A Privacy-preserved D2D Caching Scheme Underpinned by Blockchain-enabled Federated Learning

Runze Cheng†, Yao Sun†, Yijing Liu‡, Le Xia†, Sanshan Sun§, Muhammad Ali Imran†
† James Watt School of Engineering, University of Glasgow, Glasgow, U.K.
‡ National Key Lab. on Communications, University of Electronic Science and Technology of China, Chengdu, China
§ College of Physics and Electronic Engineering, Sichuan Normal University, Chengdu, China
Email: Yao.Sun@glasgow.ac.uk

***Abstract*—Cache-enabled device-to-device (D2D) communication has been widely deemed as a promising approach to tackle the unprecedented growth of wireless traffic demands. Recently, tremendous efforts have been put into designing an efficient caching policy to provide users better quality of service. However, public concerns of data privacy still remain in D2D cache sharing networks, which thus arises an urgent need for a privacy-preserved caching scheme. In this study, we propose a double-layer blockchain-based federated learning (DBFL) scheme with the aim of minimizing the download latency for all users in a privacy-preserving manner. Specifically, in the sublayer, the devices within the same coverage area run a federated learning (FL) to train the caching scheme model for each area separately without exchange of local data. The model parameters for each area are recorded in sublayer chains with Raft consensus mechanism. Meanwhile, in the main layer, a mainchain based on practical Byzantine fault tolerance (PBFT) mechanism is used to resist faults and attacks, thus securing the reliability of FL updates. Only the reliable area models authorized by the mainchain are utilized to update the global model in the main layer. Numerical results show the convergence, as well as the gain of download latency of the proposed DBFL caching scheme when compared with several traditional schemes.**

***Index Terms*—D2D Caching, Federated Learning, Blockchain**

I. Introduction

To cope with the unprecedented growth of wireless traffic, caching popular contents at network edges has shown great potential. This pre-download method allows users to fetch contents from the cache server thus to significantly relieve the network pressure, and reduce the latency of users [1]. Meanwhile, D2D communication technology has been invented to enable multiple direct transmissions between pairs of nearby devices in cellular networks [2], hence the spectrum efficiency can be dramatically improved. Borrowing the D2D communication technology, a promising and attractive trend is to allow user equipment (UE) to play an active role as caching servers, and thus to establish a caching-enabled D2D network [3]-[5].

Considering the limited caching storage of D2D devices as well as the time-varying user preferences, designing an efficient caching scheme to minimize the content fetching delay is an essential yet challenging issue [6], [7]. Recently, tremendous works have been devoted into utilizing learning-based algorithms to derive an optimal caching scheme in D2D networks. Specifically, the authors in [8] designed a caching scheme based on value-based reinforcement learning (RL). The authors in [9] applied policy-based RL to cache scheme design with the assumption that the central server can accurately predict user preferences. In [3], [10], deep RL (DRL) is used to analyze the preference similarity between users thus to distribute content. All the above works aim to improve the caching hit-rate under different scenarios while paying little attention to user privacy during information exchanges in the learning process.

However, user local data (including preferences, download records, cached contents, etc.) are normally sensitive and private [11]. Thus, UEs would not like to share the local data in practice. In light of the increasing privacy concerns, federated learning is considered as a promising solution to design a D2D caching scheme. Basically, FL is an emerging decentralized learning approach that requires exchanges of learning models rather than raw data, which is beneficial for protecting data privacy. Besides privacy, another key motivation is that FL can achieve satisfied learning performance by sharing training models even under the case with insufficient local training data [12].

In spite of these superiorities, one critical challenge faced is the reliability of model updates in FL process, especially in untrusted D2D networks with faults and malicious attacks. Specially, once some D2D nodes do not send the updates or send fault updates, the performance of the global model in FL may be heavily degraded, leading to a poor cache hit. To tackle this challenge, blockchain, which is a distributed ledger technology providing immutable and persistent data records [13], can be expected to serve as an information verification and storage tool in FL. In addition, using blockchain brings a by-product to overcome the issue that users in D2D networks may show a low willingness to transmit data for others and be indolent in model training [14]. Introducing the blockchain-based incentive scheme provides rewards for the users who play an active and positive role in this system. It is promising to attract more users to actively join the caching model training and D2D content sharing.

In this study, we propose a privacy-preserved D2D caching scheme by exploiting FL underpinned by a double-layer blockchain architecture. Simulations demonstrate both the convergence and the performance gain of the proposed DBFL caching scheme compared with the traditional schemes. The main contributions of our work are listed:

1) We formulate the D2D caching problem as a multi-agent Markov decision process (MDP) problem, and propose a novel caching scheme named DBFL.
2) In DBFL, we develop an FL with the double-stage cluster-based framework to establish a reliable and privacy-friendly learning scheme. This FL method allows users to train models in a distributed way without raw data exchange.
3) We exploit a double-layer blockchain architecture to underpin the above FL. Specifically, multi-sublayer chains based on the Raft consensus mechanism are used to store area models, and stimulate users to participate in D2D caching. Meanwhile, a main blockchain with the PBFT consensus mechanism verifies area models to resist Byzantine faults, ensuring the accuracy of the global model.

The remainder of this paper is organized as follows. The system model is described in Section II, followed by presenting the problem formulation in Section III. Then, we propose the DBFL caching scheme in IV. We show simulations in Section V. Finally, the conclusion is presented in Section VI.

II. System Model

A. D2D Network

We consider a D2D network that consists of a central base station (BS) and multiple UEs with the capability of caching and D2D communication. Each UE can provide cached contents via D2D links as an independent server [8]. Moreover, we assume that the BS has an ample cache capacity, and all the required contents can be obtained from the BS [15].

Let $\mathcal{U} = \{1, 2, \ldots, \alpha\}$ be the set of all the considered UEs in the coverage of the BS. For a specific UE u, it requests content from the item library $\mathcal{F} = \{1, 2, \ldots, \kappa\}$ with the content size $\mathcal{S_F} = \{s_1, s_2, \ldots, s_\kappa\}$. Moreover, we denote $\mathcal{C_U} = \{c_1, c_2, \ldots, c_\alpha\}$ as the set of storage capacity of UEs. The fetching process is first broadcasting the request to the nearby UEs when UE u requires item f. Subsequently, the UEs that locally store the required item f send a reply, and the UE u is connected to the nearest item holder to obtain the required item. If no UE replies to the request, the BS should take over the request.

B. Content Caching Models

In this work, we consider three possible caching models as shown in Fig. 1 including self-caching, D2D-caching, and BS-caching, which are illustrated as follows.

1) Self-caching: UE u first checks whether the item has been cached in the local storage. The request will be satisfied immediately if the local cache hits.

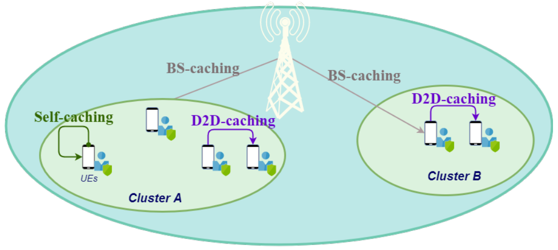

Fig. 1. Three different caching models in the communication community, 1) self-caching, 2) D2D-caching, and 3) BS-caching.

2) D2D-caching: If the required content items are not cached at the local device, UE u will search the nearby devices within the radius of R_{D2D} trying to get the required item.
3) BS-caching: BS-caching will be the last method for UE u to obtain the desired content when neither self-caching nor D2D-caching hits the requirement. The cellular BS will receive the request of UE u and transmit items from the local storage center.

C. Transmission Latency

There are two transmission scenarios: 1. D2D transmission, 2. BS-UE transmission. We assume that the D2D links do not interfere with each other [8]. For D2D transmission, let $SNR_{u,v} = P_D \cdot G_{u,v}/\sigma_N^2$ represent signal-to-noise ratio (SNR), where P_D denotes the transmission power of user device, $G_{u,v}$ is the channel gain of D2D transmission, and σ_N^2 is the Gaussian white noise power. Furthermore, for the channel gain of D2D link, we have $G_{u,v} = k_D \cdot d_{u,v}^{-\varepsilon_D}$, where k_D and ε_D denote the path loss constant and exponent of the D2D link respectively. The transmission rate of a D2D pair UE u and v is $\omega_{u,v} = B_D \cdot \log_2(1 + SNR_{u,v})$, where B_D is the available bandwidth of D2D link [8].

For BS-UE transmission, we use a similar way to calculate the SNR $SNR_{u,0}$ and transmission rate $\omega_{u,0}$ of UE u. We assume that both the wireless bandwidth and transmit power are evenly allocated among the multiple serving devices. In addition, $\tau_{uv}^f = s_f/\omega_D$ is the transmission latency for u to fetch item from UE v while $\tau_{u0}^f = s_f/\omega_B$ is the transmission latency for u getting item from the BS.

III. Problem Formulation

In this section, the D2D caching scheme design is formulated as a multi-agent MDP with the aim to minimize the download latency for all users. The following is the illustration of the state, action, and reward function.

A. Action

Let $\mathcal{A}_u = \{a_{u,1}, a_{u,2}, \ldots, a_{u,f}\}$ be the action space of UE u, where $a_{u,f} \in \{0, 1\}$ denotes the action for UE u cache content item f. If UE u caches the item f, $a_{u,f} = 1$, otherwise $a_{u,f} = 0$. Note that the total size of cached content items cannot exceed the storage capacity c_u of UE u, *i.e.*, $\sum_{i=1}^{\kappa} s_i \cdot a_{u,i} \leq c_u$.

B. State

The state of the environment for UE u at time t is denoted as $\mathcal{S}_u^t = (\mathcal{P}_u^t, \mathcal{Q}_u^t)$, where $\mathcal{P}_u^t = \{p_{u,1}^t, \ldots, p_{u,f}^t\}$ is the estimated local popularity in the coverage range of u. $\mathcal{Q}_u^t = \{q_{u,1}^t, \ldots, q_{u,f}^t\}$ is the local hit-rate of file f. Specially, $p_{u,f}^t = \frac{n_{u,f}^t}{\sum_{i=1}^{\kappa} n_{u,i}^t}$, where $n_{u,f}^t$ is the number of requests for f. $q_{u,f}^t = \frac{m_{u,f}^t}{n_{ue}}$, where n_{ue} is the number of UEs among the coverage of UE u, and $m_{u,f}^t$ is the number of UEs fetching content f from UE u.

C. Reward and Return

Every UE receives the same reward, where the reward is calculated based on the total transmission latency reduction. The transmission latency of UE u for fetching content item f at time t is given by

$$\Gamma_{u,f}^t = \left(1 - a_{u,f}^t\right) \cdot \left[Z_{u,f}^t + \tau_{u0}^f \cdot \prod_{\mu \in \mathcal{N}_u} \left(1 - a_{\mu,f}^t\right) \right], \quad (1)$$

where $\mathcal{N}_u$ denotes the neighbors of u, μ is the UE with the μ-th lowest latency for sending the item f to UE u, and $Z_{u,f}^t$ denotes the lowest latency of UE u to fetch the content item f from neighbors.

The lowest latency of fetching content f can be calculated as

$$Z_{u,f}^t = \sum_{\mu=1}^{|\mathcal{N}_u|} \left(\tau_{u,\mu}^f \cdot \prod_{\nu=1}^{\mu-1} \left(1 - a_{\nu,f}^t\right) a_{\mu,f}^t \right), \quad (2)$$

where $\prod_{\nu=1}^{\mu-1} \left(1 - a_{\nu,f}^t\right) a_{\mu,f}^t$ is an indicator function with value of 0 to denote that no UE can fetch content faster than μ. If $|\mu|$ is larger than the total number of neighbors, it means that no UE in $\mathcal{N}_u$ stores the item f. In addition, when UE u fetches item f, the transmission latency reduction is as

$$H_{u,f}^t = \tau_{u0}^f - \Gamma_{u,f}^t. \quad (3)$$

Therefore, the total reduction of the transmission latency for UE u at time t is given as

$$\begin{aligned} Y_u^t \left(a_{u,1}^t, \ldots, a_{u,f}^t\right) &= \sum_{i=1}^{\kappa} d_{u,i}^t \cdot H_{u,i}^t \\ &+ \sum_{i=1}^{\kappa} \sum_{\mu \in \mathcal{N}_u} a_{\mu,i}^t \cdot d_{u,i}^t \cdot H_{\mu,i}^t, \end{aligned} \quad (4)$$

where $d_{u,f}^t \in \{0, 1\}$ is a binary decision variable that indicates whether UE u requests for the item f at time t, and $a_{\mu,f}^t = 1$ denotes that the neighbor μ stores the item f.

The total reward obtained by UEs is given as

$$\begin{aligned} R_t\left(\boldsymbol{a_1^t}, \ldots, \boldsymbol{a_u^t}\right) &= \frac{1}{|\mathcal{N}_u|} \sum_{j=1}^{\alpha} \sum_{i=1}^{\kappa} d_{j,i}^t \cdot Y_{j,i}^t \\ &+ \frac{1}{|\mathcal{N}_u|} \sum_{j=1}^{\alpha} \sum_{i=1}^{\kappa} \sum_{\mu \in \mathcal{N}_L} a_{\mu,i}^t \cdot d_{u,i}^t \cdot Y_{\mu,i}^t. \end{aligned} \quad (5)$$

To ensure the highest overall reward, some UEs may sacrifice their own storage space to meet the needs of others. Hence, in some cases, the content transmission latency of partial UEs could be relatively high. To avoid this problem, we add a constraint that the maximum average latency for each UE to fetch an item cannot exceed the threshold y_{max}.

The discounted return is composed of short term reward and long term reward, which is denoted as

$$\hat{R}_t = \sum_{i=0}^{m-1} \gamma^i R_{t+i} + \gamma^m \hat{R}_{t+m}, \quad (6)$$

where γ denotes the discount rate and $0 < \gamma < 1$, the time $t > 0$.

IV. DBFL Caching Scheme Design

In this section, we design a caching scheme DBFL by exploiting a blockchain-enabled FL framework as shown in Fig. 2. This is a cluster-based and privacy-preserved scheme that allows UEs to learn cooperatively without exchanging users' private information. Besides, it resists Byzantine and omission faults by utilizing a double-layer blockchain. We illustrate the three steps of DBFL: 1) task publishment and local model training; 2) sublayer blockchains and area models update; 3) main blockchain and global model update.

A. Task Publishment and Local Model Training

Initially, the cache sharing and model training tasks are broadcasted by the publisher. UEs respond to the publisher and send their relevant information (storage, battery, CPU/GPU model and etc.) for verification. Subsequently, they download the initial states and models for training. The UEs update the states after all content requests from UEs are satisfied in the decision slot. Then, the average latency reduction of users can be calculated, which is used to update the reward. At time t, the state space, action space, reward, and the next state can be packed as a transition (s_t, a_t, r_t, s_{t+1}). Besides, we use a larger buffer with a capacity n_b to store these transitions. The transition management using the first-in, first-out (FIFO) method. Furthermore, we use the Stochastic Gradient Descent (SGD) scheme to randomly sample a transition from the buffer to compute the temporal difference (TD) error then calculate the stochastic gradient. The approximate value function parameterized by the weight $\boldsymbol{\omega}$ is shown as

$$Q(s_t, a_t; \boldsymbol{\omega}) \approx r_t + \gamma Q(s_{t+1}, a_{t+1}; \boldsymbol{\omega}). \quad (7)$$

In a double deep Q network (Double DQN), the best action is given by

$$a^* = \underset{x}{\operatorname{argmax}}\, Q(s_{t+1}, a; \boldsymbol{\omega}). \quad (8)$$

The action evaluation in Double DQN using TD target network is as

$$y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a^*; \boldsymbol{\omega}^-), \quad (9)$$

where $\boldsymbol{\omega}^-$ is the weight parameter of the TD target network [16]. The TD target network has the same structure as that of the deep Q network (DQN), but with different weight

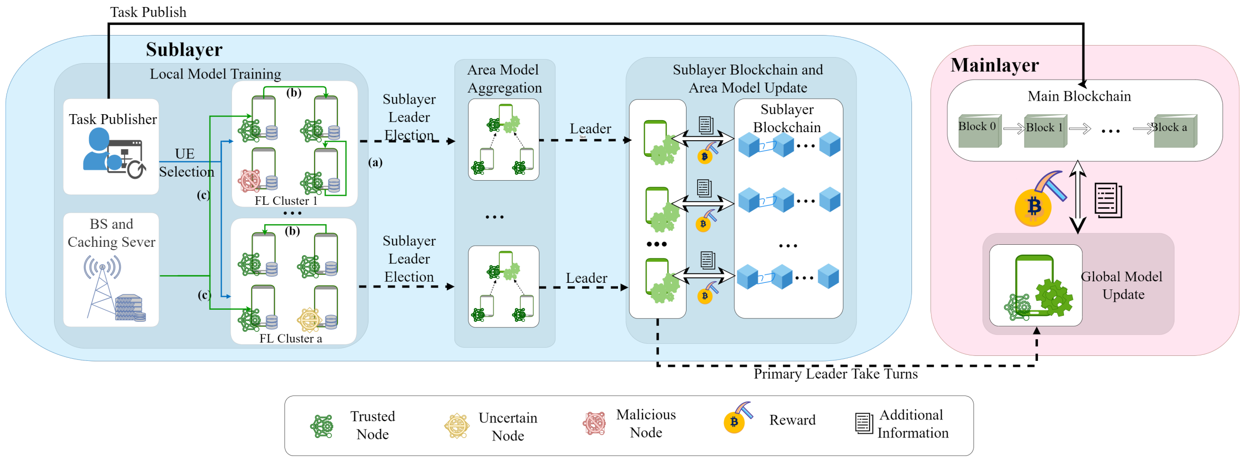

Fig. 2. Blockchain and federated learning framework of the D2D caching system.

parameters. Moreover, γ is used to balance the immediate reward and future rewards. We use Double DQN to shorten the difference between $\boldsymbol{\omega}^-$ and $\boldsymbol{\omega}$ by minimizing the loss function, given by

$$L_t(\boldsymbol{\omega}) = \frac{1}{2}\left[Q(s_t, a_t; \boldsymbol{\omega}) - y_t\right]^2 = \frac{\delta_t^2}{2}, \tag{10}$$

where δ_t is the target error.

We draw samples randomly from the pool of stored samples. Then, we use the loss to update $\boldsymbol{\omega}$ as per the following rule

$$\begin{aligned}\boldsymbol{\omega_{u,t+1}} &= \boldsymbol{\omega}_{u,t} - \alpha \cdot \boldsymbol{g_{u,t}} = \boldsymbol{\omega_{u,t}} - \alpha \cdot \frac{\partial\left(\delta_t^2/2\right)}{\partial \boldsymbol{\omega}} \\ &= \boldsymbol{\omega_{u,t}} - \alpha \cdot \delta_t \cdot \frac{\partial Q(s_t, a_t; \boldsymbol{\omega})}{\partial \boldsymbol{\omega}},\end{aligned} \tag{11}$$

where g_t is the stochastic gradient and α denotes the learning rate.

B. Sublayer Blockchains and Area Models Update

As mentioned, various omission faults (agents send false data accidentally) and Byzantine faults (malicious agents send false and fake data, hard to detect) may exist in practical cache-enabled D2D networks. Thus, in this study, blockchain is applied to this cache sharing network to secure a reliable FL performance.

In D2D networks, limited by the low computing capability of user devices, the proof-based consensus mechanisms with high consumption (such as Proof of Work and Proof of Stake) cannot be directly applied to the subchain. Besides, due to the numerous nodes, the complexity of using PBFT or Byzantine Fault Tolerance (BFT) in the sublayer is extremely high. In this problem, we only consider the omission fault in the sublayer chains. Hence, consortium blockchains are exploited, where every device needs to get a permit to participate. Considering the high device density, the consensus mechanism of these consortium blockchains should have high scalability. Moreover, it must be implement-friendly subject to the light capability of D2D nodes. Fortunately, these requirements are consistent with the characteristics of Raft.

There are three categories of nodes in Raft consensus, saying leader, candidate, and follower [17]. Each cluster can select only one trusted UE with the best performance as the leader, and the rest of UEs then become followers. Candidates are the immediate state of followers trying to become the new leader node.

The area model update in the sublayer chains under Raft consensus is composed of two stages.

1) Leader election stage: the leader UE in each cluster sends heartbeats to all followers. Once followers can not receive the heartbeats from the leader, a leader election stage will be initiated. These followers transfer the candidate state and first vote for themselves. Then, the candidates will send a set of data to request other UEs to vote for them. If a candidate wins the majority of votes, it will become the new leader and regularly send heartbeats to all UEs to maintain its rule.
2) FL model update stage: the leader UE receives local models and other additional information (training time, data size and etc.) from followers. To resist the attacks or faults, the leader will check whether the training speed matches to the training time and data size, thus to determine the authenticity of a local model. Only those verified local models can be used to update the area model. The update function is shown as

$$\boldsymbol{\omega_a^{t+1}} = \boldsymbol{\omega_a^t} - \alpha \cdot \sum_{u \in U_c} \frac{s_{u,n}}{S_a} \cdot \boldsymbol{g_{t,u}}, \tag{12}$$

where S_a is the total size of training data samples from n UEs in area a, $S_a = \sum_{u \in U_a} s_{u,n}$.

The updated area model and the relevant information will be packed into a new block in the sublayer blockchain. These datasets are transparent that can be checked and verified.

C. Main Blockchain and Global Model Update

Only leader nodes of sublayer areas have the right to involve in the global model update. Therefore, the computational complexity of using the PBFT in the mainchain can be decreased due to that a few nodes are allowed to participate in the consensus process. It is capable of utilizing PBFT to detect Byzantine faults and preventing the global model from being severely affected by malicious nodes.

In PBFT, the nodes are divided into two categories: 1) primary node and 2) secondary node. There is one sublayer leader taking turns to become the primary node in each round, while the rest of sublayer leaders are secondary nodes in PBFT. The ultimate goal is that these primary and secondary nodes reach a consensus on a principle of the minority obeying the majority. Once the primary node received a request for area model verification, it first checks the loss between the previous global model and every area model, and then evaluates the cache prediction performance of these models. Only well-performed area models with high hit-rate are used to update the global model. Then, the primary node broadcasts the verification request to all the secondary nodes. Unlike Raft, secondary UEs in PBFT are able to challenge the reliability and rationality of the primary UE. Specifically, secondary nodes can check the cache hit-rate of the global model, and determine whether the model update is valid and reasonable. The request is successfully served when there are $f+1$ nodes that reply with the same result. Here should note that f is the maximum number of malicious nodes that PBFT can be tolerated, $f = (n_a - 1)/3$, where n_a is the total number of nodes [18].

The global model update is a large time-scale task, i.e., the global model update once after several updates on area models. The update function of the global model is given by

$$\boldsymbol{\omega}_g^{r+1} = \sum_{a \in A} \frac{1}{n_a} \cdot \boldsymbol{\omega}_{\boldsymbol{a}}^{\boldsymbol{r}}, \tag{13}$$

where $\boldsymbol{\omega}_g^{r+1}$ is the weights of the updated global model at round r.

V. Simulations and Discussions

In this section, we conduct simulations to compare the performance of our proposed DBFL scheme with the three following caching schemes:

1) Zipf random: UEs randomly cache content with an assumption that the content popularity obeys Zipf distribution.
2) DRL-based: This caching scheme is value function-based, it enables multi-agent cooperatively learning.
3) FL-based: This scheme is similar to the DBFL caching scheme, except for the involvement of blockchain. In other words, each agent in this scheme runs a DRL to train the local model without using blockchain to assist the global model aggregation.

Both omission and Byzantine faults are considered. The omission fault is a UE that accidentally stops the model update process for more than 10 rounds, while the Byzantine fault is to upload random updates maliciously.

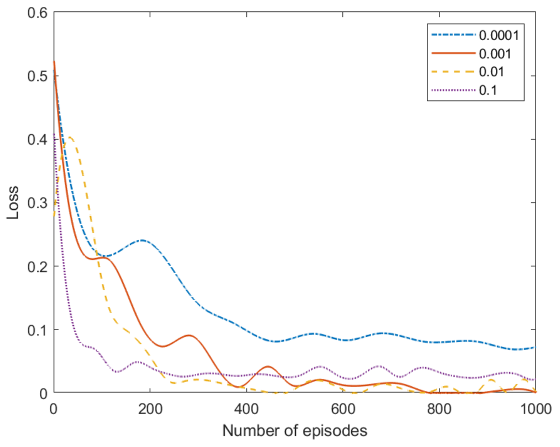

Fig. 3. Training process of DBFL under different learning rate

A. Simulation Setting

We consider a cache-enabled D2D network scenario with 3 clusters covering 5 UEs separately. There are 10 content items with the same size. We assume that a UE can fetch items from any neighbors via the D2D link within the communication range. All UEs have the same storage with the size of $c_u = 1000MB$. The content request probability of a UE is modeled by Zipf distribution [8]. We set different content popularity parameters based on the heterogeneous preferences of UEs.

For each DQN in DBFL, we set the number of input layer neurons, hidden layer neurons, and output layer neurons as 20, 10, and 10, respectively. We use $Sigmod$ as the activation function from the input layer to the hidden layer, and $ReLU$ from the hidden layer to the output layer. The size of the reply buffer in local model training is set to 100. Each round, we randomly select 5 continuous transitions from the buffer to calculate the target loss. The learning rate $\alpha = 0.001$ and the influence factor $\gamma = 0.1$.

B. Numerical Results

We first conduct the simulation in an ideal environment without any fault or malicious nodes to verify the convergence of DBFL. Fig. 3 shows the loss curves of our proposed DBFL under different learning rates. We find that all these four curves of loss value converge after around 500 episodes. Moreover, DBFL with a learning rate of 0.001 achieves a fast and stable convergence speed.

With the aim of evaluating the attack/fault resistance of our proposed DBFL-based caching scheme, we compare the reduced transmission latency of these four caching schemes under an omission fault. Fig. 4 shows that the reduced latency of all the four schemes. From this figure, we can see that the DBFL-based scheme and FL-based scheme achieve higher average rewards than the DRL-based scheme. The reason is that the local models in DBFL and FL are updated by the global model every 10 rounds, which can ensure all nodes

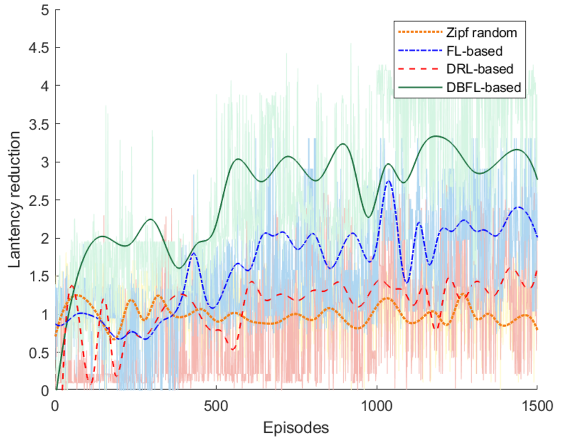

Fig. 4. Comparisons of latency reduction under omission faults

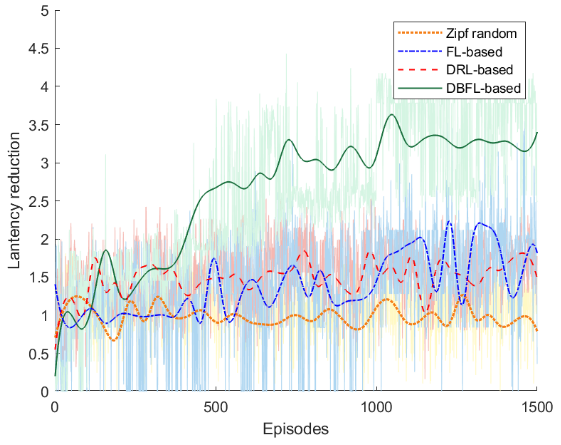

Fig. 5. Comparisons of latency reduction under Byzantine faults

obtain the well-performed model. Moreover, we observe that the DBFL-based scheme outperforms the FL-based scheme. This is because that fault nodes cannot be detected and replaced in the FL-based scheme, which degrades the accuracy of the global model in FL.

Finally, we compare the latency under Byzantine faults for all the four caching schemes as shown in Fig. 5, where a malicious node is considered in the network. From Fig. 5, we can see that the DRL-based and FL-based schemes perform unsatisfactorily and bring only a small delay reduction to the system, although their performances are better than that of the Zipf random caching scheme. Moreover, we observe that the DBFL caching scheme always achieves the highest latency reduction. The reason is that the impact of Byzantine faults is minimized by blockchain through preventing the malicious node from participating in the training.

VI. Conclusion

In this study, we have developed an intelligent and privacy-preserving caching scheme DBFL in the D2D network. DBFL is based on a framework of FL underpinned by a double-layer blockchain system. We have illustrated the blockchain consensus of each layer and streamlined DBFL, including FL model training and model data recording on the blockchain. We have conducted simulations in scenarios with and without malicious attacks, where numerical demonstrated both the improvements of caching performance and the reliability of resisting attacks. In general, this work can be seen as a pioneer to explore the interplay of blockchain and FL, thus developing an intelligent and trusted caching scheme under an unreliable wireless network.

References

[1] R. Amer, M. M. Butt, M. Bennis, and N. Marchetti, “Inter-cluster cooperation for wireless D2D caching networks,” *IEEE Transactions on Wireless Communications*, vol. 17, no. 9, pp. 6108–6121, 2018.

[2] M. Ji, G. Caire, and A. F. Molisch, “Fundamental limits of caching in wireless D2D networks,” *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 849–869, 2015.

[3] L. Li, Y. Xu, J. Yin, W. Liang, X. Li, W. Chen, and Z. Han, “Deep reinforcement learning approaches for content caching in cache-enabled D2D networks,” *IEEE Internet of Things Journal*, vol. 7, no. 1, pp. 544–557, 2019.

[4] D. Feng, G. Yu, C. Xiong, Y. Yuan-Wu, G. Y. Li, G. Feng, and S. Li, “Mode switching for energy-efficient device-to-device communications in cellular networks,” *IEEE Transactions on Wireless Communications*, vol. 14, no. 12, pp. 6993–7003, 2015.

[5] D. Feng, L. Lu, Y. Yuan-Wu, G. Y. Li, G. Feng, and S. Li, “Device-to-device communications underlaying cellular networks,” *IEEE Transactions on communications*, vol. 61, no. 8, pp. 3541–3551, 2013.

[6] W. Zhuang, Q. Ye, F. Lyu, N. Cheng, and J. Ren, “SDN/NFV-empowered future IoV with enhanced communication, computing, and caching,” *Proceedings of the IEEE*, vol. 108, no. 2, pp. 274–291, 2019.

[7] Q. Ye, W. Shi, K. Qu, H. He, W. Zhuang, and X. Shen, “Learning-based computing task offloading for autonomous driving: A load balancing perspective,” in *Proc. ICC*, vol. 21, 2021, pp. 1–6.

[8] W. Jiang, G. Feng, S. Qin, T. S. P. Yum, and G. Cao, “Multi-agent reinforcement learning for efficient content caching in mobile D2D networks,” *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1610–1622, 2019.

[9] B. Chen and C. Yang, “Caching policy for cache-enabled D2D communications by learning user preference,” *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6586–6601, 2018.

[10] R. Zhang, F. R. Yu, J. Liu, T. Huang, and Y. Liu, “Deep reinforcement learning (DRL)-based device-to-device (D2D) caching with blockchain and mobile edge computing,” *IEEE Transactions on Wireless Communications*, vol. 19, no. 10, pp. 6469–6485, 2020.

[11] Y. Sun, W. Jiang, G. Feng, P. V. Klaine, L. Zhang, M. A. Imran, and Y.-C. Liang, “Efficient handover mechanism for radio access network slicing by exploiting distributed learning,” *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2620–2633, 2020.

[12] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.

[13] Y. Sun, L. Zhang, G. Feng, B. Yang, B. Cao, and M. A. Imran, “Blockchain-enabled wireless internet of things: Performance analysis and optimal communication node deployment,” *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5791–5802, 2019.

[14] Y. Li, B. Cao, M. Peng, L. Zhang, L. Zhang, D. Feng, and J. Yu, “Direct acyclic graph-based ledger for Internet of Things: performance and security analysis,” *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1643–1656, 2020.

[15] S. Tamoor-ul Hassan, M. Bennis, P. H. Nardelli, and M. Latva-Aho, “Caching in wireless small cell networks: A storage-bandwidth tradeoff,” *IEEE Communications Letters*, vol. 20, no. 6, pp. 1175–1178, 2016.

[16] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[17] D. Ongaro and J. Ousterhout, “In search of an understandable consensus algorithm,” in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, 2014, pp. 305–319.

[18] M. Castro, B. Liskov *et al.*, “Practical byzantine fault tolerance,” in *OSDI*, vol. 99, no. 1999, 1999, pp. 173–186.