and stale gradients can win the race: Error-runtime trade-offs in distributed SGD," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2018, pp. 803–812.

[36] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proc. Mach. Learn. Syst.*, vol. 2, 2020, pp. 429–450.

[37] *The Cifar-10 Dataset*. Accessed: Mar. 13, 2023. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.



ALESSIO SACCO (Member, IEEE) received the M.Sc. and Ph.D. degrees (summa cum laude) in computer engineering from Politecnico di Torino, Italy, in 2018 and 2022, respectively. He is currently an Assistant Professor with Politecnico di Torino. His research interests include architecture and protocols for network management, implementation and design of cloud computing applications, algorithms and protocols for service-based architecture, such as software-defined networks (SDN), used in conjunction with machine learning algorithms.

ANTONINO ANGI (Student Member, IEEE) received the M.Sc. degree in computer engineering (major in data science) from Politecnico di Torino, Italy, in 2020, where he is currently pursuing the Ph.D. degree. His research interests include protocols for network architecture and management; such as natural language processing (NLP) and machine learning algorithms to software-defined networks (SDN) and intent-based networks (IBN), used in conjunction with data-plane programming languages.

GUIDO MARCHETTO (Senior Member, IEEE) received the Ph.D. degree in computer engineering from Politecnico di Torino, in 2008. He is currently an Associate Professor with the Department of Control and Computer Engineering, Politecnico di Torino. His research interests include distributed systems, formal verification of systems and protocols, network protocols, and network architectures.

FLAVIO ESPOSITO (Member, IEEE) received the M.Sc. degree in telecommunication engineering from the University of Florence, Italy, and the Ph.D. degree in computer science from Boston University, in 2013. He is currently an Associate Professor with the Department of Computer Science, Saint Louis University (SLU). His research interests include network management, network virtualization, and distributed systems. He was a recipient of several awards, including several National Science Foundation Awards and the Comcast Innovation Award, in 2021.

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement