2020 International Symposium on Reliable Distributed Systems (SRDS)

Robust P2P Personalized Learning

Karim Boubouh
UM6P
Benguerir, Morocco
karim.boubouh@um6p.ma

Amine Boussetta
UM6P
Benguerir, Morocco
amine.boussetta@um6p.ma

Yahya Benkaouz
Mohammed V University
Rabat, Morocco
yahya.benkaouz@um5.ac.ma

Rachid Guerraoui
EPFL
Switzerland
rachid.guerraoui@epfl.ch

***Abstract*—Decentralized machine learning over peer-to-peer networks is very appealing for it enables to learn personalized models without sharing users data, nor relying on any central server. Peers can improve upon their locally trained model across a network graph of other peers with similar objectives. Whilst they offer an inherently scalable scheme with a very simple cost-efficient learning model, peer-to-peer networks are also fragile. In particular, they can be very easily disrupted by unfairness, free-riding, and adversarial behaviors.**

In this paper, we present CDPL (Contribution Driven P2P Learning), a novel Byzantine-resilient distributed algorithm to train personalized models across similar peers. We convey theoretically and empirically the effectiveness of CDPL in terms of speed of convergence as well as robustness to Byzantine behavior.

***Index Terms*—peer-to-peer machine learning, personalized models, Byzantine failures, robustness**

I. Introduction

The rapid adoption of connected personal devices (e.g., mobile phones, smartwatches, and connected objects) results in a tremendous amount of valuable data being continuously generated. Data have been usually transferred, stored and computed in the cloud (Fig. 1a). But privacy concerns call for more decentralized schemes where data would remain on user devices.

As devices are gaining prominent storage and computation capabilities, we indeed witness a shift towards a new scheme where data is kept on users devices. For example, Google advocated Federated Learning [1], a scheme that enables the training of models using data distributed across many clients. Clients collaboratively train a global model under the coordination of a central server (Fig. 1b) which orchestrates the training, including clients selection, updates collection, and weights aggregation. The server represents however a single point of failure (SPOF) and may become a bottleneck when the number of clients increases significantly [2].

In this paper, we consider a fully decentralized peer-to-peer system. Here, peers aim to learn a machine learning model based on their local data with no interaction with any central entity. Each peer has a local model that it trains according to its own learning objective. The learning is in this sense personalized. Peers collaborate with similar ones to enhance the accuracy of their pre-trained models (Fig. 1c). Two peers are considered similar if they have close data distributions. In practice, this pairwise similarity can be derived for instance, from peer profiles (e.g., interests, locations, subscriptions).

The efficiency of personalized learning depends heavily on the willingness of the collaborating peers to participate in the learning process effectively. We argue that it is crucial for a collaborative learning algorithm based on a peer-to-peer scheme to address the problem of adversarial behavior from some of the peers, besides traditional challenges related to distributed systems such as asynchrony and failures. This is the challenge that we take up in this paper. We assume the presence of Byzantine peers that aim to compromise the system: these include for example free-riders that do not contribute, i.e., peers that do not send anything to the system, as well as peers that send corrupt models.

We present CDPL (Contribution Driven P2P Learning), a novel fairness-aware incentive distributed algorithm to collaboratively learn personalized models in a peer to peer scheme. Our approach essentially relies on the evaluation of neighbor models before performing any update and continuously incentivizing peers to communicate good models. For pedagogical reasons, we first consider the static case, and then show how CDPL can work in a dynamic environment where peers can join and leave the network at any time.

Our CDPL algorithm can be viewed as a robust, i.e., Byzantine-resilient, extension of *Model Propagation* [3], which is not resilient even to a single misbehaving peer. We prove that CDPL converges, and does so at least as fast as Model Propagation, when there is no misbehaving peers. In other words, we show that in terms of convergence, our algorithm has no overhead with respect to the state of the art approach which is clearly not robust. Interestingly, CDPL can defend against an arbitrary number of Byzantine neighbors (even the extreme case where all the neighbors are Byzantine). For this, we establish an upper bound on the loss function at iteration T, for each honest peer, in terms of their initial loss value.

The rest of the paper is organized as follows. We first define the problem of interest in Section II. Then, we describe in Section III the basic Model Propagation approach and prove its vulnerability to misbehaving peers. Our CDPL algorithm is presented in details in Section IV for static and then dynamic networks. We analyze the convergence of CDPL and its robustness against Byzantine peers in Section V. Our experimental results are described in Section VI, where we evaluate CDPL extensively in different settings. Finally, we review related works in Section VII before concluding the paper in Section VIII.

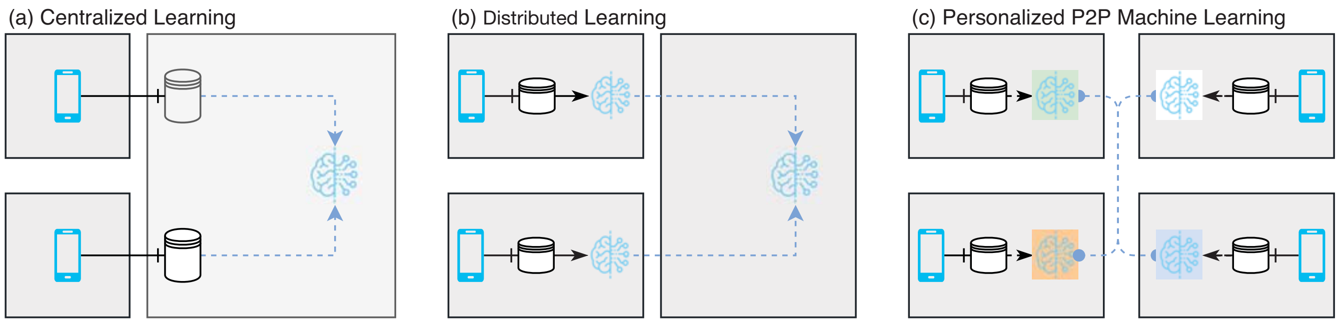


Fig. 1: Machine learning paradigms: Centralized, Federated and Personalized (P2P) machine learning.
(a) Centralized Learning: The server creates a global model based on the collected users data. **(b)** Distributed Learning (sometimes called Federated learning): Data is not shared with the server. Users receive a model from the server and use their data to improve the model. The updates are then communicated to the server to perform aggregation and output a new model. **(c)** Personalized Learning: Users train their models locally, then exchange their model parameters to enhance their own models without relying on any central server.

II. Problem Definition

In this paper, we consider a set of users aiming to train a machine learning model based on their local data while considering other users' models in the network graph. This graph represents a semantic overlay on top of the communication layer, defining a gateway between pairs of users sharing similar tasks and objectives. Each user maintains her own raw data and has her own learning objective. The idea is to take profit of similar users' data to enhance the accuracy of the built models. Each user should learn a model that involves her local data, her personal learning objective, and the models of similar users.

Formally speaking, users are organized in an undirected connected graph $G = (V, E)$. The vertex set V of size n represents peers. We denote by E the set of weighted edges between peers. The weights values reflect a notion of similarity. We denote by $W \in \mathbb{R}^{n\times n}$ the symmetric non-negative similarity matrix associated with G. Given two peers i and j, W_{ij} is the weight of the edge $(i, j) \in E$ representing the similarity between the objectives of i and j: W_{ij} tends to be large (resp. small) if i and j have similar objectives (resp. dissimilar). Furthermore, if i is not linked to j, the similarity is equal to zero (i.e. $W_{ij} = 0$). We set also $W_{ij} = 0$ if $i = j$. We assume that each peer i has access to its neighbors associated weights without having a global view of the network. We denote by $\mathcal{N}_i = \{j : W_{ij} > 0\}$ the set of neighbors of peer i, and $D_{ii} = \sum_{j=1}^{n} W_{ij}$.

Each peer i owns a set of $m_i \geq 0$ training examples $S_i = \{x_i^j, y_i^j\}_{j=1}^{m_i}$ drawn from the peer's local data distribution μ_i over a feature space $\mathcal{X}$ and a label space $\mathcal{Y}$ defining a personal supervised learning task. We consider i.i.d. training data (i.e. independent and identically distributed) following [3].

Given a convex loss function $\ell : \mathbb{R}^p \times \mathcal{X} \times \mathcal{Y}$, the goal of the peer i is to build a model $\theta_i \in \mathbb{R}^p$ with a small expected loss $\mathbb{E}_{(x_i,y_i)} \sim \mu_i \ell(\theta_i; x_i; y_i)$. Each peer i learns a local model by minimizing its local loss over S_i:

$$\theta_i^{loc} \in \underset{\theta\in\mathbb{R}^p}{\arg\min}\, \mathcal{L}_i(\theta) = \sum_{j=1}^{m_i} \ell(\theta, x_i^j, y_i^j) \tag{1}$$

Note that the size of the training samples may vary from one peer to another, depending on their activity. In fact, peers may have very few or no data at all (e.g., a device with small storage capacity, or new peers joining the network). Therefore, they may have very different models in terms of accuracy. Any exchange with a neighbor may weaken the peer's model or improve it depending on the quality of the received model. Peers have to prevent such behavior from affecting their models, especially if the neighbor is a malicious peer. We discuss below the model threat considered in our work.

A Byzantine peer is omniscient. It can collude with other Byzantine peers and has unlimited resources. Moreover, it can impersonate honest peers but cannot control their machines. Special cases of Byzantine attacks are: crash failures, communication failures, free riders who enjoy the system with no contribution by sending random vectors or re-sending the same received models and, attacks that introduce a backdoor, make the models diverge or converge to ineffective solutions, to name a few.

III. Peer-to-Peer Personalized Learning

Training models in isolation based only on local data can result in weak models if peers have very little data, or models that generalize very poorly. Our goal is to make all the network peers participate in a decentralized collaborative learning scheme where they can smooth their pre-trained models, to benefit from the shared information while maintaining accurate models with respect to their local data.

We formulate the problem (as in [3]) using graph regularization [4] to favor models that are smooth on the network graph. Denoting $\Theta = [\theta_1; \ldots; \theta_n] \in \mathbb{R}^{n\times p}$ as the vector of locally

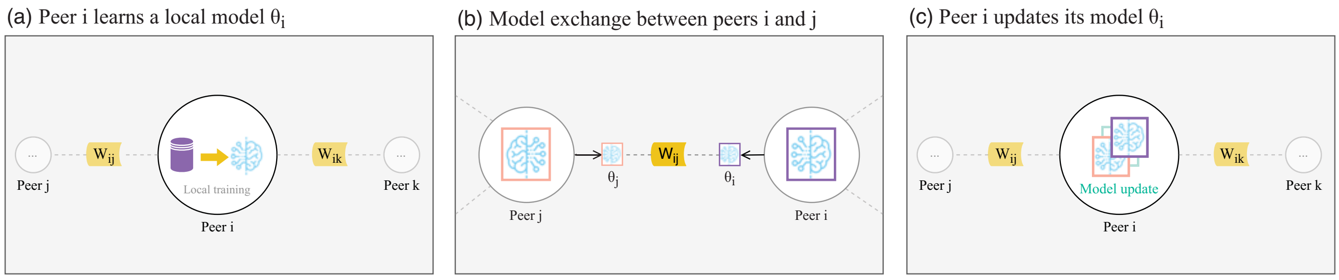


Fig. 2: Model Propagation steps.
(a) Initialization: peer i learns a local model based on its data. **(b)** Communication step: peers i and j exchange their model parameters asynchronously. **(c)** Update step: peer i updates its model based on the received models of its neighbors.

learned models θ_i for each peer i. The objective function we aim to minimize is as follows:

$$\mathcal{Q} = \frac{1}{2}\left(\sum_{i<j}^{n} W_{ij}\|\theta_i - \theta_j\|^2 + \mu \sum_{i=1}^{n} D_{ii}c_i\|\theta_i - \theta_i^{loc}\|^2\right) \quad (2)$$

Where μ is a trade-off parameter to control the effect of the network (first term) on the locally computed model, and c_i (in the second term) represents the confidence a peer has in its locally trained model. Large confidence values prevent peers from diverging too much from their original models, while low confidence values enable greater contribution from the network.

A. Model Propagation

We first present a decentralized algorithm to solve (2). Note that (2) has a closed form solution, but it requires the knowledge of the global network and the local models of all peers. In our settings, the peers only have information about their neighborhood. Regarding the time and communication models, we assume that each peer becomes active when its local clock ticks following a Poisson distribution. At each time $t \geq 0$ peers maintain (possibly outdated) knowledge about the models of their neighbors. For each peer i, we consider Matrix $\widetilde{\Theta}_i(t) \in \mathbb{R}^{n \times p}$, where $\widetilde{\Theta}_i^i(t) \in \mathbb{R}^p$ is the peer's model at time t, and for $j \neq i$, $\widetilde{\Theta}_j^i(t) \in \mathbb{R}^p$ is peer i's last knowledge of the model received by peer j.

1) Decentralized Algorithm: (Algorithm 1) Upon initialization, each peer i learns a local model θ_i^{loc} based only on its local training data (Fig. 2a). Once the local model is learned, each peer communicates with its neighbors to retrieve the information about the size of their data m_j. This latter is used to calculate the confidence value $c_i = \frac{m_i}{max_j m_j}$ (plus some small constant when $m_i = 0$) representing a normalization factor proportional to the size of the local data of the peer i.

At each timestamp t, peer i wakes up and performs the following two consecutive steps:

- **Communication step** (Fig. 2b): peer i selects randomly one neighbor $j \in \mathcal{N}_i$ and both peers exchange their models.

$$\widetilde{\Theta}_i^j(t+1) = \widetilde{\Theta}_j^j(t) \text{ and } \widetilde{\Theta}_j^i(t+1) = \widetilde{\Theta}_i^i(t)$$

- **Update step** (Fig. 2c): both peers update their models as follows. For $l \in \{i, j\}$:

$$\widetilde{\Theta}_l^l(t+1) = (\alpha + \bar{\alpha}c_l)^{-1} \left(\alpha \sum_{k \in \mathcal{N}_l} \frac{W_{lk}}{D_{ll}} \widetilde{\Theta}_l^k(t+1) + \bar{\alpha}c_l\theta_l^{loc}\right)$$

with $\alpha \in (0,1)$ such that $\mu = \frac{1-\alpha}{\alpha}$, and $\bar{\alpha} = 1 - \alpha$.

2) Convergence: The Model Propagation algorithm converges after t rounds [3] to a state where the peers reach an optimal model Θ^* within the network graph. Furthermore, since the algorithm is an asynchronous gossip based algorithm, the speed of convergence is much faster than synchronous algorithms in large decentralized peer-to-peer networks [5].

B. Model Propagation Weakness

Peer-to-peer learning can be of great help for peers to improve their models throughout the network graph. However, such a scheme induces a higher risk of Byzantine failure. In this direction, a confidence value based on data sizes presents many problems. First, the confidence value requires revealing the probably sensitive information about the size of peers local data. Second, it does not present any information related to the quality of the peer data. We formalize the non resilience of this scheme in Proposition 1.

Proposition 1 (Robustness): The Model Propagation algorithm 1 is not resilient to even a single Byzantine peer.

Proof: As defined in Section II, a Byzantine peer is omniscient and can impersonate other peers in the network. Thus, a single peer can make the whole network converge to a chosen model θ_b by proposing the vector $\hat{\theta_b}$ to every peer i:

$$\hat{\theta_b} = \frac{D_{bb}}{W_b}\left[\theta_b + \frac{\bar{\alpha}}{\alpha}c_i(\theta_b - \theta_i^{loc}) - \sum_{j \in \mathcal{N}_i/\{b\}} \frac{W_{ij}}{D_{ii}}\theta_j(t)\right]$$

Moreover, if there is a mechanism that filters models based on their norm, the Byzantine peer can select the data size variable

Algorithm 1 Model Propagation

```
Input : weighted connected graph $G$, trade-off parameter $\alpha$
Output : $\theta_j$ optimal model for every peer $j \in G$.
1: upon event $< mp.Init >$ do
2:     for $peer \in G$ do
3:         $\theta^{loc} \leftarrow learn(data)$;
4:         $\theta \leftarrow \theta^{loc}; \widetilde{\Theta} \leftarrow \{\perp\}^{k \times p}$; $\triangleright$ $k$ number of neighbors
5:         $c \leftarrow \frac{size(data)}{max(size(data), size(neighbors.data))}$;
6: end event

7: upon event $< p2p.Receive|neighbor, [\theta_j] >$ do
8:     $\widetilde{\Theta}[neighbor] \leftarrow \theta_j$;
9:     trigger $< p2p.Send|neighbor, [\theta] >$;
10:    $calculateUpdate()$;
11: end event

12: procedure exchange( )
13:    $neighbor \leftarrow randomNeighbor(neighbors)$;
14:    trigger $< p2p.Send|neighbor, [\theta] >$;

15: procedure calculateUpdate( )
16:    $\sigma = \{\perp\}^{k \times p}$;
17:    for $neighbor \in neighbors$ do
18:        $\sigma \leftarrow \sigma + \frac{W[neighbor]}{D} \widetilde{\Theta}[neighbor]$;
19:    $\theta \leftarrow (\alpha + \overline{\alpha} c)^{-1}(\alpha\sigma + \overline{\alpha} c \theta^{loc})$;
```

$m_b > \max_{j \in (1, \cdots, n)} m_j$ to be large enough in order to control the term $\theta_b - \theta_i^{loc}$ and consequently, the norm of $\hat{\theta}_b$. ■

Remark: Even in a weaker threat model where impersonation is not allowed, the algorithm is still not Byzantine resilient and the number of Byzantine peers necessary depends on the graph topology. If the graph is complete, then a single Byzantine peer can damage the whole system in the first update by sending the vector $\hat{\theta}_b$ to all peers. If the graph is not fully connected, then a single Byzantine peer can damage the whole network in less than T rounds, where T is the number of edges in the shortest path linking the Byzantine peer and the farthest peer in the network. If the graph is disconnected, then it suffices to have a Byzantine peer in each cluster to achieve the same damage.

IV. Contribution Driven P2P Learning

In the algorithm presented in the previous section, the confidence value for each peer is derived from the data size of its neighbors. In our alternative, we suggest using the quality of the received models instead of the size of the data. In other words, peers will derive a contribution factor from each received model, then decide whether the suggested exchange will contribute to their models. While this approach increases the time complexity of the algorithm as it requires evaluating the received models, it constitutes a serious filter against Byzantine peers and allows peers to defend against extreme scenarios where all the neighbors are Byzantine.

A. Contribution Factor

The contribution factor cf aims to ensure an up to date quantification of the quality of the peers models, hence the quality of their contribution in the collaboration. Instead of relying on the size of the data received from peer j, the contribution factor cf_i^j represents how the model of peer j can potentially contribute to peer i's model. By convention, we set $cf_i^j = 1$ if $i = j$.

Let ζ_i be a random variable representing the set of random samples drawn from the dataset of peer i. Let $\mathcal{L}_i(\theta_j, \zeta_i)$ be the loss using the model θ_j on the test set ζ_i. Let $p_{ij} = \frac{b_i}{\mathcal{L}_i(\theta_j, \zeta_i)}$ be the corresponding accuracy of θ_j on the same test set, where b_i is a generic constant satisfying the negative correlation between the loss and the accuracy.

As shown in CDPL (Algorithm 2), when a given peer i receives a model $\widetilde{\Theta_i^j}(t+1)$ from a peer j, during the *communication step*, the peer i evaluates the accuracy of the received model.

The contribution factor of $\widetilde{\Theta_i^j}(t+1)$ is computed based on p_{ii} and p_{ij} as follows:

$$c_i^{j'} = \frac{p_{ij}}{p_{ii}}$$

Peer i then updates the contribution factor of peer j by integrating its historical contribution as follows:

$$c_i^j(t+1) = \gamma * c_i^j(t) + (1 - \gamma) * c_i^{j'}$$

where $\gamma \in [0, 1]$ is a trade-off parameter to control the effect of the peer j's transactional history on the currently calculated contribution factor.

To avoid potential accuracy decrease in peer i's model, we set a threshold cf_{th} to determine whether the update should be accepted or ignored. Finally, if the update is accepted, we proceed to the aggregation as described in the *update step* using:

$$c_i = \frac{1}{max(c_i^j)} \; for \; j \in \mathcal{N}_i \cup \{i\}$$

B. Fairness Control

The contribution factor drives the exchange scheme by limiting the effect of weak models, while still responding back to enable peers with fewer data points to take advantage of the collaboration. However, free-riders can join the collaborative learning without performing any computation either to save resources or to keep the added value of their data for themselves. Instead, they may send random models in each exchange step, and wait for the other peers to send back their models. Hence, we suggest an incentive approach to establish a notion of fairness between peers and encourage them to participate with better models.

The literature of peer-to-peer systems suggests many approaches to tackle these issues: (1) reciprocal-based [6] where

Algorithm 2 CDPL: Contribution Driven P2P Learning

Input : weighted connected graph G, trade-off parameter α, contribution factor threshold cf_{th}, fairness_control ϵ, random sample Δ from test-set.
Output : θ_j optimal model for every peer $j \in G$.

```
1:  upon event $< mp.Init >$ do
2:      for $peer \in G$ do
3:          $\theta^{loc} \leftarrow learn(data)$;
4:          $\theta \leftarrow \theta^{loc}; \widetilde{\Theta} \leftarrow \{\perp\}^{k\times p}$;
5:          $banned, cf, p \leftarrow \{\perp\}$
6:  end event

7:  procedure exchange( )
8:      $neighbor \leftarrow randomNeighbor(neighbors)$;
9:      trigger $< p2p.Send|neighbor, [\theta] >$;

10: upon event $< p2p.Receive|neighbor, [\theta_j] >$ do
11:     $accepted,\ p_j = evaluateModel(\theta_j)$
12:     if $accepted$ then
13:         $\widetilde{\Theta}[neighbor] \leftarrow \theta_j$;
14:         $calculateUpdate()$;
15:     else
16:         if $fairnessControl(neighbor, p_j)$ then
17:             trigger $< p2p.Send|neighbor, [\theta] >$;
18: end event

19: procedure evaluateModel($neighbor, \theta_j$)
20:     $p_i \leftarrow accuracy(\theta, \Delta)$
21:     $p_j \leftarrow accuracy(\theta_j, \Delta)$
22:     $cf[neighbor] = \gamma * cf[neighbor] + (1-\gamma) * \frac{p_i}{p_j}$
23:     if $cf[neighbor] \geq cf_{th}$ then
24:         return true, $p_j$
25:     else
26:         return false, $p_j$

27: procedure fairnessControl($neighbor, p_j$)
28:     if not $neighbor \in banned$ then
29:         $diff = \frac{p_j - p[neighbor]}{p[neighbor]}$
30:         if $diff < \epsilon$ then
31:             $banned \leftarrow banned \cup \{neighbor\}$
32:         else
33:             return true
34:     return false

35: procedure calculateUpdate( )
36:     $\sigma = \{\perp\}^{k\times p}$;
37:     $c \leftarrow \frac{1}{max(1, max(cf[neighbors]))}$;
38:     for $neighbor \in neighbors$ do
39:         $\sigma \leftarrow \sigma + \frac{W[neighbor]}{D}\widetilde{\Theta}[neighbor]$;
40:     $\theta \leftarrow (\alpha + \overline{\alpha}c)^{-1}(\alpha\sigma + \overline{\alpha}c\theta^{loc})$;
```

peers give preference to peers investing their resources in the collaboration; (2) reputation-based [7], where the peers get benefits depending on the amount of resources they invest; (3) punishment-based [8] where free-riders requests are eventually ignored and forced to disconnect from the network. Inspired from these approaches, our idea is to keep track of the quality of models received from neighbors. If a peer keeps sending models of decreasing quality, it will be considered as a misbehaving peer and will be banned, thus prevented from any future exchanges during the training cycle. Meanwhile, weaker models can be tolerated during the first communication round since peers may have few data points.

We define the scheme of detecting misbehaving peer as follows: each peer i keeps note of the previous value p_{ij} of peer j, denoted as $p_{ij}(t-1)$ and initialized with value 0. Once a peer i receives a new model from peer j, peer i compares the accuracy $p_{ij}(t)$ of the received model with the previous accuracy $p_{ij}(t-1)$ by computing the percentage difference as follows:

$$\frac{p_{ij}(t) - p_{ij}(t-1)}{p_{ij}(t-1)}$$

If the computed percentage is smaller than a threshold ϵ representing the maximum decrease in j' model accuracy tolerated by i, the peer j is banned from collaboration. The value of ϵ can considerably affect the collaboration experience, as large values may lead to honest peers being banned, while small values can intensify the effect of the Byzantine attacks.

C. Dynamic CDPL

So far, we considered the network graph as a given parameter of our algorithm. We assumed a static undirected weighted graph where the edges represent the actual similarities between peers. However, the assumption of having such a relevant graph is impractical in many situations. In addition to the challenges related to building such a graph, a realistic graph would be dynamic as peers may join or leave the network at any time. Furthermore, peers profiles and preferences may change over time, necessitating the adaptation of the graph to these changes. Conceiving an optimal graph under these requirements is challenging as peers have no global view of the network. It was suggested in [9] to learn the collaboration graph and the personalized models by optimizing a joint objective function (weights and model parameters). The problem becomes bi-convex and solvable by alternating optimization of the two sub-problems. Specifically, they used a variant of coordinate descent to optimize the weights of the collaboration graph. However, such a scheme is still not robust against misbehaving peers.

We suggest DYNAMIC CDPL, an extended version of CDPL algorithm to jointly learn a contribution-based network graph where peers only retain up to date neighbors that contribute positively to their models. Peers with weak contributions or suspicious behaviors are eventually unpaired from peers neighbors list.

1) Initialization: To bootstrap the collaboration, we start with an oracle graph as input if such a graph exists. In this scenario, peers already have a set of relevant neighbors with whom they can start the collaboration. Otherwise, we start with a random network graph incorporating random neighbors (typically a small number) drawn from a peer sampling service [10] without having any information about the global network. We assume that the peer sampling service guarantees a uniform random sampling of peers even under Byzantine attacks following [11], [12].

2) Dynamic graph updates: During the collaboration scheme, peers update their set of neighbors depending on how beneficial their contributions are. For a peer i participating in the collaboration, two undesirable situations can happen: (1) a misbehaving peer j is detected and banned; or (2) a proposed update from j is ignored. In the first situation, peer i samples a random peer k from the peer sampling service to replace j. In the second scenario, i samples one peer k from the peer sampling service and evaluates its contribution factor cf_k. If $cf_k > cf_j$, then the peer i sends an *unfriend* message to peer j and replace it with k.

V. CDPL ANALYSIS

In this section, we discuss and prove the properties of our CDPL algorithm in terms of convergence and robustness.

A. Convergence

In honest environments, the original algorithm is proved to converge to an optimal solution. However, the speed of convergence depends heavily on the quality of neighbors' models. In contrast, CDPL allows peers to limit the effect of weak models resulting in faster convergence speed.

Proposition 2 (Convergence): CDPL (Algorithm 2) converges, at least as fast as Model Propagation (Algorithm 1) to an optimal solution:

$$\Theta^* = \underset{\Theta \in \mathbb{R}^{n \times p}}{\arg\min}\, \mathcal{Q}(\Theta)$$

Proof: The closed form solution to the minimization problem $\mathcal{Q}$ defined in [3] is:

$$\Theta^* = \bar{\alpha}(I - \bar{\alpha}(I - C) - \alpha P)^{-1} C \Theta^{loc}$$

Note that, it is necessary for the Matrix $(I - \bar{\alpha}(I-C) - \alpha P)$ to have a spectral radius less than 1, in order to be invertible. This is true in particular when P is a stochastic matrix and $(I - C)$ have diagonal entries less than 1. In our algorithm, $C_{ii} = c_i = \frac{1}{max(c_i^j)}\ for\ j \in \mathcal{N}_i \cup \{i\}$ and $P = D^{-1}W$ is a stochastic matrix by construction, so both conditions are satisfied. Convergence is then proved using the same arguments as in [3].

We show now that our algorithm reaches the optimal solution as least as fast as *Model Propagation*. We assume in this section that $\forall i \in (1, \cdots, n), \mathcal{L}_i$ is a convex function. Let a_{1i} (resp. a_{2i}) be generic constants such that $\mathcal{L}_i(\frac{1}{\alpha + \bar{\alpha} c_i}\theta) = a_i \mathcal{L}_i(\theta)$ (resp.$\mathcal{L}_i(c_i\theta) = a_{2i}\mathcal{L}_i(\theta)$). We have then:

$$\mathcal{L}_i(\theta_i(t+2)) =$$
$$\mathcal{L}_i\left(\frac{1}{\alpha + \bar{\alpha}c_i}\left(\alpha \sum_{j \in \mathcal{N}_i} \frac{W_{ij}}{D_{ii}} \theta_j(t+1) + \bar{\alpha} c_i \theta_i^{loc}\right)\right)$$
$$= a_i \mathcal{L}_i\left(\left(\alpha \sum_{j \in \mathcal{N}_i} \frac{W_{ij}}{D_{ii}} \theta_j(t+1) + \bar{\alpha} c_i \theta_i^{loc}\right)\right)$$

Since $\sum_{j \in \mathcal{N}_i} \frac{W_{ij}}{D_{ii}} = 1$ and $\alpha + \bar{\alpha} = 1$, we obtain by applying Jensen Inequality:

$$\mathcal{L}_i(\theta_i(t+2)) \leq \tag{3}$$
$$a_{1i}\left(\alpha \sum_{j \in \mathcal{N}_i} \frac{W_{ij}}{D_{ii}} \mathcal{L}_i(\theta_j(t+1)) + \bar{\alpha} a_{2i} \mathcal{L}_i(\theta_i^{loc})\right) \tag{4}$$

Since loss and accuracy are negatively correlated, we set b_i to be a generic constant such that $\mathcal{L}_i(\theta_j) = \frac{b_i}{p_{ij}}$. We split the reasoning into two scenarios:

- $\forall j \in \mathcal{N}_i, \quad \mathcal{L}_i(\theta_j) \leq \frac{b_i}{cf_{th}}$
- $\exists j \in \mathcal{N}_i, \quad \mathcal{L}_i(\theta_j) > \frac{b_i}{cf_{th}}$

Clearly, the first scenario results in an update for both algorithms. Thus, similar convergence properties are expected for both algorithms. The interesting scenario is the second one. In this case, some neighbors are proposing a model with a bad quality loss, which will be accounted for in the weighted sum of Algorithm 1 update but rejected in our algorithm (CDPL). Therefore, the loss function in CDPL can increase with a bounded magnitude, as opposed to Model Propagation where the increase is not controlled, which can be interpreted as a fast convergence. ∎

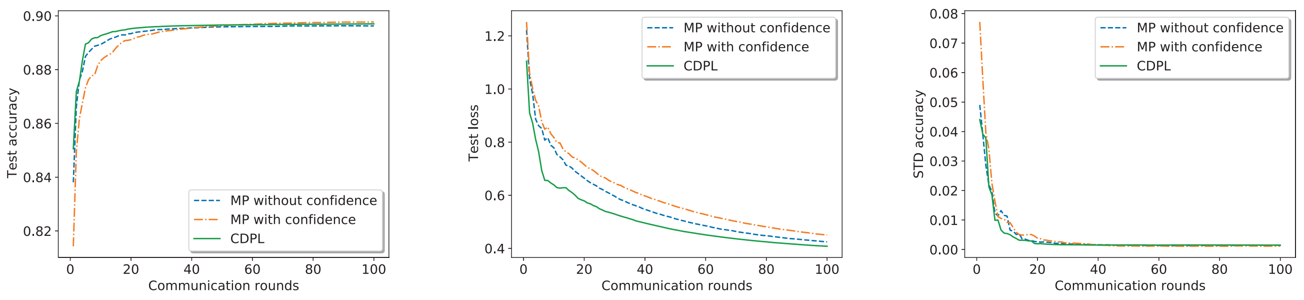
B. Robustness

The presence of Byzantine adversaries is a realistic and natural assumption, especially in open environments such as peer-to-peer networks. Here, we discuss the resilience of CDPL against Byzantine peers.

Given a node i with n_i neighbors, an extreme Byzantine attack is when all the neighbors of i are Byzantine peers colluding to deviate peer i's model. We prove that CDPL guarantees an upper bound impact of the Byzantine peers on the model of peer i in terms of accuracy, even in the worst scenario where the number of Byzantine peers $f = n_i$.

Proposition 3 (upper bound): If CDPL (Algorithm 2) is run for T iterations under a Byzantine attack, then $\exists M > 0, \quad \mathcal{L}_i(\theta_i(T)) \leq M\mathcal{L}_i(\theta_i^{loc})$ (i.e. the accuracy decrease is bounded).

Proof: This is a direct result of Proposition 2. Let us consider the extreme case where a peer i have only Byzantine peers in its neighborhood. The contribution factor is a strong filter against peers with low accuracy models. Therefore, a



(a) Test accuracy (b) Test loss (c) Standard deviation accuracy

Fig. 3: Convergence of CDPL vs. Model Propagation with/without confidence.
(a) Accuracy: the use of the *contribution factor* results in faster convergence compared to confidence (less than 20 rounds).
(b) Test loss: the *contribution factor* performs much better in minimizing the loss, as it avoids aggregating weak models.
(c) Standard deviation in peers accuracy: in all settings, peers perform well in smoothing their models.

Byzantine worker is forced to provide models above the threshold to be considered in the update. The worst attack is then to send models with the lowest acceptable accuracy. Specifically:

$$c_i^{j'} > cf_{th} \implies \frac{p_{ij}}{p_{ii}} > cf_{th} \implies \mathcal{L}_i(\theta_j) < \frac{1}{cf_{th}}\mathcal{L}_i(\theta_i)$$

Using the same assumptions on the individual loss functions in Proposition 2, we obtain using Equation 3:

$$\mathcal{L}_i(\theta_i(t+1)) \leq$$

$$a_{1i}\left(\alpha \sum_{j\in\mathcal{N}_i}\frac{W_{ij}}{D_{ii}}\mathcal{L}_i(\theta_j(t)) + \bar{\alpha}a_{2i}\mathcal{L}_i(\theta_i^{loc})\right)$$

$$\leq a_{1i}\left(\frac{\alpha}{cf_{th}}\sum_{j\in\mathcal{N}_i}\frac{W_{ij}}{D_{ii}}\right)\mathcal{L}_i\left(\theta_i(t) + a_{1i}\bar{\alpha}a_{2i}\mathcal{L}_i(\theta_i^{loc})\right)$$

Rewriting the last inequality leads to:

$$\mathcal{L}_i(\theta_i(T)) \leq$$

$$\left[a_{1i}\frac{\alpha}{cf_{th}}\right]^T \mathcal{L}_i(\theta_i(0)) + \left(\sum_{k=1}^{T}\left[a_{1i}\frac{\alpha}{cf_{th}}\right]^{k-1}\right)\bar{\alpha}a_{1i}a_{2i}\mathcal{L}_i(\theta_i^{loc})$$

Since $\theta_i(0) = \theta_i^{loc}$, the last inequality reduces to:

$$\mathcal{L}_i(\theta_i(T)) \leq$$

$$\mathcal{L}_i(\theta_i^{loc})\left(\left[a_{1i}\frac{\alpha}{cf_{th}}\right]^T + \left(\sum_{k=1}^{T}\left[a_{1i}\frac{\alpha}{cf_{th}}\right]^{k-1}\right)\bar{\alpha}a_{1i}a_{2i}\right)$$

If we further choose $cf_{th} > a_{1i}\alpha$, then:

$$\mathcal{L}_i(\theta_i(\infty)) \leq \mathcal{L}_i(\theta_i^{loc})\left(\frac{\bar{\alpha}a_{1i}a_{2i}}{1 - a_{1i}\frac{\alpha}{cf_{th}}}\right)$$

■

Remark: Note that the upper bound presented in Proposition 3 does not hold for Model Propagation (Algorithm 1). In fact, local models can vanish in the update equation when a Byzantine worker sends a large value as its data size, leaving only a weighted average of received models, which can be as well Byzantine.

In dynamic environments, DYNAMIC CDPL can be very appealing, as it allows the discovery of potentially better peers resulting in a fast convergence even if peers profiles and preferences are continuously changing.

Proposition 4: DYNAMIC CDPL algorithm converges, at least as fast as Model Propagation (Algorithm 1).

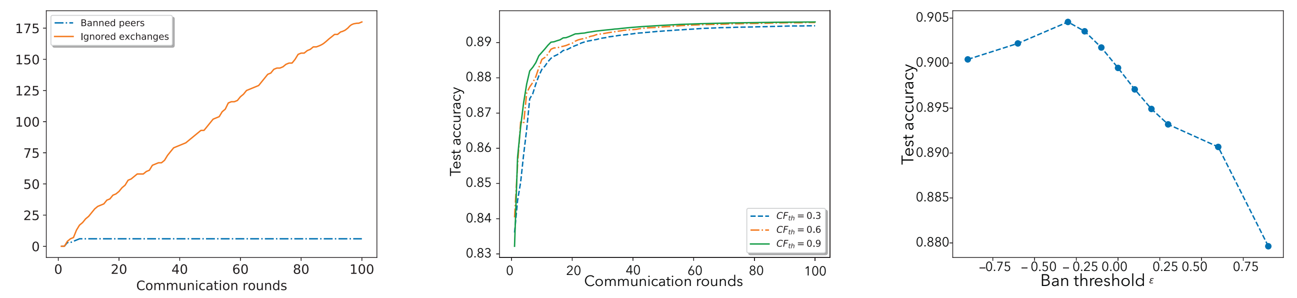
Proof sketch: A run of DYNAMIC CDPL algorithm can be divided into three phases: first, the algorithm tries to select the best neighbors for each peer by evaluating their contribution factor and setting the weights of non similar neighbors to zero (using the ban rule described in IV-B). In the second phase, the algorithm adjusts the weights of the neighbors depending on their contribution factor. In the last phase, the weights either stop updating or change with a small magnitude. Since the Matrix W and the coefficient c_i are still satisfying the conditions stated in Proposition 2 proof, we only need to show that the weight matrix W eventually reaches a stable state where every peer has identified a set of very similar neighbors, based on the contribution factor. The problem reduces then to the first setup where the matrix is static which convergence is proven in Proposition 2. ■

VI. EXPERIMENTAL RESULTS

To demonstrate the advantages of the CDPL approach compared to *Model Propagation* in terms of speed of convergence and robustness, we have implemented a peer-to-peer simulation network to enable end-to-end communication between peers of the network graph. In the following, we present the implementation environment, the considered metrics, and the obtained results. Moreover, our implementation provides experimental results of *Model Propagation* compared to the theoretical results presented in [3].

A. Implementation Environment

For this study, we have considered image classification as a learning task performed by 50 peers. The dataset we have



(a) Ignored model vs. banned peers.

(b) Effect of cf_{th} on convergence speed

(c) Effect of ϵ on convergence speed

Fig. 4: Banning peers and ignoring updates in CDPL.

considered is the well-known MNIST dataset [13]. MNIST consists of 10 categories, including digits ranging from 0 to 9, with a total of 70,000 data samples (60,000 for training and 10,000 for testing). Each peer of the network graph is assigned a non-overlapping i.i.d. sample from the MNIST training set drawn uniformly at random from a normal distribution with mean $\mu = m/n$ (m is the size of the MNIST training set) and width $\sigma = \frac{1}{2}\mu$. Peers were assigned the same test set. The network graph was built using a random network generator that randomly samples the number of neighbors for each peer from a uniform distribution. For all the experiments (unless mentioned otherwise), we set the trade-off parameter γ to 0.2 allowing 80% change in the contribution factor derived from the proposed model. The remaining 20% accounts for the history of the peer's contributions embedded in the previous contribution factor. We fix the contribution factor threshold cf_{th} to 0.8 allowing aggregation of even slightly less accurate models. The fairness control threshold ϵ was set to -0.1 to allow a 10% decrease in peers accuracy during the collaboration. The parameter α was set to 0.5.

All experiments were conducted on a computer running Ubuntu 18.04 LTS, with Intel(R) Xeon(R) W-2123 CPU 3.60GHz and 32.0G of RAM. The source code was written in Python language, and makes use of scikit-learn 0.21.3. The source code of our implementation is available on: https://github.com/karimboubouh/peernet.

The focus of our experiments was precisely to study the impact of the following parameters and configurations on the algorithms outcome: (1) Confidence factor; (2) Byzantine peers; (3) Data unbalancedness; (4) Network density. To study the impact of these parameters, we have considered the following metrics: accuracy, loss, and the number of communication rounds to study the convergence behavior. Precision, recall, and F1 score have been used to analyze the accuracy of Byzantine peers detection. In the following subsections, we present the main findings of our study.

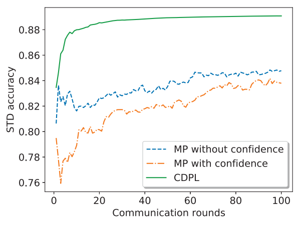
B. Convergence Rate

In our first set of experiments, we study the speed of convergence of the algorithm in three different configurations: (1) not considering any confidence; (2) using confidence based on peers data sizes; (3) using the contribution factor (CDPL). In Figure 3a, we plot the test accuracy as a function of the number of communication rounds performed to reach convergence. We observe that CDPL starts at a higher accuracy and converges faster (in less than 20 rounds). However, all configurations converge to an optimal solution after a sufficient number of rounds. Figure 3b shows a significant drop in loss when using the contribution factor compared to the other configurations. This behavior is expected as the CDPL algorithm selects only models with small losses. Thus, the models will not experience a significant variation in accuracy because of the aggregation of bad and good models. Figure 3c indeed shows that the standard deviation of the models accuracy during the collaboration task is minimum using CDPL. Furthermore, Figure 3c shows that the algorithms in all configurations perform well with higher smoothness degrees.

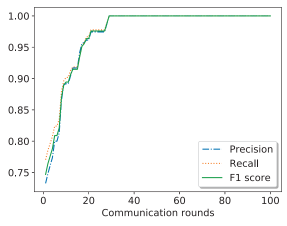
C. Effect of the CDPL Parameters

In this subsection, we compare different values of the contribution factor thresholds cf_{th} and fairness control thresholds ϵ. These parameters control the interaction with the network as they can allow for more tolerated or restricted collaboration. During this collaboration, each peer can ignore a proposed update if it does not satisfy the requirement of the peer defined by the thresholds cf_{th} and ϵ. Figure 4a shows the number of updates being ignored during the collaboration, as well as the number of peers banned after exchanging models with decreased accuracy. We observe that peers with decreasing accuracy are banned in the first few iterations, while peers keep ignoring updates if they hold no added value. This behavior shows that our algorithm can detect and filter suspicious peers at an early stage.

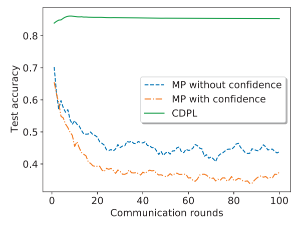
Figure 4b demonstrates that higher values of cf_{th} can result in faster convergence and better accuracy. However, imposing large cf_{th} values may prevent peers from taking advantage of important updates offered by their neighbors. The fairness control threshold ϵ on the other hand, plays a crucial role in preventing Byzantine peers from affecting peers models. However, as reported in Figure 4c, enforcing a large increase in accuracy from neighbors may significantly reduce the overall



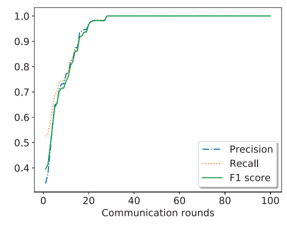
(a) Accuracy (5 Byzantine peers)



(b) Byzantine peers detection

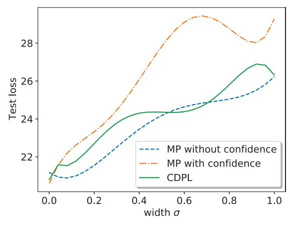


(c) Accuracy (25 Byzantine peers)

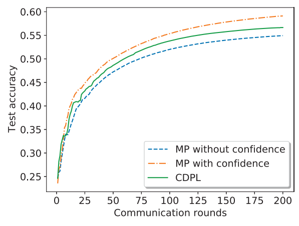


(d) Byzantine peers detection

Fig. 5: The effect of Byzantine peers attacks on peers models compared to other configurations.



(a) Loss vs. data unbalancedness



(b) Accuracy with non-iid data

Fig. 6: Effect of data unbalancedness and non-iid data.

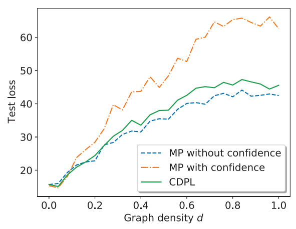


Fig. 7: Graph density vs. total test loss.

accuracy. This is due to the fact that peers will keep banning any peer not ensuring an ϵ increase of the model accuracy.

D. *Effect of Byzantine Peers*

We consider that every Byzantine peer aims to compromise the models of its neighbors by communicating random models in each communication step of the algorithms. In figures 5a and 5c, we study the behavior of honest peers in the presence of 5 and 25 Byzantine peers out of 50 peers. We measure the accuracy of honest peers models during the exchange to study the effect of attacks on those peers. Figures 5a and 5c show the great advantage of using CDPL in order to protect against Byzantine attacks. In the case of Model Propagation, the accuracy drops drastically from the first few iterations. At the same time, CDPL does not experience any decrease in accuracy as it ignores any model not reaching the contribution factor threshold. From figures 5b and 5d, we see that it takes less than 20 rounds for all peers to detect and filter any Byzantine peer aiming to disturb the collaboration task. The full detection happens after few communication rounds, mainly because peers may exchange models with honest peers first before encountering a Byzantine peer, which takes two rounds to detect it.

E. *Effect of Data Unbalancedness*

We perform several experiments to study the effect of the training data on convergence. For each peer, data is sampled uniformly at random from a normal distribution without replacement with mean $\mu = m/n$ (m is the size of the training set) while varying the width $\sigma \in [0, 1]$. When $\sigma = 0$, all the peers will have the same number of training examples. Large values of σ allow more variation in the datasets. Figure 6a shows how the loss evolves while increasing the value of σ.

In Figure 6b, we study the behavior of the algorithm out of our i.i.d. data assumption. We observe that in the case of peers holding non-iid data, even the CDPL failed to drive the peers to generalize their models and reach a good accuracy, even after running the algorithm for 200 rounds. The average accuracy of the models did not cross the value of 60%.

F. *Effect of Network Density*

To study the effect of network density, we consider an Erdős–Rényi graph generator with a density parameter $d \in [0, 1]$ to control the density of the generated graph. Small values of d results in a sparse graph. While increasing the values of d, the network graph gets denser, and peers receive more neighbors. Figure 7 presents the behavior of the algorithm in terms of test loss with respect to the graph sparsity. We conclude from the experiments that when the network graph is sparse, the algorithm has a negligible effect on the system, as each peer communicates with few peers. With the increase of graph density, the algorithm smooths the model of each peer with more neighbors (e.i., smoothing over the cluster resulting in personalized models). Finally, at a particular density factor, the graph gets almost fully connected and the models converge to a global model.

VII. Related Work

Research in decentralized machine learning has mainly focused on distributed optimization techniques [14]–[16]. The goal has mainly been to learn a single global model by minimizing the average of the local losses of agents participating in the learning task [17]–[19].

We consider in this paper a completely decentralized setting where agents have each their own local model and personalized learning objectives, which they seek to improve by communicating in a peer-to-peer manner with other agents in the network. Several such approaches have recently been presented [3], [9], [20], [21]. The approach of [3] used the idea of alternative direction method of multipliers [22]. The work of [20] tackled the privacy issue by proposing a differentially private algorithm that enables peers to learn personalized models under strong privacy requirements efficiently. To learn personalized models, the approach of [9] suggested to jointly learn the network graph with the models to achieve lower communication costs. The approach of [21] suggested a distributed algorithm for learning personalized models that requires no hyperparameters tuning.

None of these approaches however tackled the issues of Byzantine peers. Clearly, in a realistic peer-to-peer environment, Byzantine peers may join the collaboration as free-riders to take advantage of the exchanged updates without investing any resources. In the worst cases, peers can be subject to Byzantine attacks from single or colluding peers aiming to corrupt their models. Many recent Byzantine-resilient ML algorithms [23]–[26] have been proposed to defend against a proportion of malicious workers in the training phase. However, these techniques are not applicable to our case. Essentially, we deal with personalized models, one per peer, instead of one single distributed model as assumed in that line of research. In fact, our paper is the first to account for Byzantine peers in the context of personalized peer-to-peer learning. Also, most of those papers actually assume a centralized architecture where a parameter server (possibly replicated) supervises many workers in order to optimize a function using a variant of Gradient Descent: we do not assume any parameter server. Finally, the filtering of bad workers is done using gradients. In our setup, we work directly with the parameter vectors instead of the gradients which makes our protocol independent of the optimization scheme.

VIII. Conclusion

This paper proposes CDPL (Contribution Driven P2P Learning), a fairness-aware incentive distributed algorithm, to collaboratively learn personalized models in a peer-to-peer network. The originality of CDPL is its resilience to Byzantine peers. Our algorithm relies on the evaluation of neighbors models before performing any updates, which make it able to defend against an arbitrary number of Byzantine peers. Peers are continuously incentivized to communicate better models. We show theoretically and empirically that CDPL is robust and converges faster compared to the classical Model Propagation approach that is not Byzantine-resilient. As future work, we plan to explore the tightness of the bounds presented in this work as well as enhance the time complexity of our algorithm.

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics (AISTATS)*, 2017.

[2] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *International Conference on Neural Information Processing Systems (NIPS)*, 2017.

[3] P. Vanhaesebrouck, A. Bellet, and M. Tommasi, "Decentralized collaborative learning of personalized models over networks," in *Artificial Intelligence and Statistics (AISTATS)*, 2017.

[4] P. S. Dhillon, S. Sellamanickam, and S. S. Keerthi, "Semi-supervised multi-task learning of structured prediction models for web information extraction," in *Conference on Information and Knowledge Management (CIKM)*, 2011.

[5] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, and A. Scaglione, "Gossip algorithms for distributed signal processing," *Proceedings of the IEEE*, vol. 98, pp. 1847–1864, 2010.

[6] K. G. Anagnostakis and M. B. Greenwald, "Exchange-based incentive mechanisms for peer-to-peer file sharing," *International Conference on Distributed Computing Systems (ICDCS)*, pp. 524–533, 2004.

[7] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *International conference on World Wide Web (WWW)*, 2003, pp. 640–651.

[8] M. Karakaya, I. Korpeoglu, and Ö. Ulusoy, "Counteracting free riding in peer-to-peer networks," *Comput. Networks*, vol. 52, pp. 675–694, 2008.

[9] V. Zantedeschi, A. Bellet, and M. Tommasi, "Fully decentralized joint learning of personalized models and collaboration graphs," in *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[10] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Transactions on Computer Systems (TOCS)*, vol. 25, p. 8, 2007.

[11] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, "Brahms: Byzantine resilient random membership sampling," *Comput. Networks*, vol. 53, pp. 2340–2359, 2009.

[12] G. P. Jesi, D. Hales, and M. van Steen, "Identifying malicious peers before it's too late: A decentralized secure peer sampling service," *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pp. 237–246, 2007.

[13] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[14] A. Nedic and A. E. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control (TACON)*, vol. 54, pp. 48–61, 2009.

[15] S. S. Ram, A. Nedic, and V. V. Veeravalli, "Distributed stochastic subgradient projection algorithms for convex optimization," *Journal of Optimization Theory and Applications (JOTA)*, pp. 516–545, 2010.

[16] E. Wei and A. E. Ozdaglar, "Distributed alternating direction method of multipliers," *IEEE Conference on Decision and Control (CDC)*, pp. 5445–5450, 2012.

[17] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Transactions on Automatic Control (TACON)*, pp. 592–606, 2012.

[18] Z. Jiang, A. Balu, C. Hegde, and S. Sarkar, "Collaborative deep learning in fixed topology networks," in *International Conference on Neural Information Processing Systems (NIPS)*, 2017, pp. 5906–5916.

[19] H. Tang, X. Lian, M. Yan, C. Zhang, and J. Liu, "D2: Decentralized training over decentralized data," in *International Conference on Machine Learning (ICML)*, 2018.

[20] A. Bellet, R. Guerraoui, M. Taziki, and M. Tommasi, "Personalized and private peer-to-peer machine learning," in *Artificial Intelligence and Statistics (AISTATS)*, 2018.

[21] I. Almeida and J. M. F. Xavier, "Djam: Distributed jacobi asynchronous method for learning personal models," *IEEE Signal Processing Letters (SPL)*, vol. 25, pp. 1389–1392, 2018.

[22] S. P. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends in Machine Learning*, vol. 3, pp. 1–122, 2011.

[23] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Byzantine-tolerant machine learning," *ArXiv*, vol. 1703.02757, 2017.

[24] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault, "The hidden vulnerability of distributed learning in byzantium," in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018, pp. 3521–3530.

[25] G. Damaskinos, E. M. E. Mhamdi, R. Guerraoui, R. Patra, and M. Taziki, "Asynchronous byzantine machine learning (the case of sgd)," in *International Conference on Machine Learning (ICML)*, 2018.

[26] D. Yin, Y. Chen, R. Kannan, and P. Bartlett, "Byzantine-robust distributed learning: Towards optimal statistical rates," in *International Conference on Machine Learning (ICML)*, 2018, pp. 5650–5659.