Distributed Predictive QoS in Automotive Environments under Concept Drift

Georgios Drainakis*, Panagiotis Pantazopoulos*, Konstantinos V. Katsaros*,
Vasilis Sourlas*, Angelos Amditis*, Dimitra I. Kaklamani†
*Institute of Communication and Computer Systems (ICCS), Athens, Greece,
{giorgos.drainakis, ppantaz, k.katsaros, v.sourlas, a.amditis} @iccs.gr
†National Technical University of Athens (NTUA), Athens, Greece, dkaklam@mail.ntua.gr

***Abstract*—As network connectivity increasingly shapes modern vehicular applications, in-advance knowledge of Quality-of-Service (QoS) degradation could unlock the potential for efficient and safer mobility. Predictive QoS (pQoS) has long resorted to traditional Machine Learning (ML) methods but distributed approaches, such as Federated Learning (FL) have lately emerged as alternatives promising performance (and privacy) gains. Vehicular environments however, appear prone to concept drift; frequent changes in the underlying client data distribution degrade the ML model's accuracy. To mitigate drift, existing FL algorithms employ continuous model training at the expense of valuable network resources. DareFL, our drift management algorithm for distributed pQoS a) detects drift without violating FL's privacy restrictions, and b) unlike previous works, carefully schedules the (re-)training process thereafter, achieving remarkably reduced resource consumption. For evaluation purposes, we release a high-fidelity vehicular network simulator. We then realize two intuitive drift scenarios over which, DareFL consistently yields comparable accuracy to existing FL schemes, while saving up to 70% on network resources.**

***Index Terms*—Predictive Quality of Service, Federated Learning, Concept drift**

I. Introduction

Cooperative, connected and automated mobility (CCAM) services *e.g.,* automated driving (AD) functions, tele-operated driving (ToD), platooning*, etc.* are expected to unlock a series of benefits related to road safety, traffic efficiency and driving comfort. To do so, they heavily rely on mobile network connectivity, often posing stringent Quality of Service (QoS) requirements in terms of data rate, packet loss, delay*, etc.* Failure to meet these requirements *e.g.,* due to undesired network conditions, can lead to service degradation, which in turn may reduce driving experience or even compromise safety. To address such unexpected events, the notion of predictive Quality of Service (pQoS) has been introduced by the 5G Automotive Association (5GAA) [1]. pQoS enables mobile networks to provide *in-advance* notifications about predicted QoS changes. A critical situation can then be avoided by timely adjusting the behavior of the CCAM service *e.g.,* hand over an automated vehicle's control to the driver.

pQoS has been realised through Machine Learning (ML) [2], grace to its inherent ability to capture the volatility of cellular QoS parameters. ML models are typically developed using Centralized Learning (CL); data is collected from client devices *e.g.,* smartphones, on-board units*, etc.* in centralized (clusters of) servers, where model training occurs thereafter. Distributed ML solutions for pQoS have recently emerged as alternatives, exhibiting ease of scalability, privacy-preservation and most importantly communication cost reduction [3], [4]. A typical example is Federated Learning (FL), where training is collaboratively carried out by the clients; user data remains at the client devices and only the model updates are shared to the server [3].

In an ever-changing network environment, a major challenge that arises is concept drift: client data undergoes changes across time due to seasonality, trends, user habit variations*, etc.* [5]. If not mitigated *e.g.,* by re-training the drifted ML model, concept drift can lead to model drift *i.e.,* degradation of the model's accuracy. In centralized pQoS, whereby data is centrally collected, drift management *i.e., detection* and *mitigation* is performed by directly applying statistical techniques to the raw data [6]. Distributed (FL) environments however, pose limitations that render the application of centralized solutions inefficient: a) the FL server has no access to client data, therefore cannot directly detect changes in the underlying data distributions and b) client devices suffer from resource scarcity *e.g.,* processing capacity, storage, battery, and are often unreliable *e.g.,* due to connection unavailability, drop-outs*, etc.* therefore, cannot effectively host drift management mechanisms compared to a typically always-on server. As such, we pose two fundamental research questions: 1) *How is drift detection facilitated in FL, subject to the above-mentioned restrictions?* and 2) *How to reduce the resource consumption, typically inflicted by drift mitigation mechanisms e.g., re-training*?

Drift management has been addressed for FL settings [7], [8], however existing works either *disregard* FL's data privacy restrictions or employ *continuous* training schemes with increased resource consumption, as we also demonstrate in this paper. In view of these research gaps, we propose our Drift-aware resource-efficient algorithm for FL (DareFL), tailored to resource-constrained vehicular network environments. Our solution engineers a way to turn existing centralized statistical drift detection techniques [5] to distributed, without compromising data privacy. Upon detection, drift mitigation is performed by re-training the drifted ML model. The (re-)training periods are carefully selected via a control mecha-

nism, ensuring that both the ML model remains accurate at all times and the underlying network resource consumption is kept low.

Addressing the limited availability of public pQoS data [9], we have generated synthetic datasets via a network and traffic co-simulation coupled with real-world maps, to provide a careful assessment of our solution. Our datasets capture two distinct pQoS drift scenarios, one corresponding to the wireless communication dynamics and another related to changes of user (driving) behavior. The datasets are utilized via a distributed ML simulator that emulates the training process, at the same time capturing energy and bandwidth consumption aspects, based on credible measurements and commercial product bench-marking. Our contribution is the following: 1) we introduce an FL-based pQoS framework, 2) we propose a novel drift management algorithm *tailored* for FL that reduces resource consumption, while respecting the restrictions posed by FL deployment, 3) we develop synthetic pQoS datasets with realistic drift instances and a distributed ML simulator that are both shared publicly to enable experiments reproducibility.

In our results, we evaluate our proposed drift management solution (DareFL) in presence of drift against the baselines of a) Vanilla FL [4] and b) a continuous training scheme, as well as c) a representative state-of-the-art (SotA) drift-mitigation solution [8]. Simulation results over the two scenarios under study suggest that DareFL saves up to 70% on the resources for the network infrastructure, the involved clients and the server, whilst exhibiting an average of 10% reduction of accuracy against its competing resource-hungry schemes. The rest of the paper is organized as follows. Related work is presented in Sec. II. In Sec. III we analyze our system architecture, followed by our simulation framework in Sec. IV. In Sec. V, we provide a simulation-based performance evaluation of DareFL. Lastly, we conclude in Sec. VI.

II. Related Work

Drift management *i.e., detection* and thereafter *mitigation* has been extensively studied for CL, where data, training nodes and ML models are centrally co-located. Detection occurs via *data distribution-based* or *performance-based* detectors [5]. The former detectors continuously compare the training data against historical data to find drifts via statistical tests *e.g.,* Kullback-Leibler divergence [6]. *Performance-based* detectors, track the changes in the model's inference error, based on the Probability Approximately Correct (PAC) concept [5]. PAC states that a model's performance degradation implies that the learned relationship between the input data and the prediction variable is no longer valid *i.e.,* a concept drift has occurred. Upon detection, mitigation takes place by either re-training the models with new data, combining old and new models or by model re-configuration [6].

Concept drift management for FL includes a) Personalized learning, b) Asynchronous FL schemes and c) Continuous Learning techniques. In *Personalized learning* clients develop user-specific models by re-training a generic global model [10]. Other alternatives include training both a global and

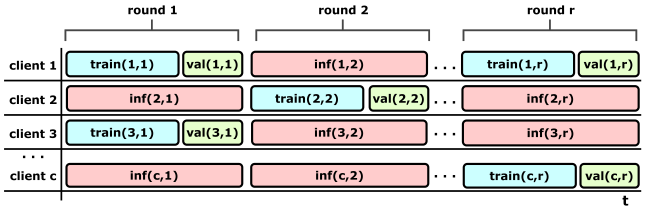

Fig. 1: Vanilla FL pQoS framework

local client models, to allow the clients to choose the best fit according to their data [11] or meta-data from other clients [12]. This can be even generalized so that clients obtain multiple personalized models from the server and perform aggregation locally (client side) [13]. Maintaining multiple models however as Personalized learning suggests can add to the system's complexity, especially in a scaled FL environment with thousands of clients. *Asynchronous Federated Learning* on the other hand suggests that each client trains and uploads its model when needed *e.g.,* when a drift occurs. Drift detection occurs on the client-side *e.g.,* by comparing new to historical data [3] or by assessing the global model's inference results [14], [15]. Drift mitigation is also handled by the clients via re-training [3], ensemble methods [14], or by adapting their local cost function [15]. Although promising, asynchronous techniques introduce an extra layer of complexity (computational, storage) to the resource-constrained clients. Finally, *Continuous Federated Learning* (ConFL) suggests to repeatedly re-train the ML model and mitigate potential drift effects. In [7], clients detect drifts using euclidean distance metrics; then the clients that experience drift are isolated from the training process by the server. This approach however, cannot be implemented in a generalized global drift that applies to all clients. Server-side drift detection has been also proposed via convex optimization [16]; mitigation occurs thereafter by adapting the number of training epochs, but is demonstrated to mainly work for near-stationary environments. Similarly to ConFL approaches, in AdaptFL [8] drift is detected by comparing the received client models in the server-side via moving average. Upon detection, the server adapts the learning rate hyper-parameter for the next training round and communicates this information to the clients. In the presence of drift the learning rate is increased, otherwise it gradually decreases. Though ConFL/AdaptFL is shown to successfully mitigate concept drift it requires constant training, which in absence of drift consumes *excessive* network resources with no extra performance gains.

III. QoS Prediction Scheme

We hereby introduce the fundamental concepts that facilitate pQoS in distributed automotive environments. Consider a centralized server *e.g.,* in a cloud infrastructure and several client-vehicles. Each vehicle is equipped with a) in-vehicle sensors for raw data acquisition, b) on-board processing units for computations and c) a modem to allow for connectivity to the server via the cellular network. Training of a pQoS ML model uses data features related to network properties and client mobility *e.g.,* signal strength, position, speed, *etc.* [9].

A. Vanilla Federated Learning framework for pQoS

Training in FL occurs in consecutive cycles *i.e.,* training rounds R, as shown in Fig. 1. Assume a total number of C client-vehicles. In each round $r \in [1, R]$ the server randomly selects a specified number of K clients for training *i.e.,* trainers and M clients for inference *i.e.,* testers, so that $K + M \leq C$ [17]. The global ML model is distributed to the selected trainers and testers. Then, each trainer trains the global model with its acquired data, creating a new local model. Each tester on the other hand tests the model on the current round's data. Upon training completion, the server collects all local models and aggregates them to an updated global model, via the FedAvg algorithm [8]. The server also collects the inference results from the testers, to produce an evaluation report of the model's performance and marks the end of round r. Subsequently, the server re-distributes the updated global model to another set of K trainers and M testers, initiating the next training round $(r + 1)$. The process repeats until termination criteria are met *e.g.,* maximum number of rounds is reached [4]. In terms of scheduling, the training process is divided into equally timed rounds of specified duration q. Clients are constantly collecting environment data (features) at a frequency f, but only use those collected within one round duration q. As such in every round, each selected trainer $c \in [1, K]$ splits its data into a training *train(c, r)* and a validation dataset *val(c, r)*. Similarly, each tester $c \in [1, M]$ creates an inference dataset *inf(c, r)*. The respective datasets are depicted in blue, green and red color in Fig. 1. Note that prior to the training process we allow for a "warm-up" period, which includes a series of test-runs with limited data for calibration in the server-side.

B. Concept drift management for distributed pQoS

Concept drift in ML refers to the phenomenon whereby the statistical properties of data change over time in unforeseen ways. Vehicular environments have been shown to experience frequent drifts whether in a suburban, rural or highway areas [9]. SotA FL algorithms either do not account for drift or resort in a repetition of the training phase, which can lead to resource-waste, as we demonstrate in Sec. V. As such, we introduce our novel Drift-aware resource-efficient algorithm for FL (DareFL) that serves a two-fold objective: 1) It timely halts training upon convergence and thus, reduces resource waste, and 2) It accurately detects drift and carefully orchestrates re-training as a targeted drift mitigation technique. Eventually DareFL monitors the models' performance and decides if further training is required. While training occurs, the round is marked as active; when training pauses, it is marked as idle. During idle rounds, client devices save on the processing-related resources *e.g.,* power consumption and the server-client communication cost is decreased. Our solution is based on the accurate detection of: a) the completion of the training process (convergence), and b) a change in the underlying data distribution (drift), to timely halt or (re-)initiate the training, respectively.

Deducing these events can be challenging in FL, due to privacy constraints and thus, we leverage on the inference results that indicate the ML model's performance. Such performance-based detectors assume that the model's performance degrades due to the effects of over-fitting or due to concept drift (see PAC model in Sec. II). Model performance evaluation is based on relative metrics that compare the model's performance against that of a baseline naive algorithm [18]. For pQoS (time-series), we employ the following rolling-means algorithm as our baseline predictor: the (naive) prediction $\widehat{y_i}$ of the sample's i dependent variable y_i (ground truth) in time t_i is the mean value of the dependent variable's last w values, where w stands for the time-window: $\widehat{y_i} = \frac{1}{w} \sum_{x=i-w}^{i-1} y_x \quad \text{with } i < w.$ Then for a series of (inference) samples, a client runs two types of inference; one using the ML model and another using the naive algorithm. The performance of each inference is calculated using the Root Mean Square Error (RMSE) metric, as the most relevant to time-series tasks [18]. The two RMSE values are compared with one another yielding a single indicator value, denoted as kpi that expresses the ML model's performance enhancement over the naive algorithm. If $RMSE_{ml}$ and $RMSE_{naive}$ mark the performance metrics of the ML models and the naive algorithm respectively, our indicator kpi of a client c in a round r becomes: $kpi^{c,r} = 100 \cdot (RMSE^{c,r}_{naive} - RMSE^{c,r}_{ml})/RMSE^{c,r}_{naive}$

DareFL functions as a synchronous FL framework, where training is divided into rounds of equal duration q. Client selection of K trainers and M testers occurs, similar to Vanilla FL (see Sec III-A). In the end of each round r, the server collects a $kpi^{c,r}$ value from each tester $c \in [1, M]$ and forms a kpi list, denoted as $\{kpi\}$. DareFL then employs a drift detection (DD) and a convergence detection (CD) algorithm (both of which require $\{kpi\}$ as input) to determine if the model needs further training or not *i.e.,* whether the next round will be active/idle, respectively. In an active round, both training and inference take place and the server receives the trained models from the trainers and the kpi (inference) values from the testers. In an idle round, training is halted; clients do not update their models and only inference and kpi collection are carried out.

The drift detection (DD) algorithm is based on the centralized Drift Detection Method (DDM) [5], which operates without accessing training data, aligning with the principles of FL. DDM detects drifts in classification problems, by evaluating the model's accuracy error rate. For a sequence of successful/unsuccessful classifications (0 and 1, respectively), assuming p_i and s_i is the error (unsuccessful) rate and standard deviation at the sequence's instance (sample) i and p_{min} and s_{min} the minimum recorded values, respectively, then the drift detection result of DDM is:

$$DDM(\beta_2, \beta_3) = \begin{cases} \text{warning}, & \text{if } p_i + s_i \geq \beta_2 \cdot s_{min} \\ \text{drift}, & \text{if } p_i + s_i \geq \beta_3 \cdot s_{min} \\ \text{no drift}, & \text{otherwise} \end{cases} \quad (1)$$

The values β_2 and β_3 denote DDM's sensitivity parameters, which are tuned via grid-search. To use DDM in distributed

settings, each client's kpi value is transformed to 0 or 1, by comparing to a tunable parameter β_1. β_1 is a threshold that expresses the minimum ML model's accuracy improvement over the baseline (naive) algorithm's accuracy, to account for a successful classification. Its value is tuned during the "warm-up" period's test-runs that provide initial statistics over the ML model's performance (see Sec. III-A). DDM is fed with the transformed $\{kpi\}$, denoted as (DDM list) $\{ddm\}$ so that a potential drift can be detected [5]. For the convergence detection (CD) algorithm, we calculate the central tendency $ckpi$ of each round's kpi list $\{kpi\}$, as the average value of all its elements. The server keeps track of all $ckpi$ values (in a per-round basis) in a $ckpi$ list, denoted as $\{ckpi\}$. Convergence detection is performed via the following rule: if $ckpi$ is not improved over the last β_4 rounds we assume that convergence is reached. Tuning is performed during the "warm-up" period, by monitoring each round's accuracy to observe its rate of change.

IV. Simulation Environment

In view of public pQoS data restrictions [9], we have fabricated two synthetic datasets[1] via a high fidelity network simulation that realistically models distinct drift scenarios in automotive environments. On top, we have built a distributed ML simulator[2] accompanied with resource consumption calculations, to utilize the datasets and evaluate our proposal. Both the datasets and the ML simulator are publicly available to promote reproducibility of results.

Generating pQoS datasets with concept drift: Our datasets represent a dynamic environment, where several client-vehicles are moving in an urban area. Each client runs a streaming cloud service constantly receiving data packets. This data can refer to various automotive applications *e.g.,* commands for ToD, video for infotainment services*, etc.* Network simulation is performed using Simu5G, a library that emulates a 5G cellular environment in OMNeT++ [19]. The simulator's radio parameters *e.g.,* channel properties, antenna settings*, etc.* are set according to the Macro-cell model proposed by International Telecommunication Union [20]. The map in our simulations comprises of an urban 600×600 m^2 area located in Drapetsona, a suburb of Piraeus, Greece. Inside this area, four 5G base-stations (gNodeBs) have been installed by the national network operator, enabling four 5G cells [21]. This area, divided into several blocks by the actual road network is integrated in our simulation by an OpenStreetMap (OSM) instance [22]. The total number of included vehicles is set to 25, according to vehicle density statistics in the corresponding country [23]. The road network's traffic is simulated by SUMO, a traffic simulation package [24] that creates a digitized version of the (real-world) OSM map and produces the route files for the vehicles. Route files are loaded in the Simu5G simulator, where a network-vehicular mobility co-simulation takes place. For each vehicle's route we assume SUMO's default parameters for urban environment *i.e.,* exponential speed model (with maximum speed restriction as defined by the OSM traffic rules) and the probability matrix at intersections for {lane keeping, turn left and right} as {0.5, 0.25 and 0.25}, respectively. The following information is collected for each vehicle using OMNeT++'s monitoring service: *timestamp*, *channel quality indicator*, *packet delay*, *measured signal to noise ratio (SNR)*, *3D client position and velocity (in Cartesian coordinates)*, *received SNR*, *radio link control throughput*, *serving cell*, *client throughput*. These features are sampled at 1 Hz and comprise the values of our synthetic time-series QoS dataset.

We have created two drift datasets that correspond to complementary cases of major long-term changes in the considered environment: 1) a network infrastructure-driven scenario (Sc1) and 2) a human behavior-driven scenario (Sc2). In Sc1 we assume that two out of four gNodeBs are switched off under a cost-reduction on/off policy. [25] For Sc2 we modify the users' mobility pattern; we assume that a "hotspot" *e.g.,* a metro station, is created in the lower-right edge of the map resulting in a traffic increase to that area [26]. This is achieved by increasing the probabilities of the routes leading to the "hotspot" in SUMO's route planning. All generated datasets have a total duration of 20 hrs (simulation time) and the respective drift event is introduced at t=10 hrs. Prior to drift, clients achieve an average throughput of 4.7$\pm$1.65 Mbps. This is decreased to 3.32$\pm$1.79 and 4.03$\pm$1.74 Mbps for Sc1 and Sc2, respectively. These changes will eventually be reflected on the model prediction's accuracy, as shown in Sec. V.

Distributed ML simulator: Our simulator implements the FL framework of Sec. III, along with the involved client-server communication and ML processing (consumption) costs. The distributed ML (training and inference) is facilitated via Pytorch, a Python deep ML library [27]. For the network resource consumption modelling, we assume a setup with a cloud server and several client-vehicles. Each vehicle, equipped with a 5G modem, communicates with the cloud server via the 5G cellular network. For the processing tasks (training, inference) the server is equipped with a Graphics Processing Units (GPU), while vehicles avail less powerful processing capabilities utilizing a common Central Processing Unit (CPU) [28]. Our simulator estimates the energy consumption imposed to the clients and the server due to processing and transmission, based on credible measurements in literature and device benchmarking. Namely, a typical 5G modem at uplink and downlink data rates of 20 and 100 Mbps consumes for transmission and reception 2.5 and 3.5 Watt, respectively [29]. A typical CPU trains an ML model relevant to pQoS data at a speed of 25 samples/sec, while a GPU reaches 900 samples/sec [30]. Those tasks require power of 200 Watt [31] and 225 Watt [30], respectively. Model aggregation however, has not been measured in literature thus we estimate the server's GPU computational speed and consumption based on the computationally-similar matrix-to-matrix multiplication at 10 models/sec and 100 Watt, respectively [32].

[1] https://zenodo.org/records/11084689

[2] https://github.com/gdrainakis/distributed_pqos

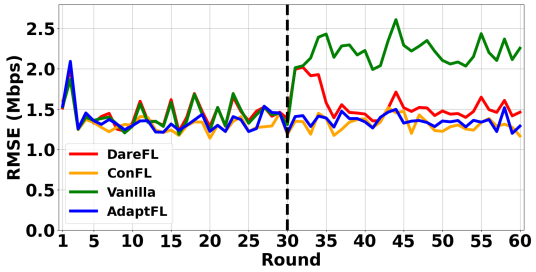

Fig. 2: Sc1 - RMSE comparison

V. Experimental Evaluation

Evaluation methodology: Our novel drift management FL algorithm (DareFL) is compared against existing solutions: 1) *Vanilla FL* (see Sec. III-A), 2) *Continuous FL* (ConFL - see Sec. II) and 3) *Adaptive FL (AdaptFL)* [8] under the two pQoS drift scenarios (Sc1, Sc2) described in Sec. IV. Vanilla and ConFL serve as baselines, while AdaptFL is selected as a representative SotA drift management FL solution. Each scenario is repeated 10 times for a total of 80 experiments (2 scenarios $\times$ 4 algorithms $\times$ 10 repetitions). QoS (throughput) prediction accuracy across time is evaluated via the RMSE metric (mean values across all clients). Resource consumption metrics include: 1) *Normalized communication cost i.e.,* total data exchanged between the server and the clients normalized to the ML model size, and 2) *Clients* and *Cloud energy cost i.e.,* the total energy consumed at each side for processing and transmission (see Sec. IV).

Throughout the experiments, round duration is set to q=1200 secs *i.e.,* a total of R=60 rounds for each scenario. Drift occurs at half-time (R=30) and lasts throughout the experiment. The total number of clients is set to 25, with K=5 trainers and M=20 testers per round, based on [33]. Trainers' data is split at a typical 80%-20% ratio [18]. Training is performed using an Long short-term memory (LSTM) model [2], consisting of: 11 input features (equal to the total features of each dataset) and 8 output features *i.e.,* a (throughput) prediction horizon of 8 sec (other automotive ML-based predictions show acceptable accuracy up to a 5 sec horizon [34]). Note that throughout Sec. V we show the results for a horizon of 6 sec for clarity, though the same principles apply to all other horizons up to 8 sec. For the LSTM we set: sliding window w=75, hidden size=50, Min-Max normalization, decay=10^{-5}, Rectified Linear Unit activation and Mean Square Error loss function, based on test-runs and related works on LSTM models [18]. Hyper-parameter tuning on our LSTM model via grid-search resulted in the following values: batch size=64, learning rate=10^{-5}, epochs=500. Leveraging on our test statistics during "warm-up" we set DareFL's parameters: β_1=0, β_4=5 rounds and default values β_2=2 and β_3=3. For fairness, AdaptFL's parameters are also tuned via grid search: β_1=β_2=β_3=0.7.

Evaluation results: DareFL's accuracy comparison against existing FL algorithms for Sc1 is shown in Fig. 2. Vanilla FL suffers from a sudden increase (51%) of the prediction error (RMSE) at the 31^{th} round, as a result of the inflicted drift. In Vanilla FL, the ML model is trained until convergence is reached, thus it cannot adapt to future drifts; as a result, the model's performance is degraded. ConFL on the other hand addresses drifts by constantly training (updating) the model, thus achieving maximum accuracy at all times. Compared to Vanilla, ConFL exhibits an average of 7% higher accuracy (in terms of RMSE) before and 40% after drift. Constant training however, results in the linear increase of the network bandwidth (see Fig. 3), energy costs for the clients (see Fig. 4) and the server (see Fig. 5). Compared to Vanilla FL, ConFL consumes 720% more bandwidth and 470%, 580% more energy in the clients and server-side at the end of the simulation, respectively. Unlike these solutions, DareFL leverages on its drift detection mechanism to ensure high accuracy, comparable to ConFL. Prior to drift, ConFL outperforms DareFL by an average of 8%. Upon drift occurrence, DareFL adapts in a handful of rounds (an average of 5.3 rounds across all experiments) and sustains similar accuracy to that of ConFL until the end of the experiment (see Fig. 2). Specifically, ConFL outperforms DareFL by an average of 11% after drift. Meanwhile, DareFL's energy and bandwidth footprint is kept relatively low (comparable to Vanilla FL), grace to its convergence detection mechanism that facilitates idle training rounds *i.e.,* saving on resources. As a result, DareFL achieves 76% lower communication costs (see Fig. 3) and 68% lower energy costs in the clients (see Fig. 4) and 74% on the server (see Fig. 5), compared to ConFL at the end of the simulation.

AdaptFL, which serves as a SotA drift management FL algorithm has similar behavior to ConFL, since it assumes constant training (see Fig. 2). Compared to DareFL, AdaptFL achieves a maximum accuracy enhancement of 9% throughout the experiment. However, it exhibits the highest resource consumption compared to all other algorithms. Specifically, its communication costs are equal to that of ConFL (see Fig. 3). Interestingly, AdaptFL consumes 200% more energy compared to DareFL and 100% more than ConFL in the clients and in the server side by the end of the simulation. This behavior occurs due to AdaptFL's drift management mechanism; at the server side it requires additional calculations for its detection process, similar to FedAvg's aggregation process (see Sec. II). The gradual adaptation of the learning rate also leads to "slower" learning in the clients-side *i.e.,* additional epochs, which increases the client energy footprint. These excessive energy costs however, have no effect on the increase of accuracy.

The findings of Sc1 are also validated in Sc2. Vanilla FL experiences a 43% accuracy drop at the 31^{th} round, which marks the effect of Sc2's drift on the model's performance. ConFL exhibits the best performance across time, due to constant training. ConFL outperforms Vanilla by 6% and 43% before and after drift, respectively. DareFL exhibits similar performance to that of Vanilla FL prior to drift however, it adapts after the the 31^{th} round, due to its drift detection mechanism. As such, it converges in similar accuracy levels as ConFL, within a margin of 5%. Compared to DareFL, AdaptFL achieves a 5% accuracy improvement before drift

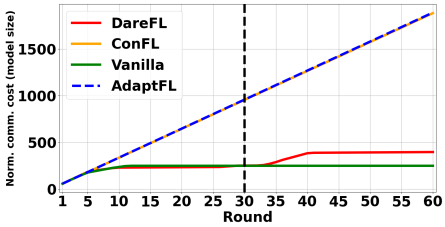

Fig. 3: Communication costs

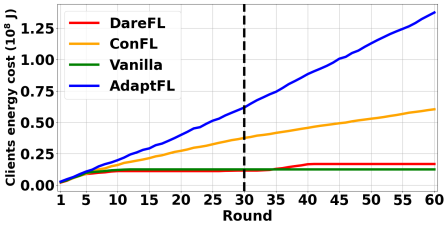

Fig. 4: Client energy costs

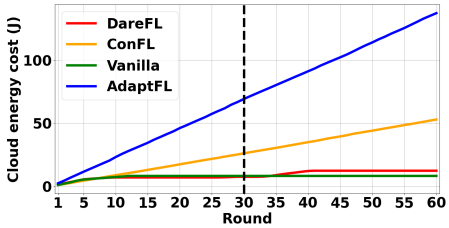

Fig. 5: Cloud energy costs

and 1% after drift. Similarly to Sc1, both AdaptFL and ConFL consume multiple times more energy and bandwidth to achieve these (minor) improvements over DareFL. The induced cost values for Sc2 are omitted, since the behavior is identical to that of Sc1 (see Fig. 3, 4 and 5).

VI. Conclusions

Drawing on the dynamicity of automotive environments that are subject to diverse drift causes, we have introduced DareFL, a novel concept drift management algorithm for predictive QoS, fully aligned to the FL principles *e.g.,* data privacy that operates at a significantly reduced resource consumption cost, compared to SotA. Using our developed (open-source) FL and network simulator we have evaluated DareFL under two complementary (infrastructure-/user- related) QoS drift scenarios. Our results suggest that DareFL achieves comparable prediction accuracy to existing solutions, whilst exhibiting an up to 70% energy and bandwidth efficiency. Interesting further research directions include the evaluation of the suggested framework in network testbeds or tuning our algorithm's parameters via ML techniques, such as reinforcement learning.

Acknowledgment

This paper is part of the 5G-IANA project, co-funded by the EU under the H2020 Research and Innovation Programme (grant agreement No 101016427).

References

[1] "Making 5G proactive and predictive for the automotive industry," Industry White Paper, 5GAA Automotive Association, 2020.
[2] H. Na *et al.*, "LSTM-based throughput prediction for lte networks," *ICT Express*, 2021.
[3] F. Casado *et al.*, "Concept drift detection and adaptation for federated and continual learning," *Multimedia Tools & Applications*, pp. 1–23, 2022.
[4] X. Yin *et al.*, "A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.
[5] F. Bayram *et al.*, "From concept drift to model degradation: An overview on performance-aware drift detectors," *Knowledge-Based Systems*, p. 108632, 2022.
[6] J. Lu *et al.*, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, pp. 2346–2363, 2018.
[7] D. M. Manias *et al.*, "Concept drift detection in federated networked systems," in *IEEE Global Communications Conference*, 2021, pp. 1–6.
[8] G. Canonaco *et al.*, "Adaptive federated learning in presence of concept drift," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–7.
[9] A. Palaios *et al.*, "Machine learning for QoS prediction in vehicular communication: Challenges and solution approaches," *IEEE Access*, 2023.
[10] E. Jothimurugesan *et al.*, "Federated learning under distributed concept drift," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2023, pp. 5834–5853.
[11] N. Harth *et al.*, "Local & federated learning at the network edge for efficient predictive analytics," *Future Generation Computer Systems*, vol. 134, pp. 107–122, 2022.
[12] C. B. Mawuli *et al.*, "Semi-supervised federated learning on evolving data streams," *Information Sciences*, p. 119235, 2023.
[13] L. Gondara and K. Wang, "Pubsub-ml: A model streaming alternative to federated learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2, pp. 464–479, 2023.
[14] F. E. Casado *et al.*, "Ensemble and continual federated learning for classification tasks," *Machine Learning*, pp. 1–41, 2023.
[15] Y. Chen *et al.*, "Asynchronous federated learning for sensor data with concept drift," in *2021 IEEE Intl. Conf. on Big Data*, pp. 4822–4831.
[16] B. Ganguly and V. Aggarwal, "Online federated learning via non-stationary detection and adaptation amidst concept drift," *IEEE/ACM Transactions on Networking*, 2023.
[17] Z. Charles *et al.*, "On large-cohort training for federated learning," *Advances in neural information processing systems*, vol. 34, pp. 20 461–20 475, 2021.
[18] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. ML Mastery, 2018.
[19] G. Nardini *et al.*, "Simu5G–an OMNet++ library for end-to-end performance evaluation of 5g networks," *IEEE Access*, vol. 8, 2020.
[20] M. Series, "Guidelines for evaluation of radio interface technologies for imt-2020," *Report ITU*, vol. 2512, p. 0, 2017.
[21] "Cellmapper," www.cellmapper.net, accessed: 2023-01-28.
[22] "Openstreetmap," www.openstreetmap.org, accessed: 2023-01-23.
[23] "Worldstats," www.nationsencyclopedia.com/WorldStats/WDI-transport-vehicles.html, accessed: 2023-01-28.
[24] M. Behrisch *et al.*, "SUMO–simulation of urban mobility: an overview," in *Procs of the Third Interl. Conf. on Advances in System Simulation (SIMUL)*. ThinkMind, 2011.
[25] N. Yu *et al.*, "Minimizing energy cost by dynamic switching on/off base stations in cellular networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 11, pp. 7457–7469, 2016.
[26] "Mobility report 2022," www.ericsson.com/4ae28d/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-november-2022.pdf, accessed: 2023-01-28.
[27] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.
[28] C. S. Evangeline *et al.*, "Safety and driver assistance in VANETs: an experimental approach for V2V," in *Int'l Conf. on Communication and Electronics Systems*, 2019, pp. 397–402.
[29] A. Narayanan *et al.*, "A variegated look at 5G in the wild: performance, power, and QoE implications," in *ACM SIGCOMM'21*, pp. 610–625.
[30] Y. Wang *et al.*, "Benchmarking the performance and energy efficiency of ai accelerators for ai training," in *20th IEEE/ACM Int'l Symposium on Cluster, Cloud and Internet Computing*, 2020, pp. 744–751.
[31] "CPU benchmarking," www.cpubenchmark.net/cpu.php?cpu=Intel+Xeon+Platinum+8168+%40+2.70GHz&id=3111, accessed: 2023-01-28.
[32] K. Fatahalian *et al.*, "Understanding the efficiency of GPU algorithms for matrix-matrix multiplication," in *ACM conf. on Graphics hardware*, 2004, pp. 133–137.
[33] A. Reisizadeh *et al.*, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *Int'l Conf. on Artificial Intelligence and Statistics*, 2020, pp. 2021–2031.
[34] S. Mozaffari *et al.*, "Deep learning-based vehicle behavior prediction for autonomous driving applications: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 33–47, 2020.