Performance Improvements in Software-defined Vehicular Ad Hoc Networks

by

Dajun Zhang, **B.Eng., M.Eng.**

A dissertation submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa, Ontario
January, 2020

The undersigned hereby recommends to the

Faculty of Graduate and Postdoctoral Affairs

acceptance of the dissertation

Performance Improvements in Software-defined Vehicular Ad Hoc Networks

submitted by **Dajun Zhang**, **B.Eng., M.Eng.**

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering

_____

_____

_____

_____

_____

Carleton University

January, 2020

Abstract

In this dissertation, we investigate performance improvements in software-defined and virtualized vehicular ad-hoc networks (VANETs) with advanced technologies. We firstly present a deep reinforcement learning (DRL) approach in software-defined vehicular ad-hoc networks with trust management. In this research, we propose to use a deep Q-learning (DQL) approach and the centralized control mechanism of SDN to address the bad influences of the malicious nodes in VANETs. The trust of each vehicle and the reverse delivery ratio are considered in a joint optimization problem, which is modeled as a Markov decision process with state space, action space, and reward function. DQL is used to obtain the best link quality policy under the bad influence of the malicious vehicles in the inter-vehicle communication for ITS. Specifically, by decoupling the control layer from the device layer, the DQL algorithm is used in a logically centralized controller in the control plane of the proposed scheme.

Secondly, we focus on distributed SDN and blockchain technology for connected vehicles in smart cities. Specifically, we propose a novel blockchain-based hierarchical distributed software-defined VANET framework (block-SDV) to establish a secure and reliable architecture that operates in a distributed way to overcome the security issues of VANETs. In order to achieve the goal of maximizing the system throughput and to ensure the trust information is not tampered with, we propose a blockchain-based consensus protocol that interacts with the domain control layer with the blockchain system. We aim to use this consensus protocol to securely collect and synchronize

the information received from different distributed controllers. Specifically, we give consensus steps and theoretical analysis of the system throughput. A dueling deep Q-learning (DDQL) approach is used in the domain control layer in order to obtain the best policy for maximizing the system throughput under the effects of malicious vehicles in vehicular networks.

Thirdly, we propose a novel framework of blockchain-based mobile edge computing for a future VANET ecosystem (BMEC-FV), in which we have adopted a hierarchical architecture. In the underlying VANET environment, we propose a trust model to ensure the security of the communication link between vehicles. Multiple MEC servers calculate the trust between vehicles through computing offloading. Meanwhile, the blockchain system works as an overlaid system to provide management and control functions. We aim to optimize the system throughput and the quality of services (QoS) of the users in the underlaid MEC system. The blocksize of blockchain nodes, the number of consensus nodes, reliable features of each vehicle, and the number of producing blocks for each block producer are considered in a joint optimization problem, which is modeled as a Markov decision process with state space, action space, and reward function. Simulation results are presented to show the effectiveness of the proposed BMEC-FV framework.

Acknowledgments

To pen down this section of my dissertation means I have almost arrived at the destination that I had dreamed every night. This exciting and unbelievable journey started when I wrote my first email to my supervisor, Prof. F.Richard Yu. Thus, I would like to begin this section by thanking him. Without his invaluable support and wonderful supervision, this would have been impossible. His technical insight and on-going encouragement have been a constant source of the motivation. The innumerable discussions with him and his ideas on the research projects have been the most dispensable input for my research. His comments and suggestions on my works not only increase the quality of the research but also provide a source of thought for my career. This work is as much his contribution as mine for the fact that he has always wholeheartedly supported every decision I have made during this wonderful study. He has better prepared me for the world and I will always be indebted to him for anything I achieve in my future life.

I would like to thank Dr. Ruizhe Yang, for her valuable comments and suggestions. I am grateful for the time and patience that she donated towards teaching me. I also would like to thank Prof. Feng Ye, Prof. Amiya Nayak, Prof. Shichao Liu, and Prof. Changcheng Huang, for serving on my dissertation committee. Special thanks go to Dr. Matt Ma, Dr. Chao Qiu and Dr. Fengxian Guo, who provided me valuable and constructive suggestions on my works.

Special thanks also go to my friend Yue Shen who gives me in Canada a memorable

and often breathtakingly beautiful adventure. I also want to express my thanks to Di Liu, Zhicai Zhang, and others who generously provide friendship in Ottawa to make my life here amazing.

There is no word to express my gratitude to my family, and in particular, my grandmother and my parents. This achievement would have not been possible without their unconditional love, support and encouragement. Thanks for always believing in me and making things easiser just so that I didn't have to worry about anything but my research. At last, I want to say thanks to my father who is an excellent professor and gives me a lot of confidence. I believe he can see my achievement even we are not in the same country.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ANN	Artificial Neural Network
AODV	Ad-Hoc On-Demand Distance Vector
DDQL	Dueling Deep Q-Learning
DL	Deep Learning
DQL	Deep Q-Learning
DRL	Deep Reinforcement Learning
DRQN	Deep Recurrent Q-Network
DNN	Deep Neural Network
ETX	Expected Transmission Count
ITS	Intelligent Transportation System
IoT	Internet of Things
LSTM	Long Short-Term Memory
MANETs	Mobile Ad-Hoc Networks
MDP	Markov Decision Process

MEC	Mobile Edge Computing
MLP	Multi-layer Perceptron
NIB	Network Information Base
P2P	Peer-to-Peer
RBM	Restricted Boltzmann Machine
RBFT	Redundant Byzantine Fault Tolerance
SDVs	Software-Defined Vehicular Networks
SDN	Software-Defined Networking
SAE	Stacked Auto-Encoder
SDN	Software-Defined Networking
SNR	Signal to Noise Ratio
VANETs	Vehicular Ad-Hoc Networks

Chapter 1

Introduction

1.1 Background

In this chapter, we main introduced the background information about the VANETs, mobile edge computing, deep reinforcement learning and blockchain technology. Meanwhile, some challenges and motivation are illustrated. Literature review shows the recent improvements about the VANETs, MEC, and blockchain technology.

1.1.1 VANETs and Mobile Edge Computing

In recent years, vehicular ad-hoc networking is one of the key components of building a smart city in the future. A large number of researchers have applied various technologies to optimize the performance of VANETs. In this research, we mainly use SDN, blockchain technology, MEC, and various deep reinforcement learning methods to ensure the secure communication of the VANET nodes, the transmission of large amounts of data, and the autonomous learning function given to the VANET nodes. In particular, centralized and distributed SDN controllers and MEC servers, as agents of different system architectures, are responsible for deep reinforcement learning tasks. At the same time, the blockchain system is used as a carrier for data sharing of the

Internet of Vehicles and has achieved consensus among multiple agents to ensure the security and stability of data transmission. In general, we aim to optimize the communication security and quality of IoV nodes and improve the performance of the system architecture through the fusion of multiple technologies.

According to the definition of European Telecommunications Standards Institute (ETSI), mobile edge computing (MEC) focuses on the environment that provides users with IT services and cloud computing capabilities at the edge of the mobile network and is intended to be closer to mobile users to reduce the delay in network operations and service delivery. The mobile edge computing architecture is divided into three levels: the system layer, the host layer, and the network layer [5–7]. The system architecture proposed by ETSI shows the functional elements of MEC and the reference nodes between each functional element.

The MEC is located between the wireless network access point and the wired network. The traditional wireless access network has the advantages of service localization and close-range deployment, which brings high-bandwidth and low-latency transmission capabilities. In the MEC mode, by sinking network services to the wireless network access side closer to the user, the direct benefits are that the user can noticeably reduce the transmission delay and the network congestion is significantly controlled. MEC provides application programming interfaces (APIs) that open basic network capabilities to third parties, enabling third parties to complete on-demand customization and interaction based on business needs.

The VANET technology can sense vehicle behavior and road conditions through various sensors, improve vehicle safety, reduce the degree of traffic congestion, and also bring opportunities for value-added services, such as vehicle positioning and finding parking locations. This technology has not yet reached maturity, and the delay from the vehicle to the cloud is still between 100 ms and 1 s, which is far from meeting the

requirements. By extending the connected vehicle cloud system to a highly distributed mobile base station environment, MEC can bring data and applications closer to the vehicle, which can effectively reduce the round-trip time of data. The application runs on the MEC server deployed at the LTE base station site and provides related functions on the roadside. By receiving and analyzing messages from nearby vehicles and roadside sensors, edge computing is able to propagate hazard warnings and delay-sensitive information with 20 ms end-to-end latency. Low latency enables nearby vehicles to receive data in milliseconds, allowing drivers to react immediately.

1.1.2 Deep Reinforcement Learning

Reinforcement Learning

The main components of the reinforcement learning (RL) include agent, environments, and action. The main consideration of reinforcement learning is a series of tasks with which the agent interacts with the environments. Consequently, no matter what the task is, any task includes a series of actions, observations, and rewards. The meaning of the reward is that the environments are changed since the agent interacts with the environments. The reward is used to represent the degree of the environments change.

Fig. 1.1 shows the process of the agent and environment interaction. In each time-step, the agent chooses an action from a given action set. This set can be a continuous or a discrete action set. The number of action sets will deeply influence the difficulty of solving the tasks.

The goal of RL is to get more rewards. In each time-step, the agent executes a new activity through the current observation. Each observation acts as the state of the agent. Consequently, the relationship between the state and action is the mapping, which means that one state corresponds with one action or corresponds with the

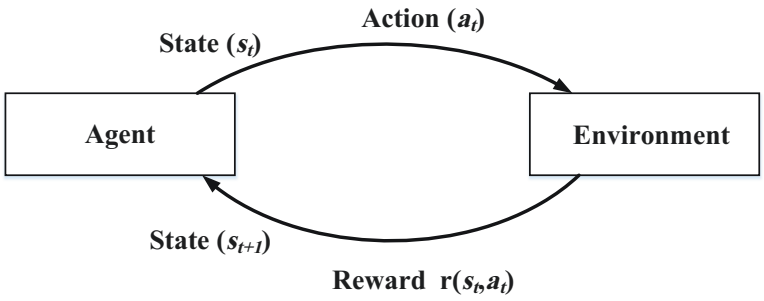

Figure 1.1: A basic diagram of a RL system. The agent takes an action according to the current state and then receives a reward. $r(\mathrm{s}_t, a_t)$ **denotes the immediate reward that the agent receives after performing an action** a_t **at the state** s_t**.**

probabilities of different actions (generally, we use the probability to represent the relationship. The highest probability represents the agents action in the next time-step). The process from the state to the action is called policy.

The main target of RL is to find an optimized policy that makes the reward achieve the highest level. We cannot know the optimized policy from the beginning of the task, so we begin from stochastic policies. We get a series of states, actions, and rewards from the stochastic policies. This is called the sample. A RL algorithm is used to enhance the policy according to those samples in order to make the rewards better than before.

There are two methods to get the optimal reward in reinforcement learning: value iteration and policy iteration. Strategy iteration is to learn the optimal policy according to each reward when the policy is unknown. For a specific MDP problem, each time a strategy is initialized at first, a value function v (s) is calculated according to the strategy, and the strategy is updated according to the greedy strategy through this value function, and the optimal strategy and the optimal value function are finally iterated. Value iteration is to obtain the optimal value function and optimal strategy according to the strategy under the condition of known policy and MDP

model.

Comparison of RL with supervised learning and unsupervised learning:

- Supervised learning is done from an already marked training set. The characteristics of each sample in the training set can be thought of as a description of the situation, and its label can be considered as the correct action that should be performed. However, supervised learning cannot learn the context of interaction, because the example of obtaining the desired behavior in the problem of interaction is very impractical. The agent can only learn from its own experience, and the behavior taken in the experience is not necessarily optimal. It is very appropriate to use RL at this time, because RL does not use the correct behavior to guide, but uses existing training information to evaluate behavior.

- The purpose of unsupervised learning is to find hidden structures from a bunch of unlabeled samples, but the purpose of RL is to maximize the reward signal.

- In general, RL differs from other machine learning algorithms in that there is no supervisor and only one reward signal. Meanwhile, the feedback of RL is delayed and not immediately generated. Time has an important meaning in RL, which the agent behavior affects a series of system states.

RL adopts the method of obtaining the sample while learning, updating its own model after obtaining the sample, using the current model to guide the next action, updating the model after the next action is obtained, and iteratively repeating until the model convergence. In this process, when the current model is available, the next action is selected to be most beneficial to the improvement of the current model. This involves two very important concepts in RL: exploration and exploitation. Exploration refers to the selection of actions that have not been executed before, to explore more possibilities; exploitation refers to the selection of actions that have

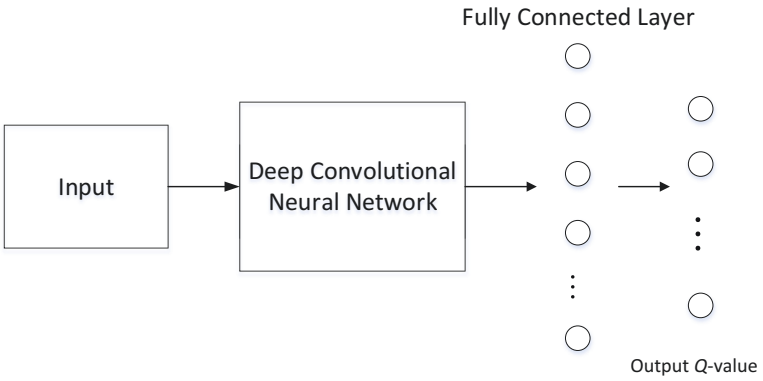

Figure 1.2: The basic structure of DQN.

been executed to refine the model of known actions.

Deep Reinforcement Learning

Deep Q*-network (DQN)*: Mnih *et al.* [49] combined the convolutional neural network with Q-learning in traditional RL and proposed a deep Q network model. This model is used to process visual perception-based control tasks and is a groundbreaking work in the field of DRL.

The input of the DQN model is a pre-processed image matrix that undergoes a nonlinear transformation of the convolutional layers and the fully connected layers. Finally, the Q-value of each action is generated at the output layer. Fig. 1.2 shows the model architecture of DQN.

Fig. 1.3 depicts the training process of DQN. In order to alleviate the instability of the nonlinear network representation value function, DQN mainly made three improvements to the traditional Q-learning algorithm.

- DQN uses the experience replay mechanism during training to obtain samples. At each time step t, the sample obtained by interfacing the agent with the environment is stored in the replay memory D. During training, a small batch of

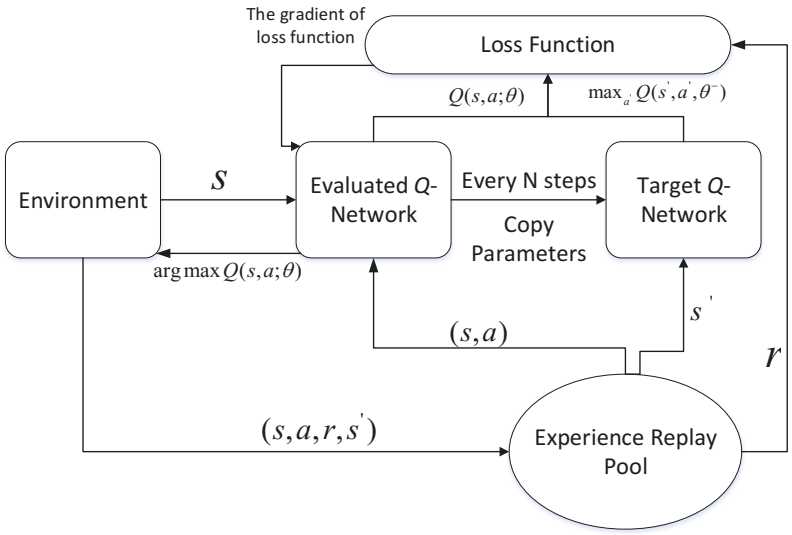

Figure 1.3: The training process of DQN. S denotes the state space, r denotes the reward.

samples is randomly selected from D at each time step, and the network parameter is updated using a stochastic gradient descent algorithm. When training deep neural networks, it is usually required that samples are independent of each other. This random sampling method greatly reduces the correlation between samples, thus improving the stability of the algorithm.

- In addition to using a deep convolutional network approximation to represent the current value function, DQN uses a target Q-network to generate the target Q-value. Specifically, the evaluated network represents the output of the current value network and is used to evaluate the value function of the current state action pair; the target Q-network represents the output of the target value network, and the target Q-value is generally used to approximate the optimization objective of the value function. The parameter θ of the current evaluated Q-network is updated in real-time, and the parameters of the current evaluated Q-network are copied to the target Q-network every N rounds of

iteration. DQN updates network parameters by minimizing the mean square error between the evaluated Q-value and the target Q-value.

- DQN reduces the reward value and error value to a limited interval, which ensures that the Q-value and the gradient value are within a reasonable range, which improves the stability of the algorithm. Experiments show that DQN exhibits a competitive level comparable to that of human players when solving complex problems such as Atari 2600 games [1]. When solving various types of visual perception-based DRL tasks, DQN uses the same set. Generally speaking, the DQN method has stronger adaptability and versatility compared with the traditional reinforcement learning approaches.

Improvements of Deep Q*-network*: The improvement of the DQN model is generally achieved by adding new functional modules to the original network. For example, a recurrent neural network structure can be added to the DQN model, so that the model has the ability to remember on the time axis. This section mainly introduces two improved versions of DQN models, namely dueling DQN and deep recurrent Q-Network (DRQN).

Dueling DQN: In many DRL tasks, the value function of the state action pair is different due to different actions. However, in some states, the value function is independent of the action. Therefore, Wang *et al.* [2] design a dueling network structure and adds it to the DQN network model. As shown in Fig. 1.4, the network structure divides the abstract features extracted by CNN into two branches, one of which represents a state-value function and the other represents an advantage function. Through this competitive network structure, the agent can identify the correct behavior more quickly in the strategy evaluation process.

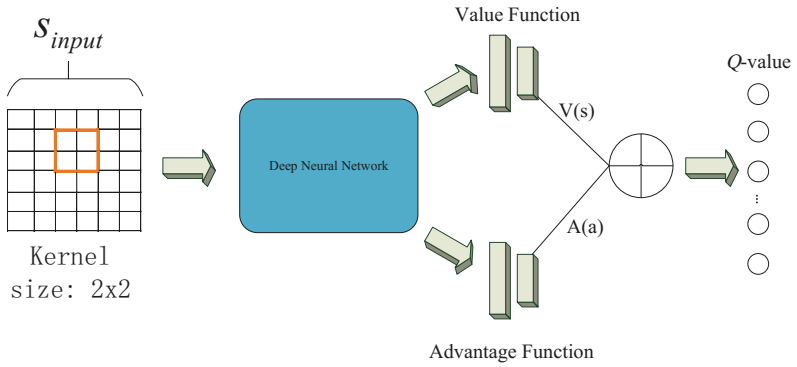

Figure 1.4: The network structure of dueling DQN.

1.1.3 Blockchain Technology

The Definition of Blockchain

Blockchain technology [3, 4] uses blockchain data structures to validate and store data, use distributed node consensus algorithms to generate and update data, use cryptography to secure data transfers and access, and use smart contracts to program and manipulate data. This is a new distributed infrastructure and computing paradigm. Briefly speaking, in the blockchain system, the transaction data generated by each participant is packaged into a data block, and then the data blocks are arranged in chronological order to form a chain of data blocks. The subject has the same data chain and cannot be unilaterally falsified. Any modification of the information can only be carried out with the consent of the agreed portion of the subject, and only new information can be added, and the old information cannot be deleted or modified, thereby realizing the inter-subject. Information sharing and consistent decision-making ensure that the identity of each subject and the transaction information between entities are not falsified, and are open and transparent. Block-s, accounts, smart contracts, and consensus form the common model of the current blockchain system.

- The change history of the state is recorded by the chain structure, and the

status "snapshot" of each change is recorded in the form of *block*.

- The identity of the participant is represented by an asymmetric key pair, and the current information is recorded in some form of state database. This part is called an *account*.
- Defining commitments between participants through chain coding, this part is called *smart contract*.
- Through a certain algorithm to achieve a consistent state between multiple nodes, this process is called *consensus*.

The Characteristics of Blockchain

The blockchain is essentially a robust and secure distributed state machine. Typical technical components include consensus algorithms, P2P communication, cryptography, database technology, and virtual machines. This also constitutes the five core competencies that are essential to the blockchain:

- Storage of data: With the development of database technology and hardware storage computing power, the size of the blockchain continues to rise. Mature hardware storage computing power makes it possible to store the same data in large quantities at the same time.
- Common data: derived from the consensus algorithm, the various entities participating in the blockchain automatically reach consensus through the agreed decision-making mechanism, sharing the same credible data ledger.
- Distributed architecture: derived from P2P communication technology, blockchain enables point-to-point information to transfer between entities.

- Tamper-proof and privacy protection: Secure the identity and common information of each subject through cryptographic tools such as public key-private keys and hash algorithms.

- Smart contract: derived from virtual machine technology, the generated digital smart contract is written into the blockchain system to drive the execution of the smart contract through preset trigger conditions.

The Challenges of Blockchain

With the current blockchain system, there are still obvious shortcomings in transaction concurrency, data storage capability, versatility, functional completeness, and ease of use.

Transaction concurrency At present, the high-level concurrent transaction capability of the open source blockchain system is generally not high, and the consensus algorithm is an important aspect that restricts performance. Typical consensus algorithms used in blockchain are: proof of working (PoW), proof of service (PoS), delegated proof of service (DPoS), practical Byzantine fault tolerance (PBFT), etc.

Another important factor limiting performance is the block structure. At present, the typical blockchain ledger is designed as a single-chain structure of blocks, which means that all transactions can only be processed sequentially. Due to the lack of parallelism in transaction processing, it is difficult to achieve performance close to traditional centralized systems.

The transaction concurrency in many practical applications usually requires processing hundreds to thousands of transactions per second, which is much higher than the performance of typical blockchains including the public chain and the alliance chain. Performance can scale dynamically as the business grows. Therefore, there

is an order of magnitude difference between reality and goal, and it is necessary to continuously optimize and improve the high concurrent transaction performance of the blockchain system.

Data storage capability In terms of data storage capacity, as the blockchain data only increases, the need for more data storage continues to increase over time, and this trend grows even more when dealing with large amounts of data.

At present, the typical blockchain system recognizes the storage of reconciliation data. The typical implementation is based on the file system or simple key-value database storage. Without the design of distributed storage, there is also a relationship between data storage capability and actual needs. Larger gaps require more efficient ways to store big data.

Versatility Blockchain needs to adapt to diversified application requirements and to meet data sharing across business chains. This means that blockchains must have sufficient versatility and standards for data records to represent various structured and unstructured information, and must meet the cross-chain requirements required to expand the scope of business.

Most of the current blockchain systems use specific consensus algorithms, encryption algorithms, account models, and storage types; they also lack pluggable capabilities and cannot adapt to different scenarios.

1.2 Motivation

Vehicular ad hoc networks (VANETs), classified as a subset of mobile ad hoc networks (MANETs) in vehicular environments, are very important in *intelligent transportation systems* (ITSs) [8–12]. ITSs aim to ensure the safety and efficiency

of the connected vehicles' communication by applying data processing and sensor technologies to vehicles. The system architecture includes two main components: the intra-vehicle network and the inter-vehicle network [13]. The intra-vehicle network includes the collection of sensors that are located in vehicles. The communication between different sensors is bridged via Wifi, Ethernet, and so on. The inter-vehicle network comprises the communication between the vehicles and surrounding devices such as roadside units (RSUs) and roadside users.

However, ITSs still have problems with inter-vehicle communication. The security issues of current VANETs can be divided into three important categories [14]: 1) communication protocol hacking and man-in-the-middle attacks will badly influence the performance of vehicle-cloud communication; 2) protocol hacking and authentication failure can degrade the performance of short-distance communication on the connected vehicles; 3) malicious nodes become vehicle-to-vehicle communication threats. Current inter-vehicle communication faces frequent access and exit of vehicles. At this stage, the management of VANETs cannot effectively be implemented in the aspect of secure access control for the vehicles, and the detection mechanism for un-trusted or out-of-control nodes has not been established perfectly. Once there is a malicious node intrusion, the vehicle-to-vehicle information can be affected by blocking, forging, or tampering with the connected vehicles' communication, thereby destroying the authenticity of the vehicle-to-vehicle communication message and affecting the transmission of the road condition information. In order to mitigate the bad influences of malicious vehicles, the trust information of vehicles has been introduced in VANETs to detect the misbehaviors of malicious nodes.

As an independent control plane, SDN brings flexibility and programmability to the network. This brings awareness into the system so that it can adapt to changing

conditions and requirements. In order to meet the demands of secure vehicle-to-vehicle communication and dynamic management, SDN has been used in VANETs, called software-defined vehicular networks (SDVs) [15–17]. Specifically, this awareness enables SDVs to make better decisions based on combined information from multiple sources (rather than just personal perception from each node). Moreover, dynamics and flexibility can respond to emergencies, suitable for emergency situations and changing needs. The awareness of SDV enables the system to make more informed routing decisions. For example, in vehicle-to-vehicle communication, malicious node intrusion will affect the authenticity of vehicle-to-vehicle communication, and then affect the routing strategy of vehicle-to-vehicle communication. When the SDN controller detects this, it can restart routing to improve network availability and reduce the impact of malicious nodes on overall network performance.

With a large number of vehicles in VANETs, the communications between different objects in a large-scale and high mobility environment generate a large volume of real-time, high-speed, and uninterrupted data streams. These data streams flow through multiple vehicles according to the established route of VANETs, which will require a large number of diverse resources from VANETs. As a result, scalable and adaptive SDN controllers are essential for efficient data processing of VANETs. With the increasing scale of current VANETs, traditional VANET frameworks with centralized SDN control mechanisms obviously cannot match the diversification of VANET traffic requirements. To tackle these challenges, a distributed SDN control strategy has emerged as a promising network design, which can efficiently and dynamically manage the resources in VANETs. Distributed software-defined VANETs (SDVs) are motivated by the performance, scalability, and availability requirements of large operator networks. There are two main categories of distributed SDN control architectures: flat SDN control and hierarchical SDN control [18]. The authors

of [19] design a distributed SDN flat control plane for OpenFlow, called HyperFlow. It localizes decision making to an individual controller, thus minimizing the control plane response time to data plane requests. Onix architecture for distributed SDN is proposed by [20]. This flat control platform uses a network information base (NIB) to aggregate and share network-wide views. Hierarchical SDN control architecture means that the network control plane is vertically partitioned into multiple layers depending on the requested services. Kandoo [21] is designed by a hierarchical two-layer control plane, which divides the control applications into global and local ones. Therefore, distributed SDN control plane architectures can mitigate the issues caused by centralized SDN control architectures such as poor scalability and performance bottlenecks.

Although distributed SDN control platforms bring many advantages comparing with traditional centralized SDN control architectures, they still have an important problem, which is how to achieve synergy among multiple controllers when they are deployed into SDVs. The existing mechanisms have the following issues: 1) the traditional consensus mechanisms bring extra overheads in each controller; 2) those mechanisms are difficult to ensure the safety and liveness; 3) the existing schemes have challenges when the SDN control plane is enlarged to a large-scale scenario.

In order to address the above challenges, we consider using recent advances in *blockchain technology* [22]. Blockchain can be seen as a distributed third-party system to achieve agreement among different nodes without using centralized trust management. In the safety and data sharing aspects of connected vehicles' communications, distributed SDV is a partially trusted environment. Therefore, we consider using blockchain technology in distributed SDV. Recently, blockchain technology has attracted the attention of many researchers in a wide range of industries. The main reason for this is that, with the blockchain technique, applications can be operated

in distributed ways to achieve trustworthiness, whereas previously they had to pass through a trusted intermediary. The authors of [23] propose a distributed blockchain cloud architecture with software-defined networking (SDN) for IoT. This framework aims to address the low latency and security issues in traditional network architecture. Sharma *et al.* [24] propose a distributed blockchain-based secure SDN architecture for IoT networks. The authors of [25] propose a novel blockchain-based anonymous reputation system (BARS) to establish a privacy-preserving trust model for VANETs. A trust-based framework block-VN is proposed in [26] that provides a reliable and secure architecture operating a distributed way to establish the new distributed transport management system. Singh *et al.* [27] use blockchain technology as a backbone of connected vehicles' data sharing in VANETs.

At present, the development of VANET technology and service capabilities continues to improve. With the help of network connections and information interaction between multiple platforms (people, cars, clouds), a large number of new applications have been spawned. The gradual miniaturization of hardware is achieving large-scale deployment in VANETs [28]. However, the large-scale deployment of VANETs is limited by the energy, memory size, and computing resources of mobile vehicles [29]. At the same time, VANET nodes have the intensive computing power and sensitive latency requirements that can rely on advanced and improved wireless technologies and computation offloading. Future VANETs not only need to support large-scale wireless access but also provide computing offload for mobile users of VANETs.

In order to meet the demands of mobile users, the future network structure will be more heterogeneous, hierarchical, and dense. In the development of VANETs, the security of vehicle-to-vehicle communication and the high requirements for delay have always been obstacles to the development of the Internet of Vehicles. One reason is the high mobility of the VANET node and the complex time-varying character of

the communication channel. However, due to the unprecedented level of VANET density in the future, communication security and latency management are highly complex. Therefore, future VANETs tend to choose smarter and more distributed access technologies.

In future communication networks, the edge clouds will be deployed in the heterogeneous network and will be able to provide computation offloading services to users [30]. One of the promising paradigms is MEC [31]. The basic idea of MEC is to migrate the cloud computing platform from the inside of the mobile core network to the edge of the mobile access network. By deploying edge nodes with functions such as computing, storage, and communication in the traditional wireless access network, users can be provided higher bandwidth and lower latency data services. Meanwhile, the deployment of the edge nodes can greatly reduce the network load of the core network, while reducing the bandwidth requirements of data services for network backhaul.

Although MEC provides a near-distance, ultra-low latency, high bandwidth, and real-time access to wireless network information services by providing computing, storage, and communication services within the wireless access network, MEC still faces many challenges. First, the fairness of resource sharing in MEC systems is one of the key factors affecting the quality of user services and the overall performance of the network. How to implement fairness-based resource optimization management and network load balancing in the case of a large number of MEC edge computing nodes and end-user access nodes in the network is the focus of research in related fields. Second, in the typical application scenarios of MEC with high rate and low latency, how to effectively ensure the continuity and seamless handover of end-users is an important issue that the MEC system needs to solve.

Currently, many mobile network control problems have been solved by constrained

optimization, dynamic programming and game theory approaches. Unfortunately, these methods either make strong assumption about the objective functions (e.g. function convexity) or data distribution (e.g. Gaussian or Poisson distributed), or suffer from high time and space complexity. However,, as mobile networks become increasingly complex such assumptions sometimes turn unrealistic. The objective functions may be further affected by large set of variables which poses severe computational and memory challenges to these mathematical approaches. On the other hand, despite the Markov property, deep reinforcement learning does not make strong assumptions to the target system. It employs function approximation which perfectly addresses the problem of large state-action space, enabling reinforcement learning to scale to network control problems that were previously intractable.

As the core technology behind Bitcoin and Ethereum, blockchain has received extensive attention in academia and industry [3,4,32]. The current blockchain system is essentially a secure, shared, and distributed ledger. It allows two parties in a peer-to-peer network to communicate and exchange resources. It is worth noting that the decision-making of the blockchain system is allocated by the majority rather than a single centralized decision process [33]. Due to the decentralized and secure nature of the blockchain, it is considered a candidate for establishing a secure and self-organizing MEC ecosystem for future wireless networks.

1.3 Literature Review

1.3.1 Trust-based VANETs

In order to meet the requirements of secure communications in VANETs, researchers have proposed many security mechanisms using trust information of vehicles. The authors of [34] propose a privacy-preserving route reporting scheme for

traffic management in VANETs. The trust information used in this paper is to mitigate the attack. An efficient VANET framework that is proposed in [35] depicts a trust model called the implicit web of trust in VANET (IWOT-V) to evaluate the trustworthiness of vehicles. Hu *et al.* [36] introduce a reliable trust-based platoon service recommendation scheme, which enables the user vehicle to avoid malicious vehicles in a VANET environment. Tan *et al.* [37] present a fuzzy logic approach to establish a trust management system of VANETs. The fuzzy logic approach is used to evaluate the trust information of vehicles. A special trust management approach for cognitive radio for vehicular ad-hoc networks (CR-VANETs) is proposed by Ying *el al.* [38]. The authors define a novel trust model in order to enhance the security of data transmission in CR-VANETs. Guleng [39] proposes a trust management approach for VANETs. The authors use a fuzzy logic-based trust calculation scheme to evaluate the direct trust of each vehicle. Meanwhile, a reinforcement learning method also is used in this paper to compute any vehicle's indirect trust value. In order to solve the problem of energy stability, Aujla *et al.* [40] propose an unique conceptual solution using electric vehicles (EVs). The proposed solution deals with the problem of managing the miscellaneous power or power deficit at the charging stations (CSs) by utilizing EVs-as-a-service (EVaaS). In this paper, EVaaS not only provides opportunities to the owner of EVs to earn profit but also it helps to balance the demand and supply at the CSs. Meanwhile, this approach also uses the SDN paradigm for enabling faster communication between the entities involved.

As we can see, there is an emerging trend to use the trust information to keep the safety of VANETs. Since most existing works couple the trust information with system control, the trust management for VANETs is still a big issue for improving the security of VANETs. Hence, the control plane needs to decouple from the data plane in VANETs. SDVs aim to manage the VANET dynamically and ensure the

security of VANETs. However, there is a problem for current SDVs: when there are lots of vehicles communicating with each other in a VANET environment, a massive amount of data (i.e., trust information) will be generated. Without the extraction of useful information, the collected data of the SDN controller holds little or no value. Hence, the reinforcement learning approach appears as a tool to deal with the massive data generated in a large VANET environment. Reinforcement learning techniques provide an efficient way to analyze and extract useful information, then make appropriate decisions.

1.3.2 Reinforcement Learning based VANETs

Researchers have proposed many different schemes based on reinforcement learning approaches to address the security problems of VANETs. The main purpose of reinforcement learning is to gain knowledge from the input data, which can help users to reduce errors and to improve efficiency for problem-solving [41]. *Reinforcement learning* (RL), which is a branch of machine learning, is about agents interacting with different environments and learning optimal policies to maximize rewards from environments in a wide range of both social science and nature [42].

Recently, the RL algorithm has been used to address the security challenges in VANETs. Here, nodes have abilities to observe and learn from the VANET environment to determine the optimal security policy. Chettibi *et al.* [43] introduce a novel dynamic fuzzy logic-based ad-hoc on-demand distance vector (AODV) routing protocol, which combines a traditional Q-learning approach to improve the protocol efficiency. Each node uses the fuzzy logic mechanism and Q-learning algorithm to decide its REQuests (RREQs) in the route discovery process. The final results show that this routing protocol has better energy efficiency. Bhorkar *et al.* [44] propose a scheme that utilizes the reinforcement learning to opportunistically route the packets

in a specific environment, which is in the absence of reliable knowledge about the channel and network model. A trust-based Quality of Service (QoS)-oriented routing is proposed in [45]. The authors introduce a cognition layer interacting with the network layer. This scheme mainly includes two phases: path learning and trust learning. The final results show the proposed scheme has better end-to-end delay and communication overhead. Xiao *et al.* [46] propose unmanned aerial vehicles (UAVs) to relay the message of an OBU and improve the communication performance of VANETs against smart jammers that observe the ongoing OBU and UAV communication status and even induce the UAV to use a specific relay strategy and then attack it accordingly. The RL approach is used to get the optimal UAV relay strategy depends on the transmit cost and the UAV channel model. Jindal *et al.* [47] propose a SDN-enabled approach, named SeDaTiVe, which uses deep learning architecture to control the incoming traffic in the network in the vehicular cyber-physical system (VCPS) environment. The deep learning approach is used to learn the hidden patterns in data packets and creates an optimal route based on the learned features.

Although a lot of researches on reinforcement learning based VANETs have been proposed by many researchers, it still has a problem that is the curse of dimensionality. The traditional reinforcement learning methods use the Q-table to represent Q-values. However, this is hardly feasible on many issues of VANETs because, in the reality of VANETs, the amount of input data is really too large. The Q-table cannot have enough space to represent the final Q-values. Meanwhile, reinforcement learning still has some instability, and the causes are described in [48], thus we can use deep Q-network to represent the Q-value. Inspired by the current advance [49], we consider using a deep Q-learning approach to solve the challenges in reinforcement learning-based VANETs. The DQL approach enables an agent to determine optimal policies from different inputs (high dimensionality). According to [49], DQL has been

significantly improved by DeepMind compared with the traditional Q-learning approach. Specifically, the deep Q-learning approach utilizes a deep neural network to approximate Q-value functions [50].

1.3.3 Blockchain based VANETs and Distributed SDN

Traditionally, many researchers have proposed multiple solutions for distributed SDN architectures [18]. In summary, based on the configuration method between the controller and the switch, we divide them into a statically configured control architecture and a dynamically configured control architecture [25].

In [19], the author designed the first distributed SDN control plane solution for OpenFlow, called HyperFlow. The way it synchronizes network-wide views between multiple controllers is through a publish/subscribe messaging example. In [20], the authors proposed a distributed architecture different from HyperFlow, called the Onix architecture. The way they synchronize controllers is to aggregate and share network-wide views using the Network Information Base (NIB).

One of the core problems of statically configured control architectures is that it can lead to uneven load distribution among distributed controllers. Based on this defect, in [51], Berde *et al.* Designed a flexible distributed SDN control plane called ONOS. The work in [52, 53] proposed a layered SDN control architecture that simultaneously considered load balancing and energy consumption of multiple controllers.

It can be seen from the above various distributed SDN architectures that one of the most important issues of distributed SDN is how to reach consensus among multiple controller instances. Regardless of the consensus protocol used (such as publish/subscribe, slicing, NIB, and controller pools, etc.), using it in a connected car environment requires many challenges to be solved.

- In the existing proposed distributed SDN architecture, the traditional consensus

protocol is implemented "in-band", which will cause each controller to generate additional overhead when processing instance events, thereby weakening the inherent functions of each controller (for example, Routing decisions, network management, etc.). In particular, in the connected car environment, this problem is more serious due to the time-varying and dynamic nature of the connected car nodes. Therefore, third-party and "out-of-band" consensus protocols are needed in SDV.

- The traditional distributed SDN architecture ignores the security and liveliness of the overall system. As the backbone of the SDN architecture, if a security problem occurs in the control plane, it will bring incalculable consequences. In an open connected vehicle environment, many cyber-attacks are more likely to affect secure communications. Therefore, a reliable and reliable consensus protocol is urgently needed in SDV.

- With the widespread deployment of connected vehicles, the scope of application of traditional distributed consensus protocols is limited. Large-scale consensus protocols should be used in SDV.

These challenges in distributed SDN have stimulated the need to explore new consensus protocols in SDV. Inspired by the successful implementation of blockchain and security key synchronization [54] and data sharing [27] in secure transportation systems, we believe that blockchain technology may be a potential method to solve the challenges in distributed SDV. Blockchain is a distributed system that provides reliable services for a group of nodes that do not fully trust each other. It can be regarded as a third-party system, and agreements between nodes can be reached without a central trust agent. It has the characteristics of reliability and can largely use the system scale. In addition, given the fact that distributed SDVs operate in

a partially trusted environment and the advantages of permissioned BCs, we utilize permissioned BCs in distributed SDVs.

1.3.4 MEC based VANETs

Researchers have proposed many different schemes based on MEC to address the data transmission of VANETs. In the MEC framework, the MEC server is the core of the entire system, and the MEC system covering the mobile terminal is composed of one or more MEC servers. By deploying the MEC server between the radio access network and the core network, the MEC system will be able to provide end-users with more efficient, lower latency computing, storage, and communication services on the wireless network side (near end of the network), and thus can improve the quality of service (QoS) experience for end-users.

Recently, the MEC approach has been used to improve the QoS in VANETs. Liu *et al.* [55] propose a SDN-enabled network architecture assisted by MEC to provide low-latency and high-reliability communication in VANETs. The proposed SDN-enabled heterogeneous vehicular network assisted by MEC can provide desired data rates and reliability in V2X communication simultaneously. Huang *et al.* [56] propose an idea and control scheme for offloading vehicular communication traffic in the cellular network to vehicle to vehicle (V2V) paths. Luo *et al.* [57] propose a multi-place multi-factor prefetching scheme to meet the rapid topology change and unbalanced traffic. Zhang *et al.* [58] propose a cloud-based MEC off-loading framework in vehicular networks. In this framework, they study the effectiveness of the computation transfer strategies with vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) communication modes.

Although many researchers have already done many excellent works on MEC based VANETs, the subject still has some problems that need to be solved. MEC

aims to provide high performance, high bandwidth, high storage, low latency services. To achieve these technical parameters, the first problem to be solved is how to make the service dynamic and adaptive. Therefore, in chapter 3, we propose a framework based on deep Q-learning and blockchain to establish a secure and self-organized MEC ecosystem for future VANETs.

1.4 Dissertation Organization and Contributions

In chapter 2, we propose to use a deep Q-learning approach and the centralized control mechanism of SDN to address the bad influences of the malicious nodes in VANETs. A deep Q-learning algorithm is introduced in [49]. Compared with the traditional Q-learning mechanism, DQL makes significant improvements. In this chapter, DQL is used to obtain the best link quality policy under the bad influence of the malicious vehicles in the inter-vehicle communication for ITSs. Specifically, by decoupling the control layer from the device layer, the DQL algorithm is used in a logically centralized controller in the control plane of a software-defined trust-based deep Q-learning framework (SD-TDQL).

In this line of research, the distinct features are as follows.

- Based on the programmability of the SDN control plane, an integrated framework named SD-TDQL is proposed to enhance the performance of data forwarding, link quality, and security of the connected vehicles' communication.
- In this framework, a trust model is designed to evaluate neighbors' behaviors that are used to forward VANET routing information. Since malicious nodes in VANETs can badly affect the network performance [59], the trust model is used to detect the bad vehicles.

- We use the expected transmission count (ETX) to characterize the vehicle-to-vehicle communication link quality. In SD-TDQL, the SDN controller located in the cloud as a decision center needs to process a large amount of input data of the VANET nodes. Due to the dynamic and time-varying nature of the connected car environment, we adopt a Markov decision process to define the state and action space of the system. A deep Q-learning approach has been used in this research. The DQL algorithm has been used in communication networks to achieve automatic routing selection [60–62]. Here, the DQL approach is used to obtain the best secure communication link policy under the bad effects of some malicious vehicles in vehicular networks.

- Simulation results with different architectures of deep Q-networks show the effectiveness of the proposed SD-TDQL framework. The results distinctly illustrate that our proposed scheme integrated the SDN and DQL approach can significantly improve network performance.

In chapter 3, we focus on distributed SDN and blockchain technology for connected vehicles in smart cities. Specifically, we propose a novel blockchain-based hierarchical distributed software-defined VANET framework (block-SDV) to establish a secure and reliable architecture that operates a distributed way to overcome the security issues of VANETs.

The main contributions of this line of research are summarized as follows.

- Based on the programmable area control plane originated from SDN, we propose an integrated framework that can enable dynamic orchestration of VANET security for communications of connected vehicles. The distributed SDN controllers in the area control layer have the abilities to collect vehicles' trust information, and each area controller can transfer its collected trust information to the domain control layer.

- In order to achieve the goal of maximizing the system throughput and to ensure the trust information is not tampered, we propose a blockchain-based consensus protocol that interacts the domain control layer with the blockchain system. We aim to use this consensus protocol to securely collect and synchronize the information received from different distributed controllers. Specifically, we give consensus steps and theoretical analysis of the system throughput.

- A dueling deep Q-learning approach is used in the domain control layer in order to obtain the best policy for maximizing the system throughput under the effects of malicious vehicles in vehicular networks. Simulation results with different parameters and architecture of deep Q-networks show the effectiveness of the proposed scheme.

In chapter 4, we focus on distributed MEC servers and blockchain technology for VANETs. We propose a novel blockchain-based mobile edge computing for future VANETs (BMEC-FV). Considering the reliable features of VANET nodes, limited communication time, and high computing loads, we consider using a permissioned blockchain system in our proposed BMEC-FV framework. Specifically, the significance of the BMEC-FV architecture is the secure data storage and sharing in vehicular edge networks. In general, distinct features of the proposed BMEC-FV are as follows:

- We design an integrated BMEC-FV framework that can enable dynamic orchestration of secure connected vehicles' communication. This architecture applies computing offloading to adaptively calculate the trust value of each vehicle. Since the BMEC-FV system is hierarchical and heterogeneous, we design a consensus mechanism based on redundant Byzantine fault tolerance (RBFT).

- In this framework, a trust model is designed to evaluate parent nodes' reliability

and link quality in an integrated manner. In order to ensure reliable communication between vehicles in a VANET environment, a node reliability estimation is designed to examine the parent (next hop to the sink) nodes' forwarding behavior.

- We propose a blockchain-based consensus protocol that provides interaction between the distributed MEC servers and the blockchain system. A detailed and theoretical analysis of the consensus protocol is presented. In the BMEC-FV system, the blockchain serves as a platform for computing tasks and data sharing. In particular, computing tasks (i.e., trustworthiness calculations in the Internet of Vehicles) are self-organized by smart contracts in the blockchain. We present a detailed derivation of the throughput of the blockchain system.

- By comprehensively considering the computational tasks in the smart contract and the computational maintenance in blockchain consensus mechanism, we formulate the vehicle trustworthiness calculation, the block size, the number of generated blocks through the block producers, and the primary node performance as a joint optimization problem. Due to the computational complexity of this optimized problem, we use the Markov decision process by defining state spaces, action spaces, and reward functions. Our goal is to optimize the throughput and QoS of the entire BMEC-FV system.

- In order to solve the high dynamics of the BMEC-FV system and the time-varying nature of the system state, we use the technique of a deep compression neural network to solve this joint optimization problem. Specifically, the BMEC-FV system uses dueling deep Q-learning to obtain the optimal strategy. The deep convolutional neural network in the BMEC-FV system uses the pruning compression method to accelerate the convergence of the system.

- Simulation results show the effectiveness of the proposed scheme with different parameters by comparing with other baseline methods.

1.4.1 List of Publication

- Dajun Zhang, F. Richard Yu, Zhexiong Wei and Azzedine Boukerche, "Software-defined Vehicular Ad Hoc Networks with Trust Management," in *Proc. ACM DIVANet'16*, Malta, Malta, Nov. 2016.

- Dajun Zhang, F. Richard Yu, and Ruizhe Yang, "A Machine Learning Approach for Software-Defined Vehicular Ad Hoc Networks with Trust Management," in *Proc. IEEE Globecom'18*, Abu Dhabi, UAE, Dec. 2018.

- Dajun Zhang, F. Richard Yu, and Ruizhe Yang and Helen Tang, "A Deep Reinforcement Learning-based Trust Management Scheme for Software-defined Vehicular Networks," in *Proc. ACM DIVANet'18*, Montreal, Canada, Nov. 2018.

- Dajun Zhang, F. Richard Yu, Zhexiong Wei and Azzedine Boukerche, "Trust-based Secure Routing in Software-defined Vehicular Ad Hoc Networks," *Springer Encyclopedia of Wireless Networks*, DOI : $https$: $//doi.org/10.1007/978-3-319-32903-1_126-1$, Aug. 2018.

- Dajun Zhang, F. Richard Yu, Zhexiong Wei and Azzedine Boukerche, "Trust-based Secure Routing in Software-defined Vehicular Ad-Hoc Networks", *arXiv preprint arXiv:1611.04012*, Nov, 2016.

- Dajun Zhang, F. Richard Yu, and Ruizhe Yang, "Blockchain-Based Distributed Software-defined Vehicular Networks: A Dueling Deep Q-Learning Approach,"

IEEE Trans. Cognitive Comm. and Net., vol. 5, no. 4, pp. 1086-1100, Dec. 2019.

- Dajun Zhang, F. Richard Yu, and Ruizhe Yang, "Software-Defined Vehicular Networks with Trust Management: A Deep Reinforcement Learning Approach," *IEEE Trans. Intell. Transp. Sys.*, submitted, 2019.

- Dajun Zhang, F. Richard Yu, and Ruizhe Yang, "Blockchain-Based Mobile Edge Computing for Future Vehicular Networks: A Deep Compressed Neural Network Approach," *IEEE Trans. Wireless Comm.*, submitted, 2020.

Chapter 2

Software-Defined Vehicular Networks with Trust Management: A Deep Reinforcement Learning Approach

The appropriate design of a vehicular ad hoc network (VANET) has become a pivotal way to build an efficient smart transportation system, which enables various applications associated with traffic safety and highly-efficient transportation. VANETs are vulnerable to the threat of malicious nodes stemming from its dynamicity and infrastructure-less nature and causing performance degradation. Recently, software-defined networking (SDN) has provided a feasible way to manage VANETs dynamically. In this chapter, we propose a novel software-defined trust-based VANET architecture (SD-TDQL) in which the centralized SDN controller is served as a learning agent to get the optimal communication link policy using a deep Q-learning approach. Specifically, we use the expected transmission count (ETX) as a metric to evaluate the quality of the communication link for the connected vehicles' communication. Moreover, we design a trust model to avoid the bad influence of malicious vehicles. Simulation results prove that the proposed SD-TDQL framework enhances the link quality.

2.1 System Model

In this section, we introduce our proposed SD-TDQL framework for VANETs. Each node in our model aims to find a secure communication link for the communication of connected vehicles under the bad influence of malicious nodes. Firstly, we introduce the trust model of VANETs and the architecture of SD-TDQL, and then illustrate the method of link quality estimation.

2.1.1 The Vehicles' Trust Derivation

The trust evaluation is used to estimate the misbehavior of each vehicle in VANETs. In our model, the trust of each vehicle is derived from its neighbor's packet forwarding ratio [63]. In our trust model, the trust value of all vehicles in a link is calculated by a method of linear aggregation. Trust application including the estimation of link quality is described in this section.

We consider that there are $\mathcal{N} = \{1, 2, ..., n, ...N\}$ vehicles in a VANET environment and these vehicles contain their trust information to establish a connection with their neighbors. The malicious vehicles will deeply degrade the performance of the proposed framework, so each vehicle needs to choose a reliable neighbor to establish a high-quality communication link. Fig. 2.1 shows our proposed SD-TDQL framework. In this figure, if vehicle N_5 is a malicious node, vehicle N_1 will select another vehicle to establish the routing path.

For many proposed trust model in VANETs, direct trust and indirect trust are two important factors. Direct trust is obtained as the first-hand trust information from neighbor vehicles. The indirect trust such as recommend trust is the second-hand trust information, which is obtained from a third party. However, since the indirect trust may lead to additional communication cost for the trust exchange [63], we only consider the direct trust value of each vehicle in order to simplify our trust model.

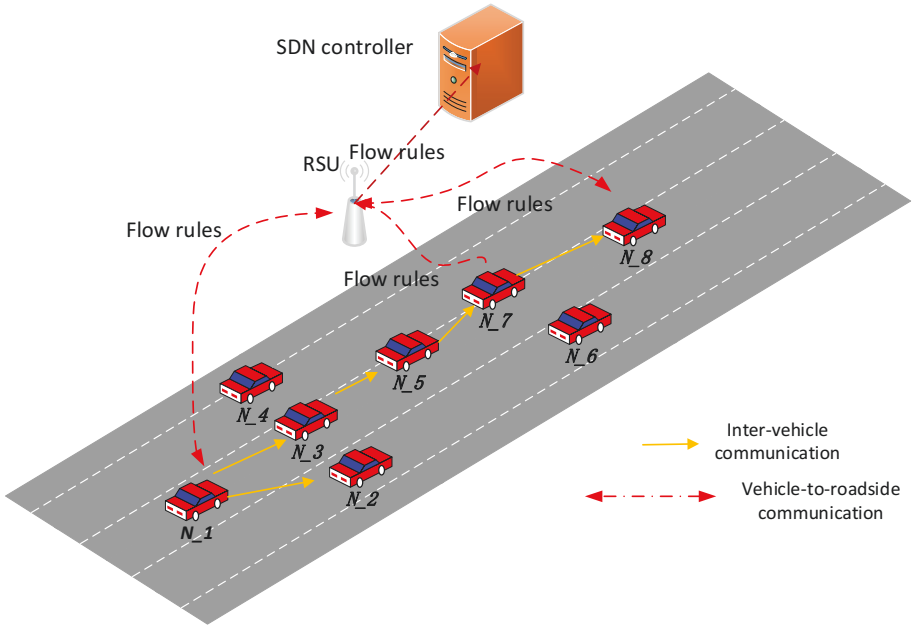

Figure 2.1: An eight-vehicle VANET topology.

We consider a circumstance in which, if the distance between a source vehicle and a destination vehicle is beyond one hop, the data packets will be dropped randomly because of the heavy traffic on a data link. Correct forwarding means that the number of packets received by the neighbor vehicles is accurate. For example, if a malicious vehicle forwards some data packets by tampering with data, these packets are not considered as correct forwarding. Here, we need to specifically point out that because the packet sequence number of the tampered data packet is different from the correct data packet before tampering, the neighbor node will consider it as untrusted data after receiving this data packet. According to sequence number, a packet is marked as either received or lost in the current time window. Therefore, this neighbor node will reduce the reliability of the sending node. In summary, we treat maliciously tampered packets as incorrectly forwarded.

In SD-TDQL, the control packets and the data packets are two important components for vehicles to build communication with each other. Specifically, the data

transfer path is decided by the control packets in the path discovery process. The forwarding ratios of the two components determine the trust level of the vehicles in the VANET environment. Hence, the trust definition of each vehicle is shown below:

$$T_{nn'}(t) = \omega_1 T^C_{nn'}(t) + \omega_2 T^D_{nn'}(t) \tag{2.1}$$

where $T^C_{nn'}(t)$ denotes the direct trust of the control packets and $T^D_{nn'}(t)$ denotes the direct trust of the data packets. The direct trust of vehicle n' for vehicle n is represented by $T_{nn'}(t)$. ω_1 and ω_2 are two weighted factors ($\omega_1, \omega_2 \geq 0$, and $\omega_1 + \omega_2 = 1$) that determine the importance of the two components ($T^C_{nn'}(t)$ and $T^D_{nn'}(t)$). In our proposed framework, we assume $\omega_1 = \omega_2 = 0.5$, which means that the control packets' forwarding ratio and data packets' forwarding ratio are all considered to determine the trust value of each vehicle in our proposed framework.

The trust value of $T^C_{nn'}(t)$ is determined by the control packet switching between two neighbor nodes in a VANET communication link. The trust computation for the control packets can be defined as:

$$T^C_{nn'}(t) = \frac{c^C_{nn'}(t)}{f^C_{nn'}(t)} \quad t \leq W \tag{2.2}$$

where $f^C_{nn'}(t)$ represents the number of control packets that should have been sent from vehicle n to vehicle n', and $c^C_{nn'}(t)$ denotes the number of control packets that a vehicle correctly forwards to its neighbor node n' at time slot t, where W represents the width of the recent time window.

Similarly, the trust computation for the data packets is:

$$T^D_{nn'}(t) = \frac{c^D_{nn'}(t)}{f^D_{nn'}(t)} \quad t \leq W \tag{2.3}$$

Table 2.1: Trust level of vehicles

Trust Level	Trust Value	Explanation
1	[0, ξ]	Malicious vehicle
2	(ξ, 0.5]	Suspect vehicle
3	(0.5, 0.85]	Less trustworthy vehicle
4	(0.85, 1]	Trustworthy vehicle

where $f^D_{nn'}(t)$ represents the number of data packets that should have been sent from vehicle n to vehicle n', and $c^D_{nn'}(t)$ denotes the number of data packets that a vehicle correctly forwards to its neighbor node n' at time slot t.

After each node exchanges its information, node n' checks whether neighbor n forwarded the data packet correctly. If so, the trust value $T_{nn'}(t)$ increases at time slot t. Otherwise, $T_{nn'}(t)$ decreases. In our proposed trust model, the trust value is limited to a continuous range from 0 to 1 (that is, $0 \leq T_{nn'}(t) \leq 1$). A trust value of 0 indicates complete distrust, while a trust value of 1 indicates absolute trustworthiness. Table 2.1 lists examples of VANET node trust levels in SD-TDQL. If there is no information interaction between the two nodes, the initial trust value will be set to 0.5 (the node with lower trust). In other words, we take a limited optimistic view of unknown connected vehicle nodes. Threshold ξ (called the blacklist trust threshold) is used to detect malicious nodes. In other words, if a node's trust value is less than ξ, it will be considered a malicious node. In particular, it should be pointed out that in SD-TDQL, the mutual transmission of trust information between VANET nodes is done through the vehicle terminal. In this thesis, the vehicle-mounted terminal refers to a communication module with integrated calculation, storage, and input and output, which can provide vehicle SDN controller with trust information of the vehicle and electronic equipment that controls the vehicle.

Each node rewards the benevolent behavior of the cooperating nodes and punishes the malicious behavior of malicious nodes based on their personal experience. In order

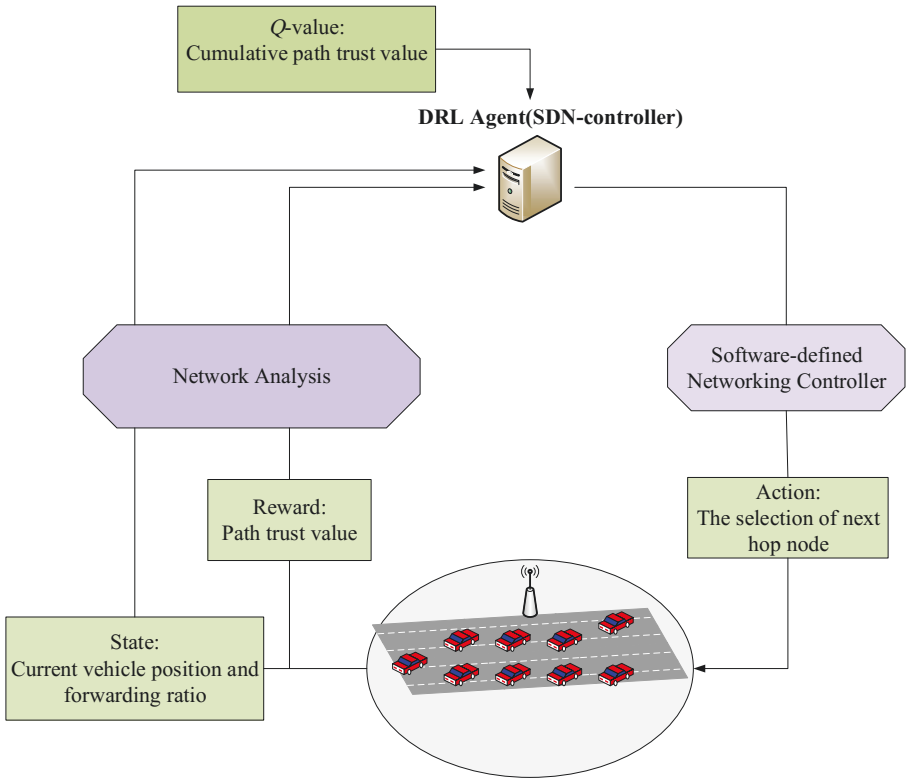

Figure 2.2: SD-TDQL interaction process.

to minimize the risk of packet transmission failure, nodes should interact with trusted nodes whose trust value is above the threshold.

The sender puts itself in promiscuous mode [64] after sending any data packet to listen for retransmissions from the forwarding node. Using this method, the node can know whether the data packets that have been sent to the neighbors are actually forwarded. Direct trust values can also be shared between neighbors using higher layers (such as the Reputation Exchange Protocol [65]).

2.1.2 The Architecture of SD-TDQL

According to [66], several modules are deployed in our proposed SD-TDQL framework. As shown in Fig. 2.3, the most important modules in SD-TDQL include a vehicles' trust information module, a storage module, a transaction management module, and a learning module. These modules are responsible for information processing in different layer respectively. The vehicles' trust information module is used to collect the trust information from each vehicle in the device layer. The storage module can be utilized to store each vehicle and physical link information in the device layer. The storage module aims to collect the global network link and vehicle information coming from the device layer. The learning module aims to make some decisions, i.e., which vehicle can be selected as the trusted neighbor.

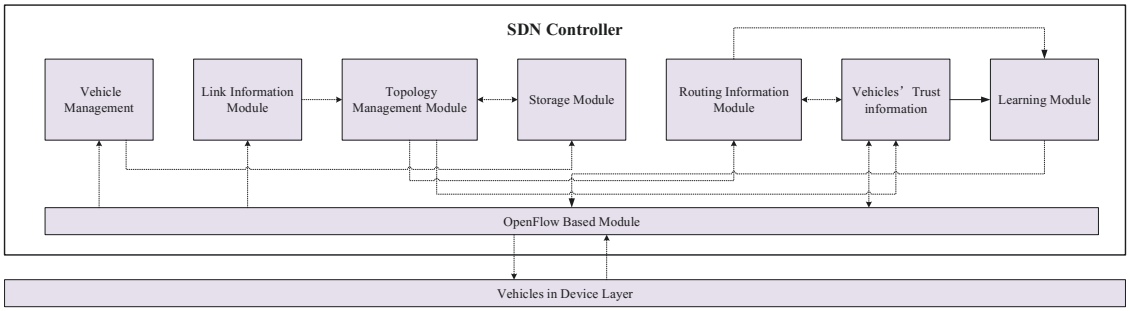

Figure 2.3: The modules in the SDN controller.

OpenFlow Based Module: OpenFlow [67] is a key mechanism in SDN technology, which is used for our proposed framework southbound interface. This module is used to receive *Packet-In* messages coming from the device layer, and to provide an interface for other modules in the SDN controllers.

Vehicle Management: This module is used to collect the device layer information sent by vehicles in the device layer. Each vehicle transfers a *Packet-In* message to the SDN controller, then this module decapsulates the coming message, and gets the vehicle information in the device layer.

Link information Module: This module collects the link information coming from the device layer. The module aims to collect the information of data packets coming from the communication of connected vehicles. Specifically, when the data packets reach from one vehicle to its neighbor, the packets are encapsulated into *Packet-In* message and transferred them to the control layer, and then the controller decapsulates the data packets and gets the link information.

Topology Management Module: This module is used to manage the topology information of vehicles received from the link information module. The responsible of this module is to add, update and delete topology information.

Routing Information Module: The routing information module is used to collect routing information that is decided by the communications of connected vehicles in the device layer. The routing information will be encapsulated and sent to the SDN controller through the OpenFlow based module as shown in Fig. 2.3.

Vehicles' Trust Information Module: This module is used to collect the trust information from each area of the device layer. The trust definition is introduced in this section. After finishing to collect those value, the SDN controller will send those trust information to the learning module for the further ETX delay calculation.

Storage Module: The storage module can be utilized to store each vehicle and physical link information in the device layer. Meanwhile, it also needs to store the deep neural network parameters from the learning module.

Learning Module: The learning module using a deep Q-learning approach aims to optimize the ETX delay of SD-TDQL. The vehicles' trust information module receives the trust information from the device management module and area topology management module. The learning module interacts with the device layer and aims to make some decisions, i.e., which vehicle can be selected as the trusted neighbor.

2.1.3 Link Quality Estimation Using Expected Transmission Count

According to [68], the *expected transmission count* (ETX) of a link is the number of data transmissions and retransmissions required to send a packet over the link. The calculation method of ETX for a route uses forwarding and reverse delivery ratios. Because of the signal attenuation and fading of wireless links, we endeavor to make the forwarding ratio more reliable (the trust of each vehicle) and to minimize the packet retransmissions or losses. In SD-TDQL, the ETX of a route is estimated by the interaction between the current forwarding vehicle and its neighbors. The forwarding ratio of each vehicle, $T_{nn'}(t)$ is the trustworthiness of each vehicle at time slot t; the reverse delivery ratio, $T^r_{nn'}(t)$ is the probability of receiving an acknowledgement (ACK).

For calculating the ETX between current forwarding vehicle n and its candidate neighbor $n^{'}$, vehicle $n^{'}$ calculates the trust value and sends it to vehicle n, which calculates the reverse delivery ratio $T^r_{nn'}(t)$. In SD-TDQL, each forwarding vehicle estimates the $T^r_{nn'}(t)$ using $Hello$ packets delivery ratios. Hence, $T^r_{nn'}(t)$ is defined as [69]:

$$T^r_{nn'}(t) = \frac{count(t-w,t)}{w/\tau} \tag{2.4}$$

where $count(t-w,t)$ is the number of $Hello$ packets received by the vehicle n so far during the time window w. w/τ is the number of $Hello$ packets that should have been received at that time. The reason why the vehicle n can detect the failed transmission of data packets is that the number of lost and successfully received data packets by vehicle n' mainly depends on the packet sequence number of each transmitted $Hello$ packets.

Thus, the ETX can be defined as:

$$ETX_{SD-TDQL} = \frac{1}{T_{nn'}(t) \times T^{r}_{nn'}(t)} \tag{2.5}$$

2.2 Problem Formulation

We have presented the link estimation method of the SD-TDQL. In order to optimize the ETX, we need to jointly consider the trust feature and reverse delivery ratio of each vehicle. In this section, we propose to use a Markov decision process to address this joint problem by defining system state space $\boldsymbol{S}(t)$, action space $\boldsymbol{A}(t)$, and reward function $r_V^{ETX}(t)$.

2.2.1 System State

State: The SDN controller (learning agent) interacts with the device layer to sense the system state space that includes vehicle trust information and vehicle reverse delivery ratio. We collect the trust information of each vehicle and reverse delivery ratio of each vehicle into two random variables: $\gamma_n(t)$ and $\delta_n(t)$. In order to enhance the communication link quality, the SDN controller interacts with the network environment to collect the system state space $\boldsymbol{S}(t)$ at time slot t. Accordingly, the trust feature of each vehicle and reverse delivery information are collected by the SDN controller. Hence, the system state space is defined as the following matrix:

$$\boldsymbol{S}(t) = \begin{bmatrix} \gamma_1(t) & \gamma_2(t) & \gamma_3(t) & \cdots & \gamma_n(t) & \cdots \gamma_N(t) \\ \delta_1(t) & \delta_2(t) & \delta_3(t) & \cdots & \delta_n(t) & \cdots \delta_N(t) \end{bmatrix} \tag{2.6}$$

Specifically, the trust feature and reverse delivery ratio of vehicles in SD-TDQL always keep changing because of the channel instability. In our proposed framework,

there are $M = \{1, ..., M\}$ states for every node's trust feature. Let $p_{\gamma_n}^{mm'} = p\{\gamma_n(t+1) = \gamma_n^{m'} | \gamma_n(t) = \gamma_n^m\}$, $m, m' = 1, ..., M$ represent the transition probability of the trust feature changing the state from m (current time slot is t) to m' (next time slot $t+1$). Hence, the state transition probability of p_{γ_n} is set to be:

$$\boldsymbol{p}_{\gamma_n}(t) = \begin{bmatrix} p_{\gamma_1}^{12}(t) & p_{\gamma_1}^{13}(t) & p_{\gamma_1}^{14}(t) & \cdots & p_{\gamma_1}^{1M}(t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{\gamma_N}^{12}(t) & p_{\gamma_N}^{13}(t) & p_{\gamma_N}^{14}(t) & \cdots & p_{\gamma_N}^{1M}(t) \end{bmatrix} \tag{2.7}$$

Meanwhile, there are $K = \{1, ..., K\}$ states for every node's reverse delivery ratio. Therefore, the state transition probability of p_{δ_n} is set to be:

$$\boldsymbol{p}_{\delta_n}(t) = \begin{bmatrix} p_{\delta_1}^{12}(t) & p_{\delta_1}^{13}(t) & p_{\delta_1}^{14}(t) & \cdots & p_{\delta_1}^{1K}(t) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_{\delta_N}^{12}(t) & p_{\delta_N}^{13}(t) & p_{\delta_N}^{14}(t) & \cdots & p_{\delta_N}^{1K}(t) \end{bmatrix} \tag{2.8}$$

2.2.2 Action Space

Action: The learning agent in the control layer needs to decide which vehicle is a reliable node that can be selected as a trusted neighbor. Therefore, the system action space is:

$$\boldsymbol{A}(t) = \left\{A_\gamma^N(t), A_\delta^N(t)\right\} \tag{2.9}$$

where $A_\gamma^N(t)$ and $A_\delta^N(t)$ represent:

1) $A_\gamma^N(t) = \{a_\gamma^1(t), a_\gamma^2(t), ..., a_\gamma^n(t), ..., a_\gamma^N(t)\}$, which means that whether vehicle n is the reliable neighbor. Meanwhile, the value of $a_\gamma^n(t)$ is $a_\gamma^n(t) \in \{0, 1\}$, where

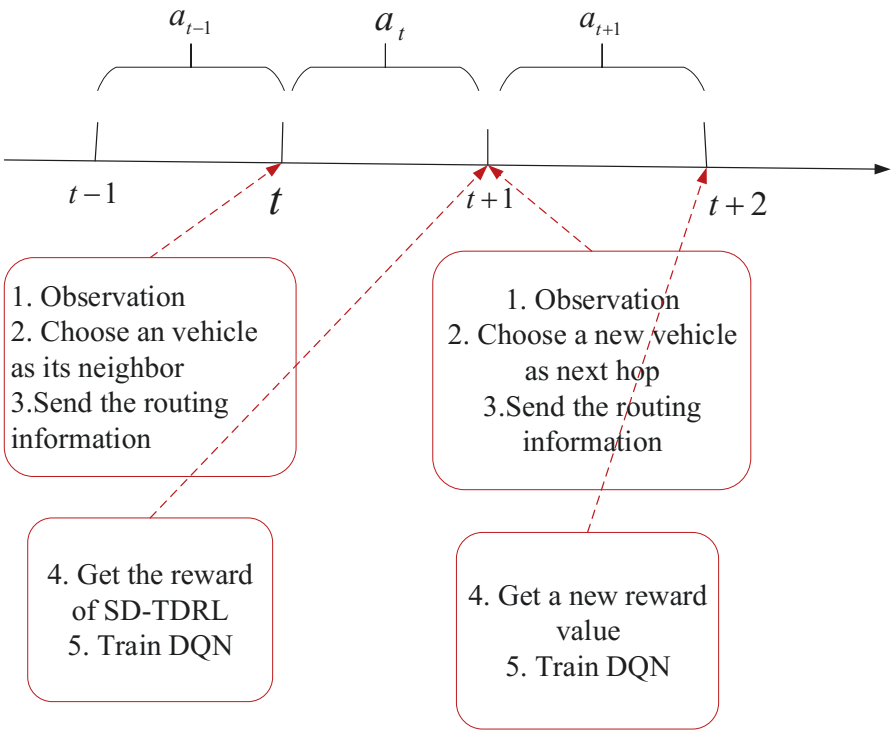

Figure 2.4: Time line for agent events: observation, state, action, reward, and training the DQN.

$a_\gamma^n(t) = 1$ denotes the vehicle n is the most trusted neighbor vehicle, otherwise it is not the trusted neighbor.

2) $A_\delta^N(t) = \{a_\delta^1(t), a_\delta^2(t), ..., a_\delta^n(t), ..., a_\delta^N(t)\}$, which means that whether vehicle n receives the $Hello$ packets. For example, if $a_\delta^n(t) = 1$, it means that vehicle n has received the ACK packets. Otherwise, the ACK message has failed to transmit to vehicle n.

2.2.3 Reward Function

Reward: In SD-TDQL, the reward is decided by the trust value of each vehicle and reverse delivery ratio in the VANET environment. In the trust model, the

trustworthiness is updated by each vehicle interacting with its neighbors in a predetermined time interval $[t,\ t+\zeta_t]$. Meanwhile, a threshold of ξ is needed to evaluate each vehicle's trust level. Each trusted vehicle is defined as $T_{nn'}(t) \geq \xi$. Otherwise, the vehicle is defined as a malicious node. Specifically, the domain of definition for each node's direct trust is [0,1].

In our proposed SD-TDQL, the reward function is used to evaluate the quality of the action performed by the SDN controller at a certain time step t. The reward function is decided by ETX delay for the communication link quality. The cumulative reward value can be represented by a serious of real numbers.

When a vehicle aims to establish a communication link with its neighbor node, the ETX delay of the communication link is calculated by the rust value and the reverse delivery ratio along the path. We assume that there is a communication link V, consisting of L vehicles, denoting as $\mathcal{L} = \{1, ..., l, ..., L\}$ and $\mathcal{L} \in \mathcal{N}$, where node l represents the $l-th$ vehicle in the communication link. So, we define the ETX as the reward for link V as:

$$r_V^{ETX}(t) = \sum_{l=1}^{L-2} \frac{1}{T_{l,l+1}(t) \times T^r_{l,l+1}(t)} \tag{2.10}$$

where $T_{l,l+1}(t) = \prod_{l=1}^{l-2} \gamma_l(t) a^l_\gamma(t)$ and $T^r_{l,l+1}(t) = \sum_{l=1}^{l-2} \delta_l(t) a^l_\delta(t)$.

In route V, the learning agent gets $r_V^{ETX}(t)$ (immediate reward) in time slot t. The SDN controller aims to optimize the long term ETX delay. Hence, the cumulative reward of ETX delay can be written as:

$$r^{long}_{ETX} = \max E\left[\sum_{t=0}^{t=T-1} \beta^t r_V^{ETX}(t)\right] \tag{2.11}$$

In SD-TDQL, a threshold β^t can be set for terminating the process.

Some challenges are presented to illustrate the difficulties for the above problem

formulation.

1) In SD-TDQL, the agent chooses an action only related to the current time slot. For example, when the agent chooses a vehicle as a trusted neighbor, its trust feature has a possibility to change another trust value according to the transition probability in the next time slot. However, the selected action may not be changed because of the threshold of ξ. Hence, the traditional optimization methods that consider the relationship between state space and action space are not suitable.

2) Considering the trust feature and reverse delivery ratio of each vehicle, the proposed SD-TDQL is high-dimensional and high-dynamical because of the transition probabilities. The traditional methods are unable to address this optimization problem.

3) In our proposed scheme, the agent aims to optimize the long-term ETX delay by step-and-step control. The SDN controller collects the state from the VANET environment at time t, and the next state at time slot $t+1$ will be influenced by the current state. Hence, the traditional methods, only considering the current state, are not suitable for this joint optimization problem.

Therefore, we consider using a deep Q-learning approach to get the optimal ETX delay policy in the next section.

2.3 Deep Q-Learning Method Used in SD-TDQL

In this section, we introduce the deep Q-learning method to find the optimal communication link quality policy π^*. The experience replay and target Q-network are designed to reduce the relevance of data.

2.3.1 Q-Learning

In the Q-learning model, there are usually two entities: an agent and an environment. The interaction between the two entities is as follows: under a state $s(t)$ of the environment, the agent takes the action $a(t)$, gets the reward $r(t)$ and enters the next state $s(t+1)$. The Q-learning method usually has some characteristics like 1) different actions produce different rewards; 2) the reward has retardance; 3) the reward of action is based on the current state. The core of Q-learning is the Q-table, in which the rows and columns of the Q-table represent the values of the states and the actions, respectively. The Q-learning model enables the agent to obtain an optimal policy π, which maps with the system state space and action space, to maximize the long-term reward.

According to [70], state-value function $V^{\pi}(s)$ and action-value function $Q^{\pi}(s,a)$ are two main components of the Q-learning method. Specifically, $V^{\pi}(s)$ measures the expected total reward that can be obtained under the current policy in the current state s:

$$V^{\pi}(s) = E^{\pi}\left[\sum_{k=1}^{\infty} \lambda^{k} r_{t+k+1} | s_t = s\right] \tag{2.12}$$

where r_{t+k+1} denotes the immediate reward at time slot $t+k+1$, and λ^{k} is the discount factor. Here, the discount factor means that the current feedback is more important than historical feedback.

Meanwhile, the action-value function $Q^{\pi}(s,a)$ indicates the expected total reward obtained after the action a is performed in the current state s:

$$Q^{\pi}(s,a) = E^{\pi}\left[r_{t+1} + \lambda Q(s_{t+1}, a_{t+1}) | s_t = s, a_t = a\right] \tag{2.13}$$

The updating method for $Q^{\pi}(s,a)$ is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(r + \lambda \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s,a) \right) \tag{2.14}$$

where α is the learning rate, and $\alpha \in (0,1]$. In fact, each $Q(s,a)$ is put into the Q-table simultaneously.

2.3.2 Deep *Q*-Learning

There is a problem with Q-learning. The states in the real situation may be infinite so that the Q-table will be infinite simultaneously. Meanwhile, Q-learning still has some instability, and the causes are described in [48], thus we can use a deep neural network (DNN) to represent the Q-value. The emergence of deep learning provides a method to solve the challenges of traditional Q-learning. The core idea of deep Q-learning is that the deep neural network enables the agent to get the low-dimensional features from high-dimensional input data by using weights and bias of the deep neural network. This feature of the DNN attracts many researchers to approximate $Q(s,a)$ by using DNN instead of the traditional Q-table. For example, $Q(s,a)$ can be approximated to a Q-value $Q(s,a,\omega)$, where ω represents the weights and biases in DNN.

Meanwhile, DQL has two important improvements in order to solve the correlation between states, which will cause the instability of Q-learning, including *experience replay* and *target Q-network* [49]. The experience replay mechanism stores the data in a replay memory, and each sample is a quad. During the training, the stored samples are randomly sampled by the experience replay mechanism, which can remove the correlation between the samples to a certain extent, thereby facilitating the convergence speed of DNN. Moreover, deep Q-learning makes another improvement, which is the *target Q-network*. That is, at the initial time step, the parameters of the

evaluation network are assigned to *target Q-network*, and then the evaluation network continues to be trained in order to update the parameters, while the parameters of *target Q-network* are fixed. After a period of training, the agent assigns the parameters of the evaluation network to *target Q-network*. These two features enable the training process to be more stable than the traditional Q-learning.

In DQL, the evaluation network is designed to minimize loss function $L(\omega)$, which can be depicted as:

$$L(\omega) = \left[r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}, \omega^-) - Q(s_t, a_t, \omega) \right]^2 \tag{2.15}$$

where ω^- is the weights and biases in the target Q-network, and ω is the weights and biases in the evaluated Q-network.

2.3.3 Deep *Q*-Learning Used in SD-TDQL

In this chapter, we apply deep Q-learning for the following reasons: 1) In SD-TDQL, the state of the connected vehicle nodes dynamically changes with time. Therefore, the traditional static-based optimization algorithm is not suitable for the scenario proposed in this chapter. Deep Q-learning based on Markov decision process can very effectively describe the dynamic change of the status of the connected car nodes; 2) Because the status of the connected car nodes changes with time, the dimension of the input data is high, and traditional optimization algorithms are difficult to handle high Input data for the dimension. However, deep Q-learning can automatically learn the abstract representation of large-scale input data, and use this representation as the basis for self-motivated RL to optimize the problem-solving strategy.

In this chapter, the SDN controller aims to get an optimal ETX delay policy for the secure communication link from the source vehicle to the destination vehicle.

In general, each vehicle has some set of actions that can be chosen from. As we described before, three-tuple $\{\boldsymbol{S}(t), \boldsymbol{A}(t), r_{ETX}^{long}\}$ has already been defined for the system's states, actions, and rewards. For example, the deep Q-network feeds back the optimal action $\arg\max Q_{\pi}(\boldsymbol{S}(t), \boldsymbol{A}(t))$ to the sender. The SDN controller will get a state at next time slot after it executes a selected action, and the controller can obtain the reward value according to the reward function as we described in the previous section.

The reward value is an important factor for the proposed SD-TDQL since it is directly related to the system state space and action space. Relatively speaking, the lower the value of the system state space, the lower the probability that the corresponding action is selected.

In order to remove correlations inside of the deep Q-network, each agent's state-action pair of each time slot is stored in the experience replay memory D. In the Q-network, the agent randomly samples from D, and the parameter ω is updated at every time instant. The protocol samples experiences from the memory D and applies the following loss function to update:

$$L(\omega) = E\left[Q_{target}(\boldsymbol{S}(t)) - Q(\boldsymbol{S}(t), \boldsymbol{A}(t), \omega)\right] \tag{2.16}$$

Meanwhile, for the target Q-value $y = r_V^{ETX}(t) + \gamma \max_{\boldsymbol{A}_{(t+1)}} Q(\boldsymbol{S}(t+1), \boldsymbol{A}(t+1), \omega^{-})$, evaluation network assigns the coefficient ω to the target Q-network as the coefficient ω^{-} at every time instants. The ϵ-greedy mechanism is used to combine the exploration and exploitation procedure, where ϵ is usually a small value, as the probability of random action. We can change the value of ϵ to get different exploration and production ratios. Algorithm 1 shows the training and updating procedure for SD-TDQL framework.

In SD-TDQL, the optimal policy is achieved by the link quality estimation using

Algorithm 1 Deep Q-learning approach for SD-TDQL framework

Initialization
  Initialize replay memory D
  Initialize the Q-network with initial weights ω
  Initialize the target $\hat{Q}$-network with weight ω^{-}
For episode j = *1, 2, 3,...,J* **do**:
  Initialize the beginning state $\boldsymbol{S}(t)$
  Initialize the action set $\boldsymbol{A}(t)$
  Initialize the time step t
  For t = *1,2,3,...,T* **do**
    Choose a random probability p
      If $(p \geq \epsilon)$,
        $a^{*}(t) = \arg\max Q(\boldsymbol{S}(t), \boldsymbol{A}(t), \omega)$,
      Otherwise,
        Randomly choose an action $\boldsymbol{A}(t) \neq a^{*}(t)$,
    A vehicle forwards packets to its neighbors
      If (not a duplicate control and data packets) **then**
        Gets the values of state space $\boldsymbol{S}(t)$
          If $(T_{nn'}(t) \geq \xi)$ **then**
            Computes $r_{V}^{ETX}(t)$ and updates
            $Q(\boldsymbol{S}(t+1), \boldsymbol{A}(t+1)) \leftarrow Q(\boldsymbol{S}(t), \boldsymbol{A}(t))$
            using equation 2.14
          Else
            Discards and waits new packets
          End If
      Else
        Discard the received packets.
      End If
    $t = t + 1$
    Store the experience replay
    $(\boldsymbol{S}(t), \boldsymbol{A}(t), r_{V}^{ETX}(t), \boldsymbol{S}(t+1))$ in D
    If (the selected vehicle changes to a untrusted
    vehicle according to p) **then**
      Choose another action in next time step
    Else
      Keep the selected vehicle unchanged
    End If
    Sample random mini-batch
    $(\boldsymbol{S}(t), \boldsymbol{A}(t), r_{V}^{ETX}(t), \boldsymbol{S}(t+1))$
    from D.
    Compute the value in evaluated Q-network: $y = r_{V}^{ETX}(t) + \gamma \max_{\boldsymbol{A}(t+1)} Q(\boldsymbol{S}(t+1), \boldsymbol{A}(t+1), w^{-})$.
    Perform a gradient descent step on the loss function
    $L(\omega)$ with respect to
    weighted factor ω.
    Every C steps copy weights into target network
    periodically $\omega^{-} \leftarrow \omega$, and update target deep
    networks.
  End For
End For

ETX and trust computation.

1) ***Framework entities***: The most important components for the SD-TDQL mechanism are the SDN controller, SDN-enabled vehicles, and a SDN-enabled road-side unit (RSU).

- ***SDN controller***: SDN controller is the logical central intelligence of our proposed SD-TDQL. The controller controls all of the behaviors of VANET nodes in the entire network environment. It consists of all vehicles' reverse delivery information and trust values. For each node in the VANET environment, the corresponding next-hop node is based on the selection policy that is decided by the SDN controller. As shown in Fig. 2.1, the SDN controller controls all the actions of underlying SDN-enabled vehicles and RSUs. Specifically, all the actions that each SDN-enabled vehicle performs are explicitly defined by the SDN controller, which will push down all the flow rules on how to treat the traffic. For example, If the node trust is higher than the threshold ξ (trusted neighbors), the selection of the controller is decided by applying an optimal policy to the Q-value for the trusted neighbors. Finally, the agent obtains the optimal Q-value and learns the optimal link quality policy accordingly.

- ***SDN-enabled vehicles***: The vehicles in the device layer are controlled by the agent of SD-TDQL. These vehicles receive control information from the agent to perform the selected action of the SDN controller.

- ***SDN-enabled RSU***: A stationary SDN-enabled RSU that is controlled by the learning agent is deployed along the roadside. It is used for auxiliary communication with the SDN-enabled vehicles.

2) ***Framework description***: The framework of the proposed SD-TDQL can be divided into the device layer and control layer. In the device layer, each vehicle

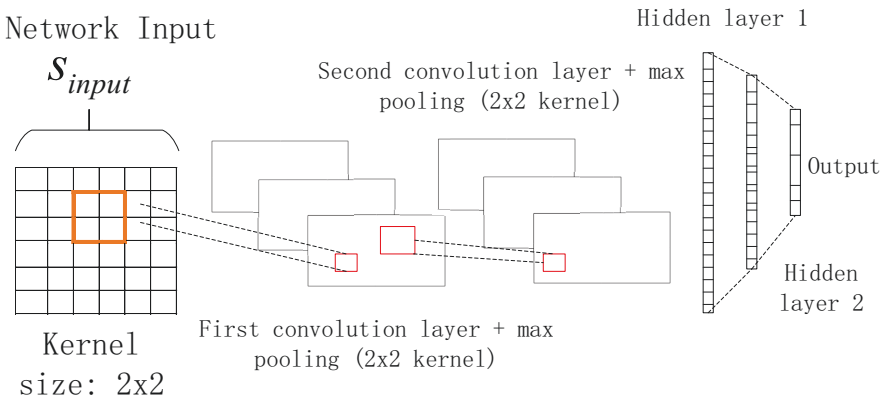

Figure 2.5: The network architecture of CNN for SD-TDQL training.

first calculates the control packet trust value and reverse delivery ratio, and then the information will be transferred to the SDN controller through the flow rules (Packet-In messages). The SDN controller stores those values in the storage module. After finishing the path discovery process, the agent aims to decide the best communication link to establish a secure data transfer path. In this phase, each vehicle calculates the trust values of data packets and reverse delivery ratio. Specifically, each node's trust value and reverse delivery ratio have the possibility of changing according to p_{γ_n} and p_{δ_n} in the data forwarding process, so the optimal link selected by the SDN controller may convert to the unreliable communication link. Therefore, a new optimal policy is needed to decide by the agent according to the new system state space and action space in the VANET environment.

DQN architecture: The DQN architecture using the convolutional neural network (CNN) is established as shown in Fig. 2.5, where the input image is a matrix of system state space and the final network output is the $Q(s\boldsymbol{S}(t), \boldsymbol{A}(t), \omega)$. S_{input} feeds to the first convolution layer that convolves the input image with 32 filters, kernel size 2×2, stride size 1 and the same padding method. The output of the first convolution layer is convolved by the second convolution layer with 48 filters, the

same stride size, and the padding method. The rectifier nonlinearity activation function (ReLU) is used as the activation function for two convolution layers. Moreover, the two max-pooling layers are connected to the first and second convolution layers. The fully connected layers that are used as two neural networks of all 512 units are designed to connect with the second max-pooling layer. Finally, the output layer of the proposed DQN outputs a final Q-value.

DQN training: The training algorithm is illustrated in Algorithm 1, and the detailed process of experience replay is shown in Fig. 2.7. In each time step t, the agent observes state-action pair $E_t = \{\boldsymbol{S}(t), \boldsymbol{A}(t), r_V^{ETX}(t), \boldsymbol{S}(t+1)\}$ from the VANET environment, and stores E_t into the replay memory $D = \{E_1, E_2, E_3, ..., E_t\}$. The learning agent randomly samples from D to form the input image, which is trained by the proposed DQN architecture. Specifically, the experience replay memory always keeps updating because the new state-action pair coming from the environment will be stored in the memory. Hence, the oldest state-action pair will be discarded because of the finite capacity of the replay memory. The SDN controller aims to obtain the optimal Q-value $Q_{\pi^*}(\boldsymbol{S}(t), \boldsymbol{A}(t))$ by training data including the input image S_{input}. Simultaneously, the target Q-value is defined as the $y = r_V^{ETX}(t) + \gamma \max_{\boldsymbol{A}_{(t+1)}} Q(\boldsymbol{S}(t+1), \boldsymbol{A}(t+1), \omega^-)$. After the agent finishes the training, it will get an optimal estimated Q-value and the best communication link quality policy.

Consequently, our proposed scheme gets the best ETX delay policy of the communication link for the connected vehicles' communication. The simulation results show the proposed framework performs better than the existing approach, and they are discussed in the following section.

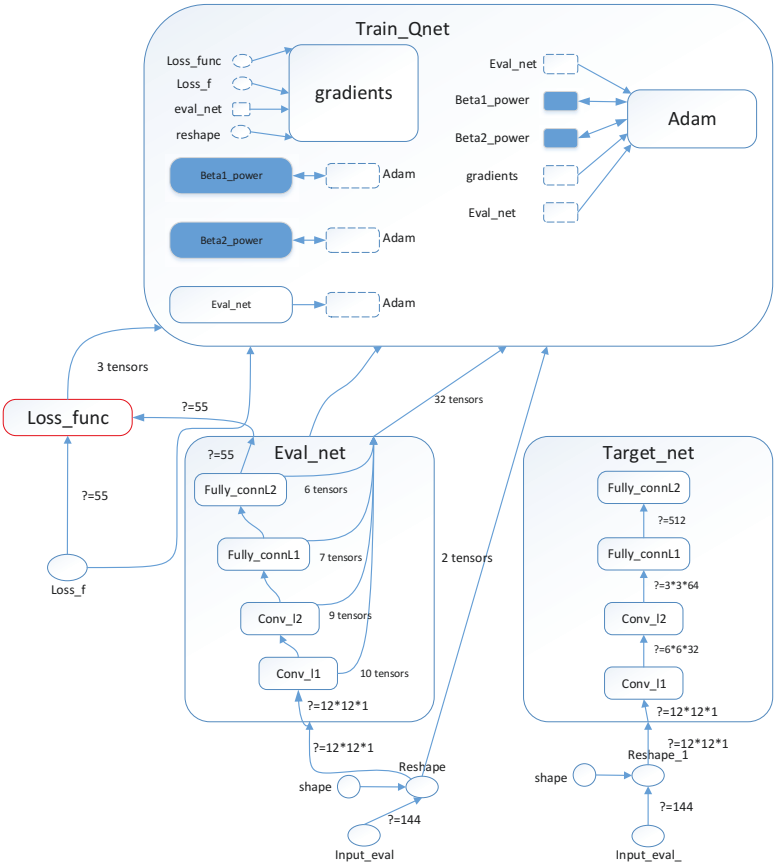

Figure 2.6: The network structure of CNN used in SD-TDQL.

2.4 Simulation Results and Discussions

Computer simulation is carried out to illustrate the performance of the proposed SD-TDQL to the optimization of ETX delay. We use TensorFlow (version 1.13.0) [71] and MATLAB in our simulation to implement the deep Q-learning. TensorFlow is a basic framework for deep learning. Meanwhile, it provides an extensive API that is related to deep reinforcement learning. In our simulation, we use multiple layers of traditional neural networks and convolutional neural networks to construct different deep network structures in order to compare training efficiency. Meanwhile, we verify SD-TDQL framework by simulation in terms of vehicle ETX delay.

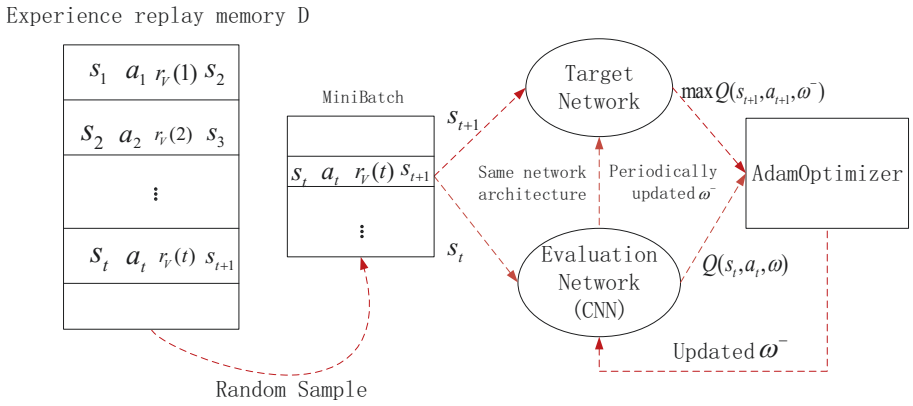

Figure 2.7: Experience replay process for SD-TDQL.

2.4.1 Simulation Setup

In our simulation, we use a computer, which has one Nvidia GPU with version GTX 1080ti. The CPU is Intel(R) Core(TM) i7-6800K with 32GB memory. The simulation tool is TensorFlow 1.13.0 with Python 3.7.2 on Ubuntu 18.04 LTS and MATLAB R2017b on Windows 10 64-bit operating system with x64-based processor.

The proposed SD-TDQL framework is compared with an existing scheme, which mainly focuses on trust-based software-defined networking for VANETs. Specifically, the concept of software-defined networking is considered in this scheme, and the scheme cannot deal with a situation in which any trusted vehicle in the device layer converts to the untrusted node. In other words, if any vehicle in the network becomes an untrusted node, the controller must restart the routing process.

In our simulations, we deploy a different number of SDN-enabled vehicles that are randomly distributed in a pre-determined VANET environment. We assume that the vehicles' states can be trusted ($T_{nn'}(t) \geq \xi$) or malicious ($T_{nn'}(t) < \xi$). Trusted vehicles can be selected as the trusted neighbor node in order to establish the best ETX delay communication link from the source to the destination. Conversely, malicious vehicles are treated as untrusted vehicles, which will badly interfere with the network

Table 2.2: Parameters used in SD-TDQL simulation

Parameter	Value
Batch size	32
Experience replay buffer size	2000
learning rate	0.00001
Discount factor	0.9
Data rates	1 Mbps, 2 Mbps, 5.5 Mbps and 11 Mbps
Total training steps	40000
Number of nodes	8, 12, 20, 24, 28 and 32

performance. In our simulation, an example of the state-transition matrix for the system state space is:

$$\boldsymbol{p} = \begin{pmatrix} 0.125 & 0.25 & 0.0625 & 0.0625 & 0.2 & 0.1 & 0.0625 & 0.1375 \\ 0.25 & 0.125 & 0.2 & 0.0625 & 0.0625 & 0.1 & 0.0625 & 0.1375 \\ 0.0625 & 0.25 & 0.125 & 0.0625 & 0.2 & 0.1 & 0.0625 & 0.1375 \\ 0.0625 & 0.25 & 0.125 & 0.0625 & 0.1 & 0.2 & 0.1375 & 0.0625 \\ 0.0625 & 0.25 & 0.0625 & 0.125 & 0.1 & 0.2 & 0.1375 & 0.0625 \\ 0.1375 & 0.25 & 0.0625 & 0.125 & 0.1 & 0.2 & 0.0625 & 0.0625 \\ 0.2 & 0.25 & 0.0625 & 0.125 & 0.1 & 0.0625 & 0.1375 & 0.0625 \\ 0.0625 & 0.1375 & 0.0625 & 0.125 & 0.1 & 0.2 & 0.25 & 0.0625 \end{pmatrix} \tag{2.17}$$

The architecture of deep Q-network used in SD-TDQL framework is a traditional neural network with 6, 8, and 10 hidden layers and CNN. We use Tensorboard to show an example of the DQN architecture using CNN shown in Fig. 2.6. In our simulation, we use AdamOptimizer to improves the traditional gradient descent by using momentum (the moving average of the parameters) and promotes hyperparametric dynamic adjustment [72]. The values of the rest of the parameters are summarized in Table 2.2.

2.4.2 Simulation Results

Fig. 2.8 shows the relationship between the training episodes and loss function under different schemes. In our simulation, the loss function $L(\omega)$ is used to estimate the degree of inconsistency between the estimated Q-value of the SD-TDQL and the

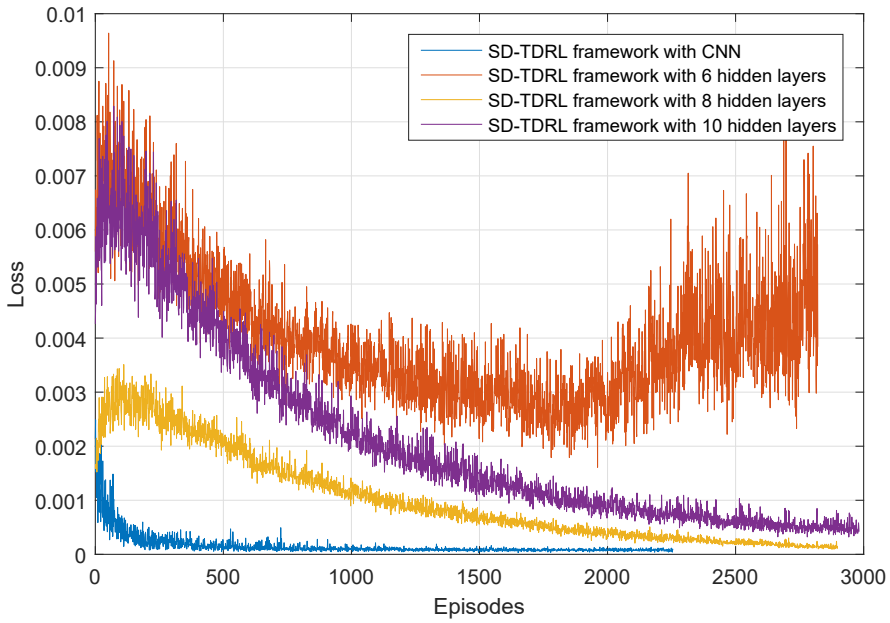

Figure 2.8: Comparison of the loss with different network architecture.

target Q-value. As shown in Fig. 2.8, we can conclude that the traditional neural network with 6 hidden layers has the highest loss compared with the other three network structures. Meanwhile, it does not converge in the pre-determined episodes. It means that the learning agent does not get the optimal communication link policy in the pre-determined episodes. As the hidden layers increase to the 8 and 10 hidden layers, the learning agent successfully gets the optimal policy because the loss function $L(\omega)$ converges in the pre-determined episodes. From this figure, we also can see that the best performance is obtained from the DQN architecture using CNN. The degree of convergence using CNN is the best compared with the other three network structures. Meanwhile, the inconsistency of the loss function using CNN is lower than the other three network structures, and it reflects that the SD-TDQL has the best robustness.

Fig. 2.9 and Fig. 2.10 show the convergence comparison using different DQN architecture with distinct learning rates. The learning rate is an important hyperparameter

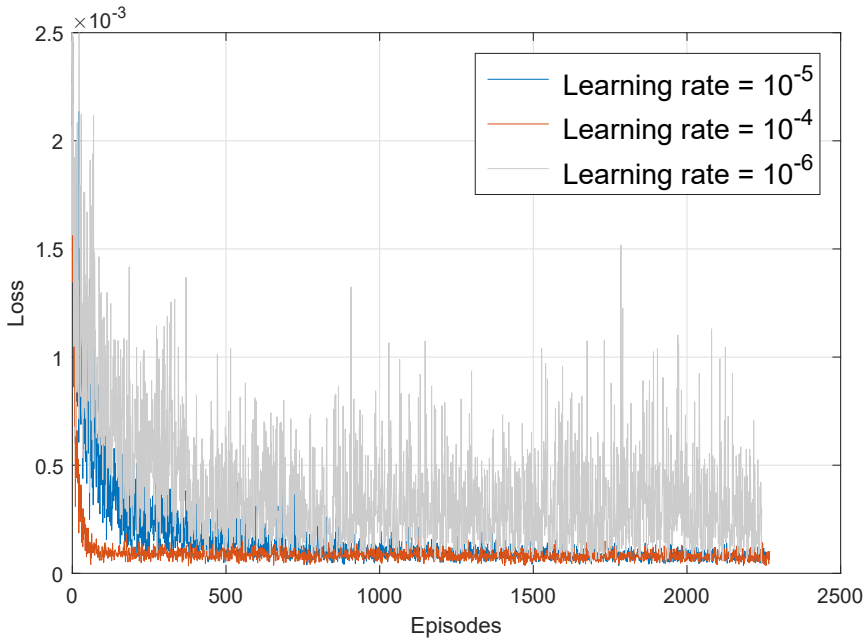

Figure 2.9: Convergence performance using CNN.

that controls the speed at which we adjust the weight of neural networks based on the loss gradient. The learning rate directly affects how fast the model can converge to a local minimum (i.e., to achieve the best accuracy). In general, the greater the learning rate, the faster the neural network learns the optimal policy. If the learning rate is too small, the network is likely to fall into the local optimum. However, if the learning rate is too large and exceeds the extreme value, the loss will stop falling and it will repeatedly oscillate at a certain position. As shown in these two figures, when the learning rate is equal to 10^{-6}, the training curve does not converge within the given episodes and accompanies by a high degree of oscillation. Through this figure, When the learning rate is equal to 0.00001, SD-TDQL has the best convergence performance. Meanwhile, we can see that the loss of SD-TDQL using CNN is much less than the DQN architecture using the normal neural network. Therefore, we choose CNN as our deep Q-network to train the SDN controller.

Fig. 2.11 shows the comparison of ETX delay of four schemes with different

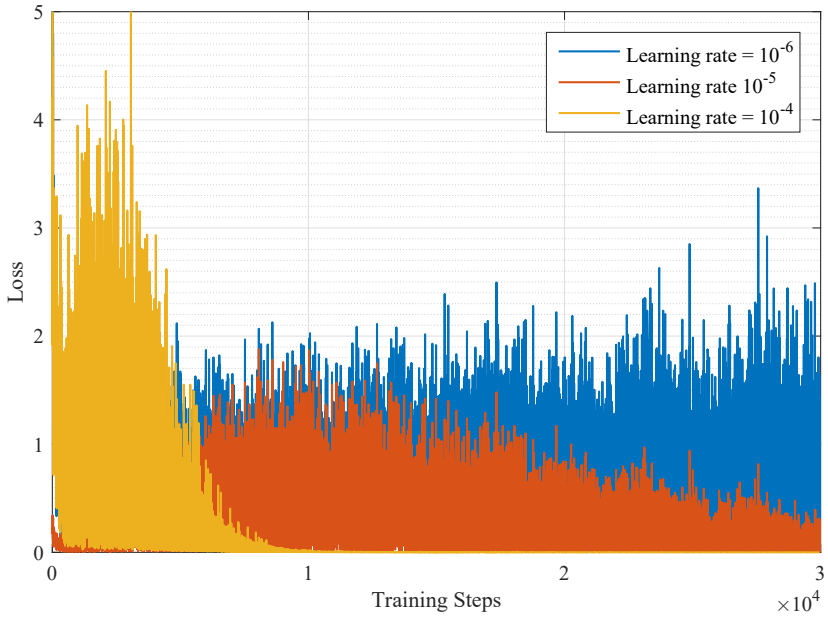

Figure 2.10: Convergence performance using normal neural network.

network architectures. From the figure, we can see that the ETX delay when using the CNN architecture has the best performance. This is because the CNN architecture has better training efficiency compared with other schemes. Meanwhile, the blue line is more stable than the other three schemes. Moreover, we can see that the agent is in the stage of continuous learning at the beginning of the training process, so the four training curves continue to rise to certain episodes. After increasing the training episodes, the reward value becomes stable, which means that the SDN controller has learned an optimal policy for ETX delay.

Fig. 2.12 shows the convergence comparison of the ETX delay using CNN architecture with different learning rates. From this figure, when learning rates are 0.00001 and 0.000001, the training curves are highly scaled, which leads it to miss the global optimum. Comparing the two curves of yellow and purple, although the yellow curve has a faster convergence speed, it is not very stable after convergence compared with the purple curve. Therefore, we choose the learning rate as $1e^{-2}$ because of its

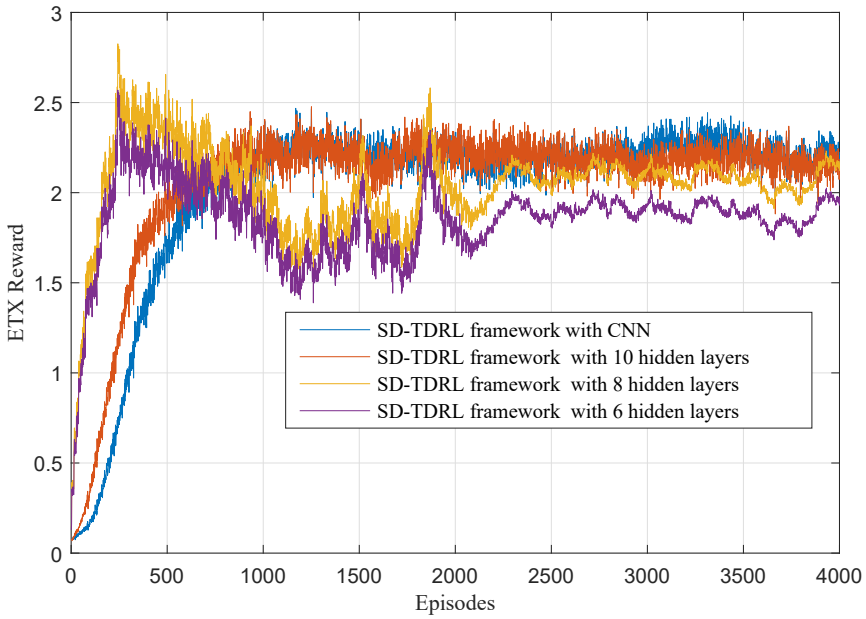

Figure 2.11: Comparison of the ETX delay with different network architecture.

acceptable convergence speed and better learning stability.

Fig. 2.13 shows the average ETX delay comparison under different data rates. As the data rate grows, we can see that the ETX delay of all schemes all increases. This is because a higher data rate increases the possibility of losing packets. Through this figure, we can conclude that the ETX delay of SD-TDQL with CNN and normal neural network is better than the existing scheme. This is because the agent does not select malicious vehicles after the path and trust learning, and establishes a secure routing path. So the proposed scheme reduces the possibility of packet loss of each vehicle in VANETs. Meanwhile, if some trusted vehicles in the selected communication link change to the malicious nodes, the SDN controller will sense the new system state space and make a decision to select other trusted vehicles.

Fig. 2.14 shows that the proposed schemes have a slightly lower average ETX delay than the existing scheme as the number of vehicles grows. As the number of

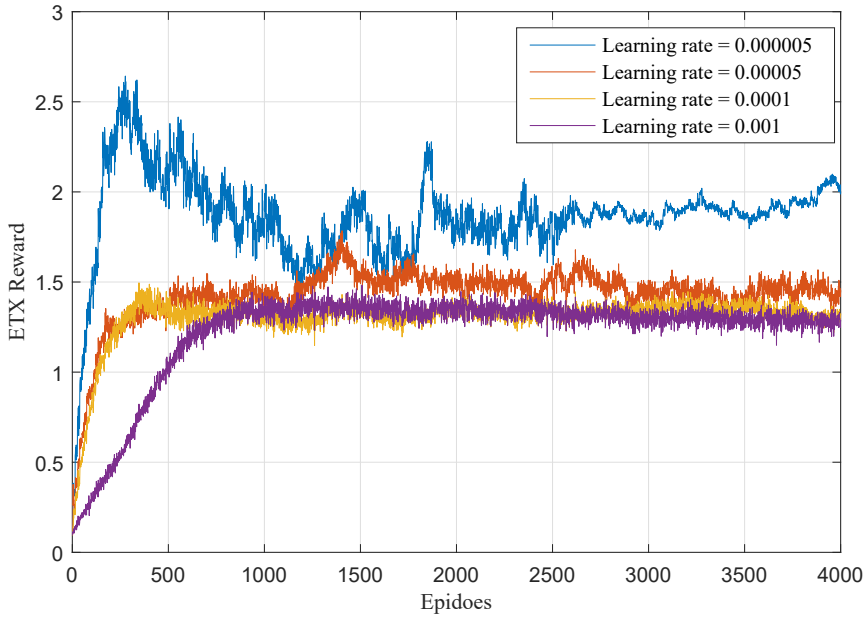

Figure 2.12: Comparison of the ETX delay with different learning rates.

vehicles grows, we can see that the ETX delay of all schemes all increases. However, the proposed schemes are still better than the existing scheme because of the trust feature of each vehicle. This feature enables the link quality of each proposed scheme to achieve a higher level than the existing scheme.

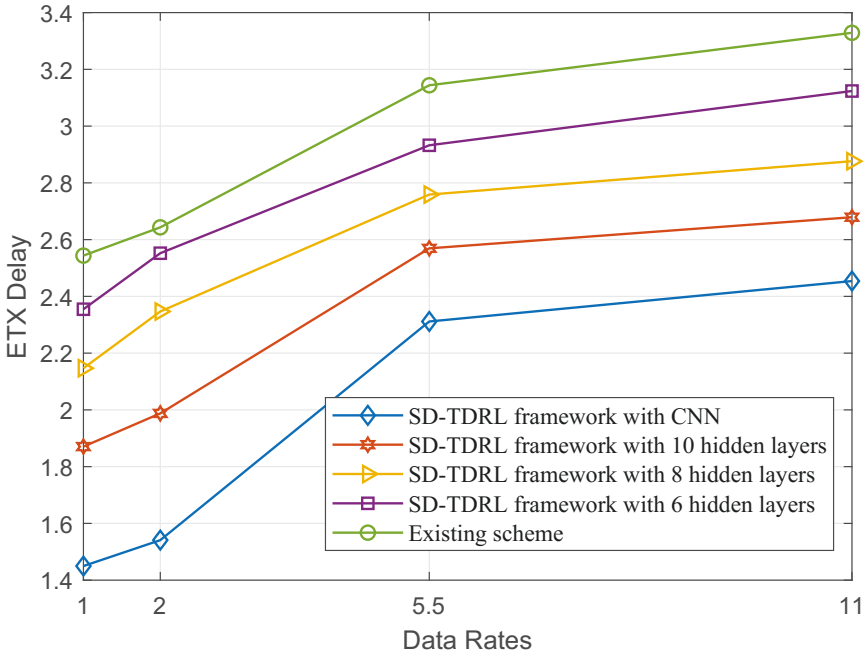

Figure 2.13: ETX **delay comparison with different data rates.**

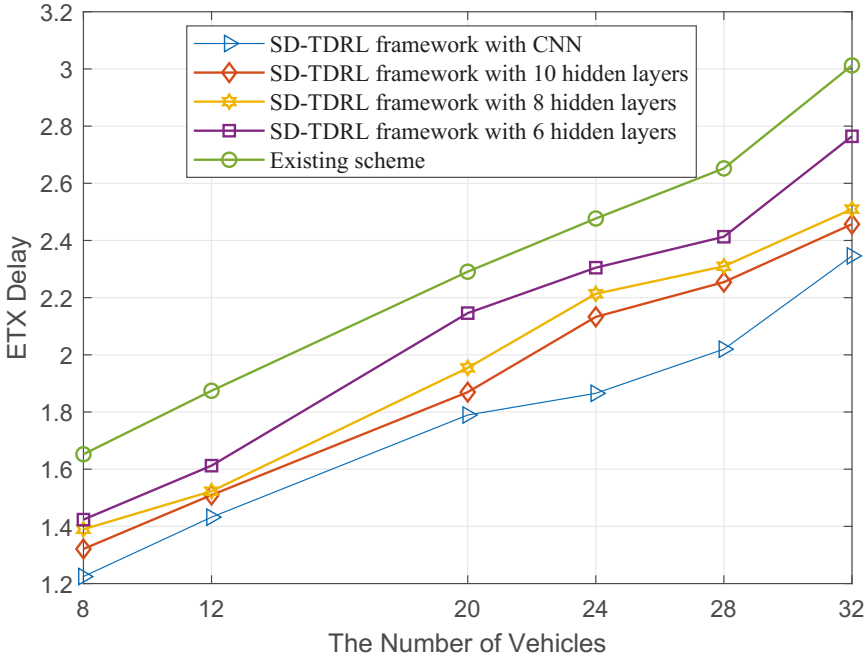

Figure 2.14: The ETX **delay comparison with different numbers of vehicles.**

Chapter 3

Blockchain-based Distributed Software-defined Vehicular Networks: A Dueling Deep Q-Learning Approach

Vehicular ad hoc networks (VANETs) have become an essential part in smart transportation systems of modern cities. However, because of dynamicity and infrastructure-less of VANETs, the ever increasing number of network security issues become obstacles for the realization of smart cities. Software-defined VANETs have provided a reliable way to manage VANETs dynamically and securely. However, the traditionally centralized control plane makes it vulnerable to malicious nodes and results in performance degradation. Therefore, a distributed control plane is necessary. How to reach a consensus among multiple controllers under complex vehicular environment is an essential problem. In this chapter, we propose a novel blockchain-based distributed software-defined VANET framework (block-SDV) to establish a secure architecture to overcome the above issues. The trust features of blockchain nodes, the number of consensus nodes, trust features of each vehicle, and the computational capability of the blockchain are considered in a joint optimization problem, which is modeled as a Markov decision process with state space, action space and

reward function. Since it is difficult to be solved by traditional methods, we propose a novel dueling deep Q-learning (DDQL) with prioritized experience replay approach. Simulation results are presented to show the effectiveness of the proposed block-SDV framework.

3.1 Blockchain-based Distributed SDV

In this section, we describe the architecture of block-SDV and the manner of multiple controllers interconnecting with each other using blockchain. We first introduce the network model, followed by the detailed consensus steps along with the theoretical analysis.

3.1.1 The Blockchain System Connected to Domain Controllers

Fig. 3.1 shows an overview of the hierarchical network architecture of proposed block-SDV, which can be divided into three layers, i.e., device layer, area control layer, and domain control layer. At the edge of the network, the connected vehicles in the device layer communicate with each other and send filtered data to the area control layer. The filtered data is the data packets that are transferred by a vehicle to its neighbors. These filtered data will be encapsulated into Packet-In message and sent them to a corresponding area controller, which decapsulates the data packets and extracts the link information. In the area control layer, it aims to collect vehicles and link information and sends to the domain control layer. The domain control layer operates in the distributed blockchain manner. The controllers in this layer collect the filtered data from the area control layer and share the data to other controllers.

In particular, block-SDV is able to share the model parameters of a domain controller training through the blockchain to other domain controllers in a transactional manner. In our proposed block-SDV architecture, each domain controller and area controller have different modules to interact with the blockchain and the device layer in block-SDV. For example, the local events and local OpenFlow commands will be encapsulated as several transactions in a domain controller, and then they are transmitted to the blockchain system. The transaction management module aims to collect these transactions. In an area controller, vehicles trust information module is used to collect the trust information coming from each area of the device layer. After collecting those values, the area controller will transfer those trust information to the domain control layer for further throughput calculation. These controllers continue to train and fine-tune based on the trained model parameters, thereby improving training efficiency and increasing system throughput. Because of the communication overhead between different control layers, the implementation of block-SDV is based on the two-tier architecture.

We assume that there are N nodes in a blockchain system, which can be defined as $\mathcal{N} = \{1, 2, ..., n, ..., N\}$. In a blockchain, distributed servers are blockchain nodes that manage and store information and authenticate by each other. In the domain control layer, there are M controllers, which can be expressed as $\mathcal{M} = \{1, 2, ..., m, ..., M\}$ deploying in a distributed manner and connecting with the blockchain system. For the consensus of block-SDV, we borrowed the idea of delegated proof of stake (D-PoS) algorithm to elect consensus nodes. The domain controllers select some nodes from $\mathcal{N}$ as consensus nodes through trust features of blockchain nodes. Under this consensus approach, those who hold tokens on a blockchain may select block producers through a trust features of blockchain nodes. The consensus node ensures that all nodes comply with the protocol rules and that all transactions are performed in

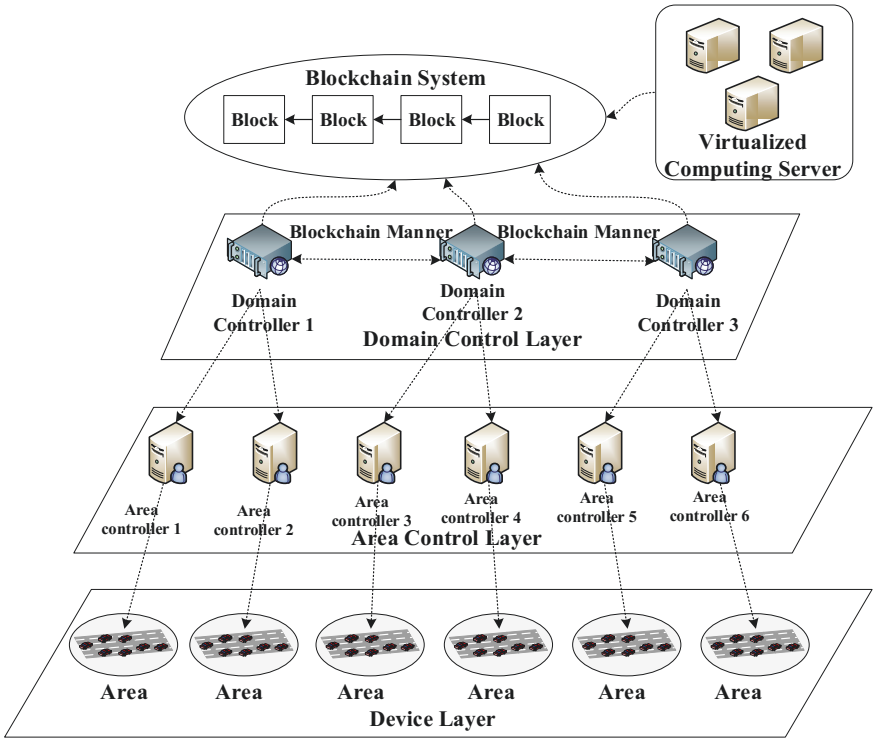

Figure 3.1: Proposed framework of block-SDV.

a reliable manner. The consensus nodes can be described as $\mathcal{C} = \{1, 2, ..., c, ..., C\}$ and $\mathcal{C} \in \mathcal{N}$. Specifically, each controller has multiple accounts that register on the blockchain. Each account of blockchain sends transactions to the blockchain system and is equipped with a public and private key to interact with the blockchain. Every domain controller needs to make a smart contract, i.e., they desire to share their neural network parameters with each other, and sign with their private key in order to guarantee the effectiveness of the smart contract. A smart contract is a computer protocol that communicates, validates, or enforces a contract in an informational manner. Smart contracts allow for trusted transactions without third parties, which are traceable and irreversible. Then the signed smart contracts from different domain controllers are transferred to the blockchain as different transactions. The verification

node is used to verify the data during transmission. These nodes determine the correctness of the data transmission by comparing the hash values of the blocks. If the hash values match, the verification passes. If the hash value is inconsistent, the data changes during the transmission and the verification fails. The validating nodes will validate those smart contracts and try to achieve consensus with each other. If the consensus is achieved successfully, the event in each smart contract will be executed. Hence, several cryptographic messages are needed to sign blocks, such as signatures and message authentication codes (MACs). According to [73], the message can be depicted as follows:

- $\langle k \rangle_{\chi_{cc'}}$ means that the message k is sent from node c to node c' containing a single MAC in blockchain.
- $\langle k \rangle_{\chi_c}$ means that the message k is signed with a public key from a node c.
- $\langle k \rangle_{\chi_{\vec{c}}}$ means that the message k is authenticated by a vector of MAC with sender c.

In the proposed framework, lots of computing resources are needed to verify through the above cryptographic messages and the smart contract. Unfortunately, computation of networking resources and blockchain system are limited in the block-SDV system. Virtualized distributed ledger technology (vDLT) [74] introduces an idea of virtualization into current blockchain systems. Hence, we virtualize these two resources to improve the throughput of proposed block-SDV.

3.1.2 Consensus Mechanism Analysis

In the proposed block-SDV, the local events and local OpenFlow commands are collected as $Transaction\ \#1$, $Transaction\ \#2$,..., $Transaction\ \#D$ in each domain controller.

The consensus mechanism in block-SDV is to ensure that all consensus nodes in the blockchain execute and write each transaction into distributed ledgers in the same order, and accounting nodes only need to synchronize the ledger information from the consensus nodes, so there are no need to participate in the consensus process. As shown in Fig. 3.2, after verifying signatures, MACs, and smart contract, block-SDV sends back a validated block to the domain control layer and appends this block in the blockchain. The domain controllers analyze the payload information, i.e., each domain controller knows parameters of deep neural network from other domain controllers, so as to share the parameter of each controller to improve the training efficiency and to synchronize the global network view among the multiple domain controllers.

Next, we depict the detailed consensus mechanism named redundant Byzantine fault tolerance (RBFT) inside the blockchain as shown in Fig. 3.3 along with [75]. We assume the Byzantine failure model, in which at most $f = \frac{C-1}{3}$ nodes are faulty [76]. RBFT has been used in some real scenarios, such as Hyperchain project [77]. It is hosted by the Linux Foundation, and develop applications with a modular architecture. Thus, we consider that the RBFT can be used in some real scenarios when deploying SDN controllers.

As we know, practical Byzantine fault tolerance (PBFT) is a popular consensus mechanism used in blockchain. Comparing with PBFT, RBFT has added a procedure called transaction verification. The primary node packages the transaction into blocks and verify it, and adds the verification result in the $PRE-PREPARE$ message for broadcasting to the whole network. The detailed consensus steps inside the proposed block-SDV using RBFT-DPOS are described as follows:

1. **All controllers select the number of consensus nodes.** As we described before, there are M domain controllers in the domain control layer. These controllers

aim to select appropriate consensus nodes according to the trust features of all nodes in blockchain. The consensus nodes are chosen by preference of votes cast by the domain controllers.

2. **All controllers send request messages to all the nodes.** A controller in the domain layer sends its transactions (request message) $\left\langle \langle transactions, m\rangle_{\chi_m}, m\right\rangle_{\chi_{\vec{m}}}$ to all the consensus nodes. This message contains the controller ID m and total number of transactions D. It is signed with m's private key, and then authenticated with a MAC authenticator for all nodes in the blockchain. Once receiving the message, a consensus node first verifies the MAC. If the MAC is valid, then the signature of the message is verified by this consensus node. If the MAC is invalid, further request messages will not be processed.

3. **All the nodes propagate the request message to all other replicas.** In this phase, each node propagates the receiving request message to all other consensus nodes once the request has been verified. This process ensures that every correct node will eventually receive a request as long as at least one correct node receives the request before. For example, once a node c receives the $PROPAGATE$ message $\left\langle PROPAGATE, \langle transactions, m\rangle_{\chi_m}, c\right\rangle_{\chi_{\vec{c}}}$ from node c', node c first verifies the MAC authenticator. If the MAC is valid, then c verifies the signature of the receiving transactions. If the signature is valid, node c sends the $PROPAGATE$ message to all other nodes. Each replica generates $(C-1)$ MACs and $(C-1)$ signatures, and each replica receives $f+1$ propagate messages from other replicas.

4. **Replicas of the primary node execute three phase actions to process the receiving request.** As shown in Fig. 3.3, when the $(C-1)$ replicas receive requests, the primary sends $PRE-PREPARE$ messages $\langle PRE-PREPARE, Pr, m, H(m)\rangle_{\chi_{\vec{Pr}}}$ authenticated by MAC for replicas, where Pr is the primary's ID and $H(m)$ is the hashed results of issued block. The primary

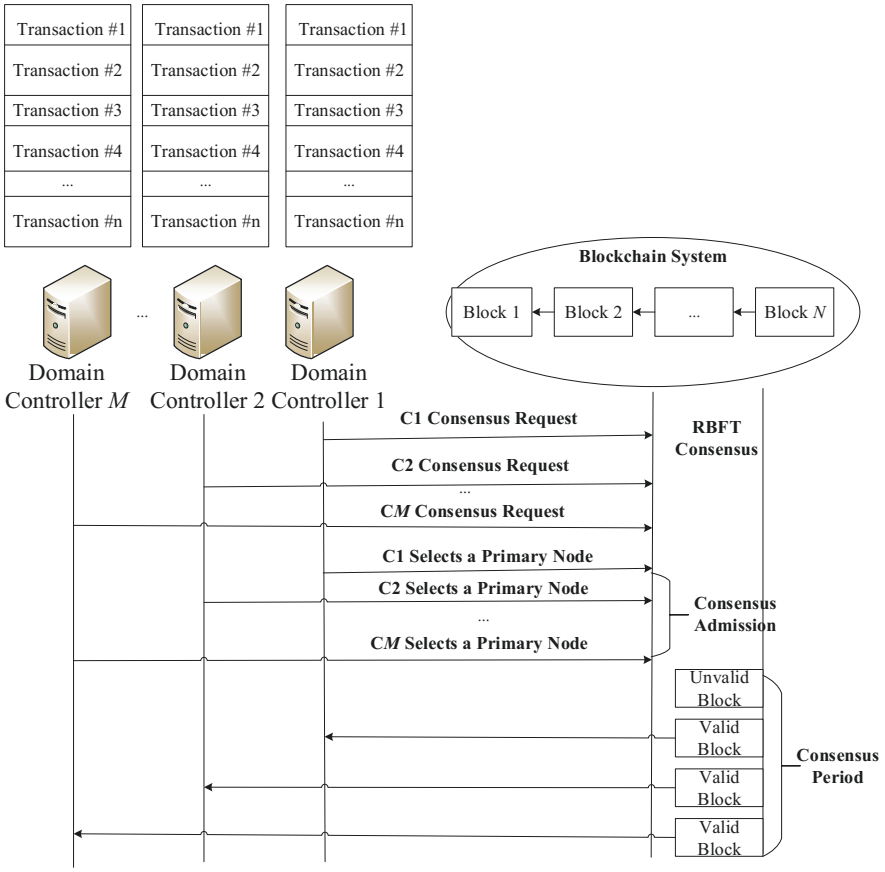

Figure 3.2: The interaction between domain controllers and blockchain.

node generates $(C-1)$ MACs for all other replicas.

When a replica receives a $PRE-PREPARE$ message from the primary node, it verifies the validity of the MAC. It then replies $PREPARE$ message $\langle PREPARE, Pr, m, H(m), r\rangle_{\chi_{\vec{r}}}$ to all other replicas, where r is the replica's ID and each replica generates $(C-1)$ MACs. Meanwhile, each replica needs to follow the reception of $2f$ matching $PREPARE$ message from distinct replicas that are consistent with the proper $PRE-PREPARE$ message.

After finishing to process the $PREPARE$ message, each replica sends a commit message that is authenticated with a MAC authenticator. For example, a replica c generates $(C-1)$ MACs to all other replicas, and it verifies one MAC to check the

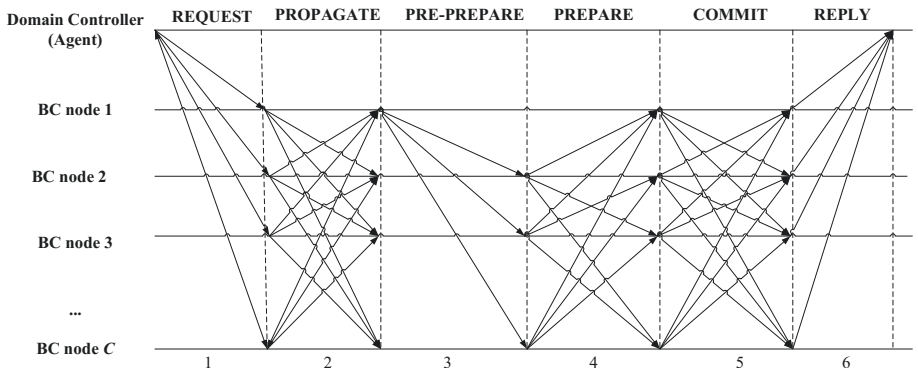

Figure 3.3: The consensus procedures inside of the blockchain.

validity of the incoming $COMMIT$ message $\langle COMMIT, Pr, m, H(m), r\rangle_{\chi_{\vec{r}}}$. After receiving incoming $2f+1$ matching $COMMIT$ messages from distinct replicas, c' will verify the smart contract. If valid, it will append this block to the blockchain.

5. **The nodes execute the request and send reply message to all controllers in the domain layer.** After the request message operation is executed, each node sends reply message $\langle REPLY, bl, c\rangle_{\chi_{c,m}}$ to a controller in the domain layer, where bl is an ID of a valid block and c is a node's ID. After receiving the reply messages from the blockchain system, each controller needs to verify $f+1$ valid reply message and D transactions in the reply message in order to confirm that the smart contract is executed successfully in each replica. If D transactions matching with the $f+1$ reply messages, the controller accepts this block and gets the information inside the block to update its network view.

In block-SDV, we made some improvements to the traditional DPoS-BFT consensus algorithm [77]. In the traditional DPoS algorithm, anyone can be chosen to participate in block production, as long as they can convince the holder (domain controller account) to vote for it, they will have the opportunity to participate in block production. Since the proposed block-SDV is a permissioned blockchain system, the traditional DPoS-BFT consensus mechanism is not suitable for block-SDV. Therefore, each domain controller selects the nodes participating in the consensus according to

the trust degree of each blockchain node. Since the malicious nodes in blockchain degrade the overall system performance, the consensus mechanism in block-SDV ensures that each consensus node does not do evil in the process of participating in the consensus. In block-SDV, if the master primary node of the system is malicious or the performance is degraded, each domain controller selects a new master primary node according to the trust degree of the nodes in each protocol instance.

3.2 The Throughput Analysis of Block-SDV

In this section, we introduce a VANET trust model in the device layer. The area controller is used to collect the vehicles' data coming from the device layer, and transfer the data to the domain control layer.

3.2.1 Trust Derivation

No matter what kind of trust model is used, direct trust and indirect trust are available. Direct trust value in the VANET environment means the first-hand routing information of neighbor nodes and easy to be obtained. Indirect trust value is second-hand routing information about nodes, such as recommend trust information from a third party. In order to simplify our trust model, we only use the history of direct interactions among vehicles to compute trust. Specifically, direct trust assessment is evaluated by the transmitted packets among the adjacent vehicles in their communication range, irrelevant the destination vehicle.

We assume that the distance between the source vehicle and destination vehicle is more than one hop, and the transmitted packets will be dropped randomly because of some unexpected causes, such as black-hole attack and changeable channel environment. In the device layer of proposed block-SDV, trust evaluation is decided by the

Table 3.1: A neighbor's table format in each vehicle

Neighbor's ID
Vehicle's Postion
Velocity
Heading
Available Throughput (AT)
Trust value ($T_{v_b v_b'}(t)$)
Last Packet Sequence Number
Packet Buffer

assessment of forwarding behavior of neighbors by a sender. A wireless link exists between any two vehicles only if they are located within the transmission range of each other. In other words, the two vehicles are adjacent to each other. Meanwhile, each vehicle is equipped with an On-Board-Unit (OBU) that interacts with a Road-Side-Unit (RSU) to upload the vehicle-to-vehicle routing message. For example, a vehicle v communicates with its neighbor vehicle v' by sending packets, when receiving the coming packets, vehicle v' gives a trust score for sender v. Hence, we consider to set up the trust calculation for vehicles in each area of the device layer.

In each area of block-SDV, a data packet contains the packet sequence number to ensure the freshness of the information carried by the packet. The header information includes the sender ID used to determine the node in the VANET environment. Meanwhile, the header of a packet also contains the position information and velocity of the sending node. Furthermore, a header also records the trust between adjacent nodes. Each vehicle in an area manages a neighbor table that holds information obtained from the HELLO packet as shown in Table 3.1. In detail, when a packet reaches a certain neighbor node, the node will decapsulate the packet and will store the information in the neighbor table.

Trust computation: In the trust model, we assume that there are $\mathcal{V} = \{1, 2, ..., v, ...V\}$ vehicles in each area of the device layer and there are $\mathcal{B} = \{1, 2, ..., b, ..., B\}$ areas in the device layer. Moreover, each vehicle interacts with

its neighbors to update their trustworthiness. Vehicle $v_b, v_b^{'}$ in an area b, interact with each other in a predetermine time interval $[t,\ t + \delta_t]$. Direct trust of each vehicle is computed based on the interaction of two neighbor vehicles using the following formula:

$$T_{v_b v_b^{'}}(t) = \frac{f^C_{v_b v_b^{'}}(t)}{f_{v_b v_b^{'}}(t)} \quad t \leq W \tag{3.1}$$

where $T_{v_b v_b^{'}}(t)$ denotes the direct trust for vehicle v_b towards its neighborhood $v_b^{'}$. $f_{v_b v_b^{'}}(t)$ denotes the total number of packets forwarded from v_b towards node $v_b^{'}$, and $f^C_{v_b v_b^{'}}(t)$ denotes the number of packets that correctly forwarded in the time period t, where W represents the width of the recent time window. Here, the packet sequence number is used to determine the number of lost and correct forwarding packets. According to the sequence number of each packet, a packet is marked as either correctly received or lost in the recent time window W. This is done by comparing the newly received packet sequence number with the last received packet sequence number in the neighbors' routing table. Therefore, each node enables to detect the correct forwarding.

After each interaction, vehicle $v_b^{'}$ checks whether vehicle v_b forwards packets correctly at time slot t. If so, the trust value $T_{v_b v_b^{'}}(t)$ increases. Otherwise, the trust value $T_{v_b v_b^{'}}(t)$ decreases. In the trust model of block-SDV in device layer, the trust value range of each vehicle is limited from 0 to 1 (i.e., $0 \leq T_{v_b v_b^{'}}(t) \leq 1$). The trust value of 0 means complete distrust whereas the trust value of 1 implies absolute trust. If there is no interaction between two vehicles in each area, the initial trust value is set to 0.6 (less trustworthiness vehicle). Here, we introduce a threshold θ to depict the trust threshold, which is used to distinguish the malicious vehicles. In other words, if the trust value of any vehicle is less than threshold θ, it can be regarded as a malicious node. In order to minimize the possibility of transmission failure, each vehicle in

Table 3.2: Trust level of vehicles

Trust Level	Trust Value	Explanation
1	[0, θ]	Malicious vehicle
2	(θ, 0.6]	Suspect vehicle
3	(0.6, 0.85]	Less trustworthy vehicle
4	(0.85, 1]	Trustworthy vehicle

an area needs to establish the communication with its most trusted neighbor vehicle whose the trust value is higher than the trust requirements as shown in Table 3.2 [78].

3.2.2 The Theoretical Analysis of Network Throughput

If we do not consider the available bandwidth when selecting a trusted neighbor vehicle as next hop, it might result in high packet delays and losses. Therefore, we define available throughput (AT) to evaluate the maximum data rate that can be successfully sent by each vehicle v_b as:

$$AT_{v_b v_b^{'}} = \frac{T_{v_b v_b^{'}} \cdot f_{v_b v_b^{'}}}{t_{trans}^{v_b v_b^{'}} + t_{pro}^{v_b v_b^{'}}} \tag{3.2}$$

where $T_{v_b v_b^{'}} \cdot f_{v_b v_b^{'}}$ denotes total number of packets successfully transmitted by vehicle v_b, and $t_{trans}^{v_b v_b^{'}}$ represents the total time of vehicle v_b spent in transmitting packets and $t_{pro}^{v_b v_b^{'}}$ represents propagation time of transmitting those packets. According to [69], $t_{trans}^{v_b v_b^{'}}$ includes two different components: successful packet transmission time and unsuccessful packet transmission time:

$$t_{trans}^{v_b v_b^{'}} = \sum_{z=1}^{U_s} t_z^s + \sum_{z=1}^{U_f} t_z^f \tag{3.3}$$

where U_s denotes the total number of packets successful to be transmitted to v_b's neighbor vehicle $v_b^{'}$, and U_f represents the total number of packets failure to be

transmitted to its neighbor $v_b{}'$.

We assume that the number of transmission attempts of vehicle v_b to transmit a packet z to its neighbor successfully $v_b^{'}$ is $U_s^{'}$ using IEEE802.11p enhanced distributed channel access (EDCA). Hence, the successful packet transmission time can be defined as follows:

$$t_z^s = \overline{CW}_{U_s'} + T_f + \sum_{u=1}^{U_s'} (\overline{CW}_u + T_s) \tag{3.4}$$

where $\overline{CW}_u$ denotes the average contention window size for the packet transmission attempts, and $\overline{CW}_{U_s'}$ denotes the average contention window size for the last successful attempt of transmitting packet z. According to [79], T_s represents the successful transmission time for an access category, and it can be defined as follows:

$$T_s = T_{AIFS_{AC_u}} + T_{Header} + T_{packet} + T_{SIFS} + T_{ACK} + 2\nu \tag{3.5}$$

where ν denotes the propagation delay, T_{Header} and T_{packet} represent the time cost for packet header and data information. T_{SIFS} and T_{ACK} are the time cost for short inter-frame space and ACK acknowledgment. $T_{AIFS_{AC_u}} = AIFS_{AC_u} \times T_{slot} + T_{SIFS}$ denotes the time cost of arbitration interframe space on access category (AC_u).

Similarly, the unsuccessful packet transmission time, which is dropped after a maximum number of retransmissions ($U_f^{'}$), is:

$$t_z^f = \sum_{u=1}^{U_f'} (\overline{CW}_u + T_f) \tag{3.6}$$

where T_f denotes the unsuccessful transmission time for an access category, and it

can be defined as follows:

$$T_f = T_{AIFS_{AC_u}} + T_{Header} + T_{packet} + T_{SIFS} + T_{ACK_{timeout}} + \nu \tag{3.7}$$

where $T_{ACK_{timeout}} = T_{SIFS} + T_{slot}$ is the ACK timeout duration.

Finally, According to [69], the average contention window size can be defined as:

$$\overline{CW}_u = \frac{\min\left(CW_{\max}, 2^u \times (CW_{\min} - 1)\right) \times T_{slot}}{2} \tag{3.8}$$

where $CW_{\max}$ and $CW_{\min}$ are the maximum and minimum contention window size defined based on the IEEE 802.11p access category.

Simultaneously, we need to define the $t_{pro}^{v_b v_b^{'}}$ as:

$$t_{pro}^{v_b v_b^{'}} = \frac{d_{v_b v_b^{'}}}{\lambda} \tag{3.9}$$

where λ is propagation speed, which depends on the physical medium of the link, and $d_{v_b v_b^{'}}$ represents the distance between two neighbor vehicles $v_b(x_{v_b}, y_{v_b})$ and $v_b^{'}(x_{v_b^{'}}, y_{v_b^{'}})$, and can be defined as:

$$d_{v_b v_b^{'}} = \sqrt{(x_{v_b^{'}} - x_{v_b})^2 + (y_{v_b^{'}} - y_{v_b})^2} \tag{3.10}$$

Therefore, the available throughput $AT_{v_b v_b^{'}}$ between two vehicles v_b and $v_b^{'}$ can be obtained from (3.2), (3.3) and (3.9).

3.2.3 The Theoretical Throughput Analysis of Blockchain System

As described in Section 3.1.2, there are six steps when blockchain nodes try to achieve consensus with each other. In each step, the cost can be divided into two parts: the cost of the primary node and the cost of each replica, which can be defined as follows.

1) **Theoretical analysis for all the controllers sending request messages to all the nodes**: We assume that there are D transactions transferred to the blockchain system from domain control layer, and a fraction g of transactions sent by the controller are correct [80]. We consider that each consensus node generating one MAC, verifying one MAC and one signature needs α, α, and β cycles, respectively. Hence, the cost of a primary node in this step can be shown as:

$$C_p^{req} = (\alpha + \beta) + \frac{D}{g}(\alpha + \beta) \tag{3.11}$$

and the cost of each replica C_r^{req} is the same as C_p^{req}.

2) **Theoretical analysis for the propagation procedure**: In this phase, we assume that the primary node first receives the incoming transactions, and verifying the MAC and signature, then it sends $PROPAGATE$ messages to all other consensus nodes. Thus, the cost of the primary node is:

$$C_p^{pro} = (\frac{D}{g} + C - 1)(\alpha + \beta) \tag{3.12}$$

and the cost of each replica is:

$$C_r^{pro} = (\frac{D}{g} + 1)(\alpha + \beta) \tag{3.13}$$

3) **Theoretical analysis for the pre-prepare procedure**: In this phase, the primary node generates $C-1$ MACs for all replicas, who will verify the MACs and signatures of $\frac{T}{g}$ transactions and $C-1$ MACs of $PROPAGATE$ message. Meanwhile, each replica stores the receiving transactions. Therefore, the cost of the primary node can be shown as:

$$C_p^{pre} = (C-1)\alpha \tag{3.14}$$

and the cost of each replica is:

$$C_r^{pre} = \alpha + \frac{D}{g}(\alpha + \beta) \tag{3.15}$$

4) **Theoretical analysis for the prepare procedure**: In this phase, the primary node needs to verify $2f$ matching $PARPARE$ messages, and each replica generates $C-1$ MACs for all other consensus nodes as well as verifies $2f$ MACs. Thus, the cost of the primary node is:

$$C_p^{par} = 2f\alpha \tag{3.16}$$

and the cost of each replica is:

$$C_r^{par} = (C-1+2f)\alpha \tag{3.17}$$

5) **Theoretical analysis for the commit procedure**: In this phase, the primary node generates $C-1$ MACs and verifies $2f+1$ MACs. Each replica also needs to generate $C-1$ MACs for all other nodes and to verify $2f+1$ MACs. Therefore, the

cost of the primary node is:

$$C_p^{com} = (C + 2f)\alpha \tag{3.18}$$

and the cost of each replica C_r^{com} is the same as C_p^{com}.

6) **Theoretical analysis for the reply procedure**: In this phase, the primary and replicas need to generate $\frac{D}{g}$ MACs for transactions' operation of M domain controllers. Therefore, the cost of the primary node is:

$$C_p^{rep} = \frac{MD}{g}\alpha \tag{3.19}$$

and the cost of each replica C_r^{rep} is the same as C_p^{rep}.

Hence, the final cost of the primary node for each transaction is:

$$\begin{aligned} C_p &= C_p^{req} + C_p^{pro} + C_p^{pre} + C_p^{par} + C_p^{com} + C_p^{rep}/D \\ &= (\frac{M+2}{g} + \frac{C}{D})\alpha + (\frac{2}{g} + \frac{C}{D})\beta + \frac{4f + 2C - 1}{D}\alpha \end{aligned} \tag{3.20}$$

For the replicas, the cost for each transaction can be calculated as:

$$\begin{aligned} C_r &= C_r^{req} + C_r^{pro} + C_r^{pre} + C_r^{par} + C_r^{com} + C_r^{rep}/D \\ &= (\frac{M+3}{g} + \frac{2}{D})\alpha + (\frac{3}{g} + \frac{2}{D})\beta + \frac{4f + 2C}{D}\alpha \end{aligned} \tag{3.21}$$

We assume that multi-core computation modules run in parallel on distinct cores. Each core has a computation speed of η Hz. Meanwhile, we consider that the primary node has the possibility k trust to affect the performance of the blockchain system, and $k \in (0, 1]$. Therefore, the throughput $BT_{blockchain}$ of the blockchain system is at

most:

$$BT_{blockchain} = \min\left[\frac{k\eta}{C_p}, \frac{k\eta}{C_r}\right] tx/s \tag{3.22}$$

Here, the $BT_{blockchain}$ is used to estimate the number of transactions that the blockchain system can process per second.

3.3 Problem Formulation

In previous section, we have already proposed the throughput model of block-SDV. In order to optimize the throughput, we jointly consider three different scenarios. In this section, we formulate this joint optimization problem as a Markov decision process, in which we define system state space, action space, and reward function.

3.3.1 System State Space

The learning agent (each controller in the domain control layer) needs to know the state space $S(t)$ at time slot t. From the above description, the state space $S(t)$ includes the trust feature of vehicles $T_{v_b v_b'}(t)$, trust feature of each node in the blockchain $\phi_p^C(t)$, and computing capability $\mu_E(t)$. Here, since there are no centralized security service, all nodes and controllers have distinct trust features. Moreover, $\gamma_N(t)$ denotes that which node is the consensus node, i.e., if $\gamma_n(t) = 1$, it represents that node n can be selected as a consensus node. Hence, the system state space jointly considers the trust feature of each node in the blockchain, the trust feature of each vehicle in device layer, computing capabilities of each edge computing servers, and the number of consensus nodes in the blockchain. The state transition diagram is

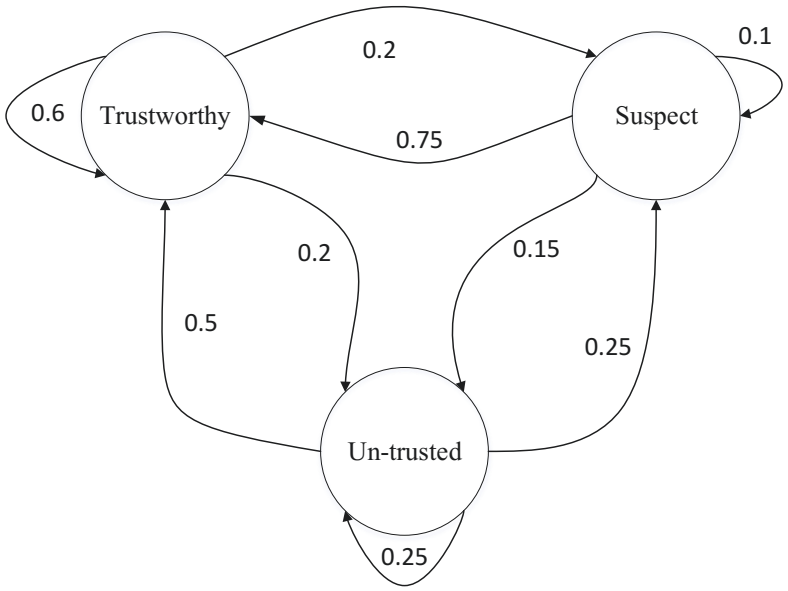

Figure 3.4: An example of the state transition diagram for the trust features of each blockchain node.

shown in Fig. 3.4. The system states can be defined as:

$$S(t) = \left\{ T_{v_b v_b'}(t), \phi_p^c(t), \gamma_N(t), \mu_E(t) \right\} \tag{3.23}$$

Here, the trust feature of each vehicle in each area of the device layer has already been described in Section 3.2.1. Since the communication channel is always changing, we hardly know the trust feature for each vehicle in an area of the device layer in the next time slot t. Hence, the state transition probability for $T_{v_b v_b'}(t)$ is $\zeta_{T_{v_b v_b'}(t) T_{v_b v_b'}(t+1)}$, and the state transition probability matrix can be represented as $\Upsilon = [\zeta_{T_{v_b v_b'}(t) T_{v_b v_b'}(t+1)}]_{J \times J}$. We also consider the trust feature of blockchain nodes because the learning agent needs to know the trust feature of the nodes in the blockchain system so as to select the primary node in a consensus period. Thus, the trust feature of a node $c \in \{1, 2, ..., C\}$ can be modeled as random variable $\phi_p^c(t)$ at time slot t. Since the trust feature of each consensus node is always changing, the transition

probability of $\phi_p^c(t)$ is $\Phi_{\phi_p^c(t)\phi_p^c(t+1)}$, and the state transition probability matrix can be defined as $\Psi = [\Phi_{\phi_p^c(t)\phi_p^c(t+1)}]_{H\times H}$. Meanwhile, each domain controller enables to adjust the number of consensus nodes dynamically and can be modeled as random variable $\gamma_n(t)$ at time slot t. Because of the varying trust feature of each consensus node, we do not know the number of consensus nodes at next time slot t. Hence, the transition probability for $\gamma_n(t)$ is $\Gamma_{\gamma_n(t)\gamma_n(t+1)}$, and the transition probability matrix can be defined as $\Omega = [\Gamma_{\gamma_n(t)\gamma_n(t+1)}]_{Y\times Y}$. Finally, we use virtual computing servers to do computing tasks in the block-SDV. There are many different computing tasks in the blockchain system, such as verifying signatures, generating MACs, and verifying MACs. Let $H_o = \{d_o, q_o\}$ denote a computing task related to message o, where d_o means the size of message o, and q_o means the required number of CPU cycles to complete this task. We do not know the computational resources for the block-SDV at next time slot. Therefore, we model the computation resources of edge computing server $e \in \{1, 2, ..., e, ...E\}$ as a random variable $\mu_e(t)$ at time slot t. There are T time slots, which starts when the domain controller issues un-validated block and terminates when the controller is replied a validated block.

3.3.2 System Action Vector

The learning agent in the domain control layer needs to decide which one is a primary node in the blockchain system, which edge computing server is selected for computing resources, how many consensus nodes and select the trusted neighbor vehicles. Therefore, the system action vector is:

$$A(t) = \{A_{\mathcal{V}}(t), A_p(t), A_{\mathcal{N}}(t), A_{\mathcal{E}}(t)\} \tag{3.24}$$

where $A_{\mathcal{V}}(t)$, $A_p(t)$, $A_{\mathcal{C}}(t)$ and $A_{\mathcal{E}}(t)$ are described in the following.

1) $A_{\mathcal{V}}(t) = \{a_1(t), a_2(t), ..., a_v(t), ..., a_V(t)\}$ represents the action vector of vehicles

in each area of device layer. The value of $a_V(t)$ is $\{0, 1\}$. For example, if $a_1(t) = 0$ at time slot t, it means that vehicle 1 is not to be selected as trusted neighbor vehicle. If $a_1(t) = 1$, it means that vehicle 1 can be selected by learning agent as trusted neighbor vehicles. Moreover, each vehicle aims to choose a most trusted node (highest trust value) as its neighbor for communication. Therefore, let a route P, consisting L nodes, representing as $\mathcal{L} = \{1_b, ..., l_b, ..., L_b\}$ and $\mathcal{L} \in \mathcal{V}$, where node l_b denotes the lth node in the route. The total trust for this routing path in an area b can be depicted as $\prod_{l_b=1}^{L_b} T_{l_b, l+1_b}(t)$.

2) $A_p(t) = \left\{a_p^1(t), a_p^2(t), ..., a_p^c(t), ..., a_p^C(t)\right\}$ represents the action vector of consensus nodes in the blockchain. The value of $a_p^c(t)$ is $\{0, 1\}$. For example, if $a_p^1(t) = 0$ at time slot t, it is mean that node 1 is a replica. If $a_p^1(t) = 1$, it is means that node 1 can be selected by learning agent as primary node. As we described in Section 3.1.2, the blockchain system only has one primary node at the same time, so $\sum_{c=1}^{C} a_p^c(t) = 1$.

3) $A_{\mathcal{N}}(t) = \{a_1(t), a_2(t), ..., a_n(t), ..., a_N(t)\}$ represents the action vector of selecting the number of consensus nodes. The value of $a_n(t)$ is $\{0, 1\}$. For example, if $a_1(t) = 0$ at time slot t, it is means that node 1 is not to be selected as consensus node. If $a_1(t) = 1$, it is means that node 1 can be selected by learning agent as consensus node.

4) $A_{\mathcal{E}}(t) = \{a_1(t), a_2(t), ..., a_e(t), ..., a_E(t)\}$ represents the action vector of the edge computing servers. There are E edge computing servers, and the set of those computing servers is represented by $\mathcal{E} = \{1, 2, ...e, ..., E\}$. The value of $a_e(t)$ is $\{0, 1\}$. When the learning agent selects a edge computing server to offload, the value of the action is 1, otherwise, the value of action is 0. Therefore, $\sum_{e=1}^{E} a_e(t) = 1$. The execution time of computing task H_o is $t_o = q_o/\mu_e(t)$, and the computing rate η is $\sum_{e=1}^{E} a_e(t) \frac{d_o \mu_e(t)}{q_o}$.

3.3.3 Reward Function

In order to improve the system throughput, we model the throughput, which combines the network throughput with the throughput of the blockchain system, as the immediate reward function. According to (3.2) and (4.26), system state space and action vector, the immediate reward $r(t)$ can be defined as follows:

$$\begin{aligned} r(t) =& \sigma \left[\sum_{b=1}^{B} \frac{\prod_{l_b=1}^{L_b} T_{l_b,l+1_b} \cdot f_L}{\sum_{l_b=1}^{L_b} (t_{trans}^{l_b,l+1_b} + t_{pro}^{l_b,l+1_b})} \right] + \delta \min \Bigg[\\ & \frac{k\eta}{(\frac{M+2}{g} + \frac{C}{D})\alpha + (\frac{2}{g} + \frac{C}{D})\beta + \frac{4f+2C-1}{D}\alpha}, \\ & \frac{k\eta}{(\frac{M+3}{g} + \frac{2}{D})\alpha + (\frac{3}{g} + \frac{2}{D})\beta + \frac{4f+2C}{D}} \Bigg] \end{aligned} \tag{3.25}$$

where $f_L = \sum_{l=1}^{L} f_l$, $k = \sum_{c=1}^{C} a_p^c(t)\phi_p^c(t)$ and $C = \sum_{n=1}^{N} a_n(t)\gamma_n(t)$.

From the above formulation, the immediate reward of block-SDV is served as $r(t)$. Specifically, the learning agent senses the system state space $S(t)$ from the environment at time slot t, then the agent outputs a policy π that determines which action should be executed from the system action vector $A(t)$. The reward value will be returned to the learning agent at time slot t. Then the system state space changes to the next state $S(t+1)$, and the learning agent outputs new policy and gets a new immediate reward $r(t+1)$. The learning agent aims to find an optimal policy to maximize the long-term reward, and the cumulative reward r^{long} can be written as:

$$r^{long} = \max E \left[\sum_{t=1}^{t=T-1} \gamma^t r(t) \right] \tag{3.26}$$

where γ^t approaches to zero when t is large enough.

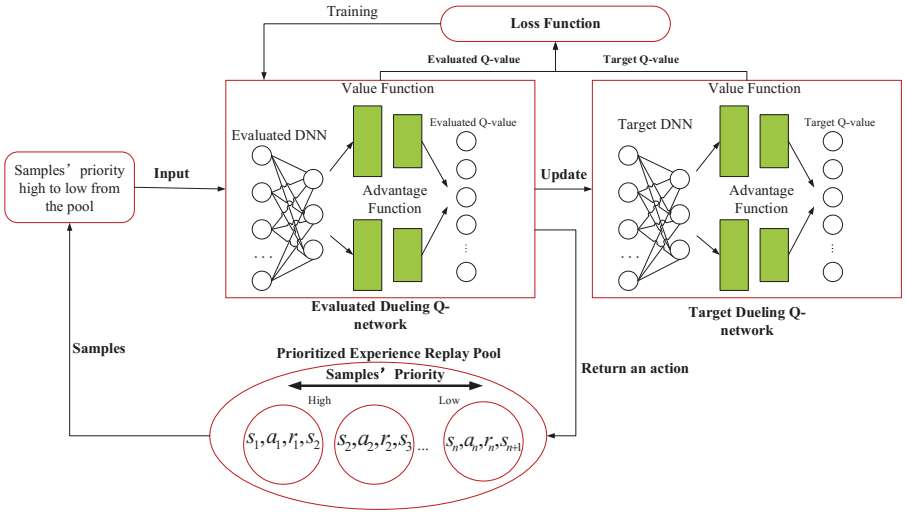

Figure 3.5: The workflows of proposed DDRL in block-SDV.

3.4 Dueling Deep Q-Learning with Prioritized Experience Replay

In this section, we consider using dueling deep Q-learning approach with prioritized experience replay to address the joint optimization problem as we described in Section 3.3. There are some reasons to use this approach: 1) The system has high-dimensional and high-dynamical because of the system state defined in (3.23), and it is hard to be solved using traditional optimization methods; 2) In block-SDV, choosing which action from action space $A(t)$ by the learning agent has no relationship with what happens in the next time slot because of the transition probability as we described in Section 3.3. Therefore, the traditional optimization method that considers the relationship between state and action is not suitable. Firstly, we simply introduce the mechanism of dueling deep Q-learning with prioritized experience replay, and then we present the approach used in this chapter.

3.4.1 Dueling Deep Q-Learning

A deep Q-learning (DQL) concept introduced by [49] aims to solve the instability of traditional Q-network. There are two important improvements compared with traditional reinforcement learning method: *experience replay* and *target Q-network*. Experience replay stores trained data and then randomly samples from the pool. Therefore, it reduces the correlation of data and improves the performance compared with the previous reinforcement learning algorithms [81]. Meanwhile, *target Q-network* is another improvement in the deep Q-learning. That is, it calculates a target Q-value using a dedicated target Q-network, rather than directly using the pre-updated Q-network. The purpose of this is to reduce the relevance of the target calculation to the current value.

Although deep Q-learning makes big improvements comparing with traditional reinforcement learning methods, many researchers still make great efforts for even greater performance and higher stability. Here, we introduce a recent improvement: dueling deep Q-learning [2]. The core idea of this approach is that it does not need to estimate the value of taking each available action. For some states, the choice of action makes no influence on these states themselves. Thus, the network architecture of DDQL can be divided into two main components: value function and advantage function. The value function is used to represent how good it is to be in a given state, and advantage function can measure the relative importance of a certain action compared with other actions. After value function and advantage function are separately computed, their results are combined back to the final layer to calculate the final Q-value. The mechanism of DDQL would lead to better policy evaluation comparing with DQL.

Algorithm 2 Dueling deep Q-network for block-SDV

Initialization
Initialize prioritized replay memory F, memory size G, $\triangle = 0$, $p_1 = 1$, and minibatch q.
Initialize the evaluated deep Q-network with initial weights. ω
Initialize the target deep Q-network with weights ω^{-}.
for episode k = *1, 2, 3,...,K* **do**:
Initialize the beginning state $S(t)$
Initialize the action set $A(t)$
Initialize the time step t, exponents κ and τ
for t = *1,2,3,...,T* **do**
Choose a random probability e
If $(e \geq \epsilon)$,
$A^{*}(t) = \arg\max Q(S(t), A(t), \omega)$,
Otherwise,
Randomly choose an action$A(t) \neq A^{*}(t)$,
obtain the immediate reward $r(t)$ and next state $S(t+1)$.
Store experience $(S(t), A(t), r(t), S(t+1))$ into prioritized experience replay memory F
for $j = 1$ to q **do**
Sampling $j \sim P(j) = p_j^{\tau} / \sum_i p_i^{\tau}$.
Compute $w_j = (G \cdot P(j))^{-\kappa} / \max_i w_i$.
Compute TD-error:
$\rho_j = R_j + \gamma_j Q_{target}(S_j, \arg\max Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$.
Update sample j priority $p_j \leftarrow |\rho_j|$.
Accumulate weight-change:
$\triangle \leftarrow \triangle + w_j \cdot \rho_j \cdot \bigtriangledown_{\omega} Q(S_{j-1}, A_{j-1})$.
end for
Update weights $\omega \leftarrow \omega + \eta \cdot \triangle$, reset $\triangle = 0$.
Calculate two streams of evaluated deep networks including $V(S(t); \omega, \xi)$ and $A(S(t), A(t); \omega, \iota)$ and combine them as $Q(S(t), A(t); \omega, \xi, \iota)$ using (3.31).
Calculate target Q-value $Q_{target}(s)$ in target deep networks:
If $S(t+1) = S_{terminal}$
$Q_{target}(s) = r_s$
else
The target Q-value is
$Q_{target}(s) = r_s + \gamma \max_{A(t+1)} Q(S(t+1), A(t+1); \omega^{'}, \xi^{'}, \iota^{'})$.
Train evaluated deep networks to minimize the loss function $L(\omega)$.
Every C steps copy weights into target network periodically $\omega^{-} \leftarrow \omega$, and update target deep networks.
end for
end for

3.4.2 Prioritized Experience Replay

Prioritized experience replay [82] (PER) can make traditional experience replay more efficient and effective. The core idea of prioritized experience replay is that the learning agent can more effectively learn from some samples than others. In experience replay pool of deep reinforcement learning, some samples may be more or less redundant. In other words, these samples may not be useful to the learning agent. Therefore, the core idea of PER is not on random sampling, but on the priority of samples in experience replay pool. So this method is more efficient in finding the samples when the learning agent needs to learn.

PER uses temporal-difference(TD) error to evaluate the priority of the samples. If the TD-error of a sample is larger, the priority of this sample is higher than others, which means that this sample has a high priority to be learned by the agent. According to [82], the TD-error ρ_j can be depicted as:

$$\rho_j = R_j + \gamma_j Q_{target}\left(S_j, \arg\max Q(S_j, a)\right) - Q(S_{j-1}, A_{j-1}) \tag{3.27}$$

where R_j represents the reward value of sample j, and $Q(S_j, a)$ denotes the Q-value of sample j.

The greedy TD-error has some problems like the greedy policy is not very accurate and easy to be overfitting. In order to solve those problems, the probability of sampling j is:

$$P(j) = \frac{p_j^\tau}{\sum_k p_k^\tau} \tag{3.28}$$

where $p_j \geq 0$ represents the priority of sample j. The exponent τ denotes how much the prioritization is used.

After we get the priority of a sample j, we can get the importance-sampling weight

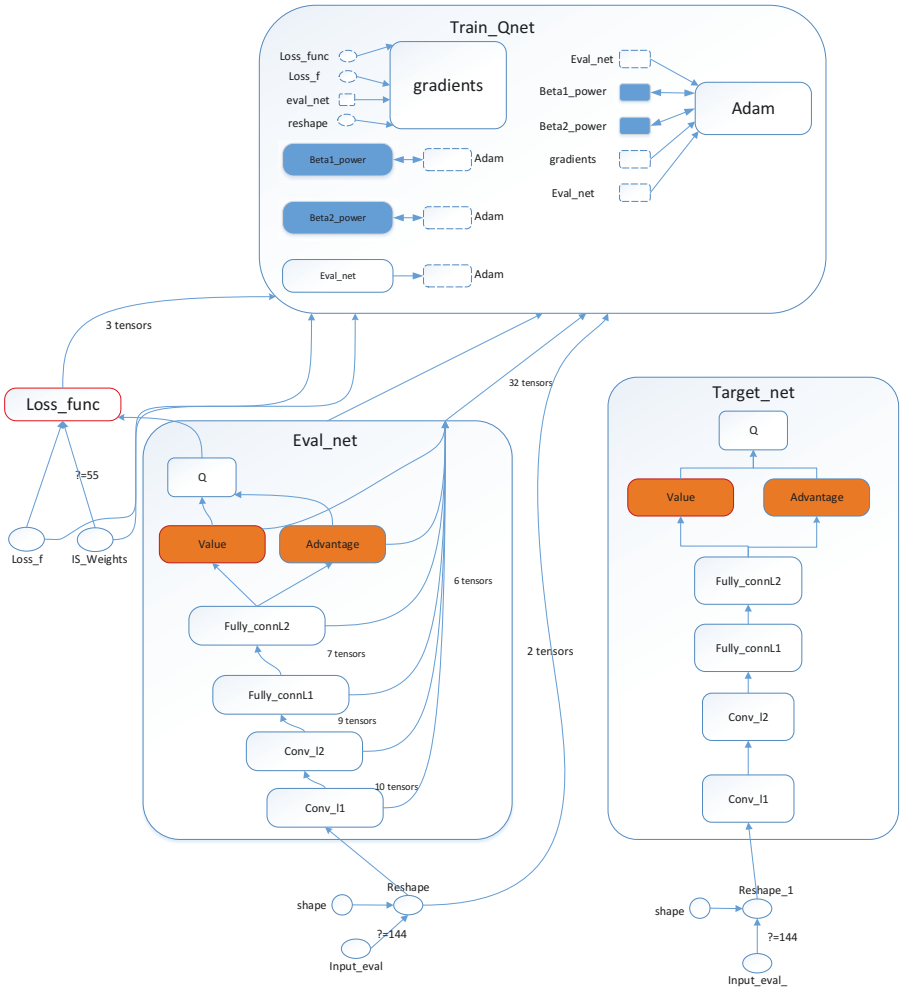

Figure 3.6: The TensorFlow graph in TensorBoard using CNN with PER.

(IS-weight) as:

$$w_j = \left(\frac{1}{G} \cdot \frac{1}{P(j)}\right)^{\kappa} \tag{3.29}$$

where κ is used to adjust the bias.

3.4.3 Dueling Deep Q-Learning with Prioritized Experience Replay Approach

In block-SDV, there are a lot of states. No matter what action the learning agent takes, it doesn't affect the next state transition. According to [2], dueling deep Q-network (DDQN) has better performance than nature deep Q-network. On the other hand, according to [1], experience replay mechanism uses a uniform sampling method. However, in block-SDV, some samples in the experience replay pool have a larger amount of information. For example, there is only one primary node in block-SDV, which means that only one state in $\zeta_a(t)$ has the highest reward. If using uniform sampling, the learning efficiency of the agent is not very well. Therefore, we use DDQN with PER in our proposed block-SDV. The workflow of DDRL is shown in Fig. 3.5.

In the DDQN, the value function $A(S(t), A(t))$ and advantage function $V(S(t), A(t))$ are aggregated as final output $Q(S(t), A(t))$. The dueling network architecture is shown in Fig. 3.6. The final Q-value under policy π at time slot t can be denoted as:

$$Q_\pi\left(S(t), A(t)\right) = V_\pi\left(S(t)\right) + A_\pi\left(S(t), A(t)\right) \tag{3.30}$$

In Algorithm 1, $V(S(t);\omega,\xi)$ is a scalar, and $A(S(t), A(t);\omega,\iota)$ is an $|\mathcal{A}|$ dimensional vector. ξ and ι are the parameters of two different streams.

Specifically, the Q-function of DDQL in block-SDV under policy π at time slot t can be defined as follows:

$$\begin{aligned} Q_\pi(S(t), A(t);\omega,\xi,\iota) = V_\pi(S(t);\omega,\xi) + \Bigg(& A_\pi(S(t), A(t);\omega,\iota) + \\ & \frac{1}{|A|}\sum_{A(t+1)} A_\pi(S(t), A(t+1);\omega,\iota)\Bigg) \end{aligned} \tag{3.31}$$

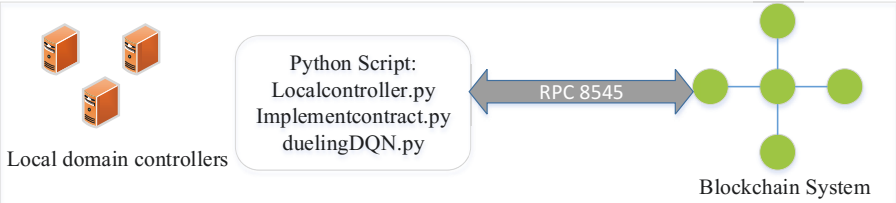

Figure 3.7: A simulation structure of proposed blockchain technology.

In Algorithm 1, the loss function $L(\omega)$ can be depicted as:

$$L(\omega) = E\left[Q_{target}(S(t)) - Q(S(t), A(t); \omega, \xi, \iota)\right] \tag{3.32}$$

3.5 Simulation Results and Discussions

In this section, we describe our simulation setup, configurations, and simulation results. We use computer simulation to evaluate the performance of our proposed scheme. First, we introduce our simulation setup, and then show the simulation results and corresponding discussions.

3.5.1 Simulation Setup

The simulation has been implemented in a GPU-based server with the processor of Intel(R) Core(TM) i7-6600 CPU and 16GB memory. The software environment that we use is TensorFlow 1.6.0 [71] with python 3.6 on Windows 10 64-bit operating system. These two simulation tools have been widely used in industry and academia. TensorFlow is an open source software library for machine learning, and it has been widely used to deploy new machine learning algorithms and experiments. Therefore, we can confirm that the system performance of our proposed block-SDV can estimate and approximate the performance in realistic networks.

First, we introduce the throughput model in the area control layer, in which we

Table 3.3: Simulation parameters

Simulation Parameter	Assigned value
Topology	Random and grid
Packet size	1024 bytes
IEEE 802.11 MAC	802.11p
Topology covered area in device layer	$5 \times 5km^2$.
Data rates	5.5 Mbps
Mobility	Static (none)
Number of vehicles	32
Network	OpenFlow
α	4MHz
β	0.05MHz
The block size	1Mb

assume that there are six areas in the device layer. As shown in Fig. 3.1, each area includes several SDN-enabled vehicles and one RSU. And some distributed SDN controllers are randomly deployed within the area control layer to manage the corresponding area. We assume that each vehicle's state can be trustworthy (trust level higher than 0.6), suspect (trust level between $[\theta, 0.6]$), and malicious (trust level lower than θ). In our simulation, we assume that the threshold of trust level θ is 0.5. Trustworthy vehicles are trusted nodes that aim to establish a secure routing path from the source to the destination. The proposed block-SDV provides a method for good entities to avoid working with suspected and malicious vehicles. Each vehicle has a possibility to remain its state unchanged or change its state at next time slot t. Therefore, we set the transition probability of each vehicle in each area. An example of the transition probability matrix for each vehicle in an area can be set as $\Upsilon = ((0.1, 0.3, 0.5, 0.1), (0.5, 0.2, 0.2, 0.1), (0.6, 0.2, 0.1, 0.1), (0.3, 0.6, 0.05, 0.05))$.

In our simulation, the trust feature of each node in the blockchain can be illustrated as trustworthy, suspect and un-trusted. The most trusted node can

be selected as the primary. The transition probability matrix of each node is $\Psi = ((0.6, 0.2, 0.2), (0.75, 0.1, 0.15), (0.5, 0.25, 0.25))$.

Each node in the blockchain has a possibility to become consensus node, which can be selected from each domain controller by the preference of votes. The voting feature of each blockchain node is high, medium and low. We set the transition probability matrix of each blockchain node as $\Omega = ((0.5, 0.3, 0.2), (0.8, 0.1, 0.1), (0.45, 0.15, 0.4))$.

The deep Q-network used in this chapter is the normal neural network with 8, 9, and 10 hidden layers and the convolution neural network (CNN). We use Tensorboard to show the architecture of deep Q-network using CNN. In our simulation, we use two convolution layers, two max-pooling layers, and two fully connected layers in an evaluation network and a target network. In our simulations, the architecture of the evaluation network is the same as the target Q-network. However, only the evaluation network is trained depending on the gradient descent method, and we replace the target Q-network by trained Q-value every 5 steps (after 200 steps). The values of the rest parameters are summarized in Table 3.2.

As shown in Fig. 3.7, each domain controller will run a blockchain node and will form the network. Meanwhile, the communication between each domain controller and blockchain system is done via *RPC* (remote procedure call). In our simulation, we need to establish our smart contract logic to the blockchain system. It contains a mapping for every account (inside the local domain controllers) to message, transactions and tags. Specifically, *implementcontract.py* is a Python script for sending our contract to the blockchain system; *localcontroller.py* is used to store transactions and to send these transactions to the blockchain system; *duelingDQN.py* aims to use dueling DQN with prioritized experience replay approach to get the block-SDV system state space and to learn an optimal policy in order to maximize the system throughput. In our simulation, there are four schemes simulated for the performance

comparison:

- Proposed DDQL-based scheme with PER using CNN. In this scheme, each learning agent selects most trusted vehicles, an appropriate number of consensus nodes, the most trusted consensus node in the blockchain system as the primary node for the consensus of the smart contract, and the virtualized computing server with better computing capability. The architecture of deep Q-network using in this scheme is CNN. Comparing with traditional deep Q-network, CNN can perform more efficient feature extraction, which improves training efficiency. Therefore, this scheme should have the best performance.

- Proposed DDQL-based scheme with PER using 8, 9, and 10 hidden layers neural network. This scheme also selects the most trusted controller, the most trusted node as the primary node, and a better computing capability of the server. The performance should be worse than the proposed scheme using CNN.

- Existing scheme 1 [43] with local computational capabilities, without the selection of trusted vehicles, and with traditional view changes. Meanwhile, the DDQL approach is also employed in this scheme. This scheme allows the connected vehicles to communicate with each other randomly without considering the trust features, and to select the primary node using traditional view changing protocol without considering the trust feature of each blockchain node. Moreover, this scheme does not use hierarchical network architecture. Compared with the DDQL-based scheme with PER, the advantage of using DDQL with PER can be shown.

- Existing scheme 2 [45] with local computational capabilities, without the selection of trusted vehicles, and with traditional view changes. Meanwhile, the DDQL approach is not employed in this scheme. This scheme allows domain

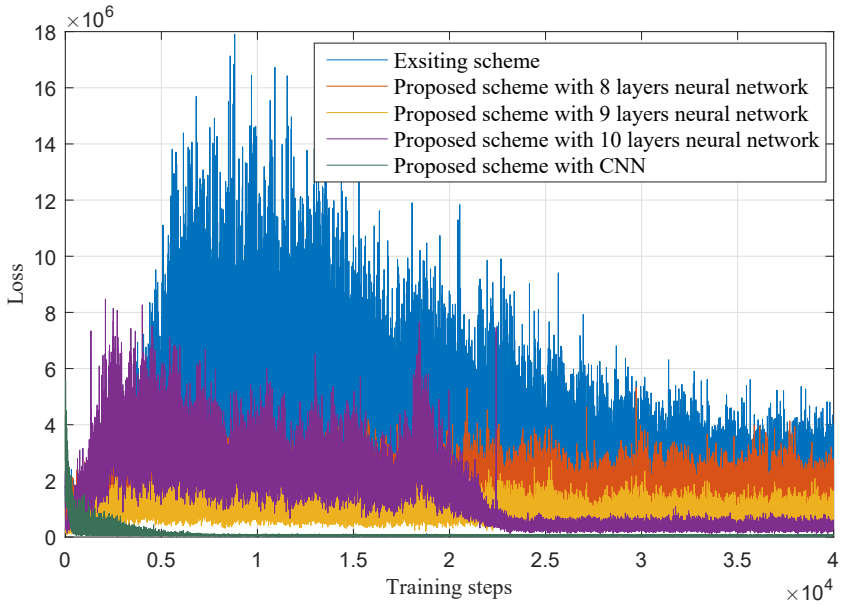

Figure 3.8: Training curves tracking the loss of block-SDV under different schemes.

controllers to access to the blockchain system randomly. Each domain controller will randomly select the blockchain nodes as consensus nodes. Compared with the DDQL-based scheme with PER, the advantage of using DDQL with PER can be shown.

3.5.2 Simulation Results

Fig. 3.8 shows the convergence performance comparison of different scenarios in the proposed schemes using dueling deep reinforcement learning with PER. The value of loss function $L(\omega)$ reflects the degree of Q-value fitting. At the beginning of training, deep neural networks (8, 9, 10 hidden layers and convolution neural network) do not have enough knowledge of the uncertain environment, and with the increasing experiences from the prioritized experience replay pool, the training loss is higher. When the agent learns enough experiences, the training loss decreases at the same time. Such increasing and decreasing performance of the training loss illustrates the

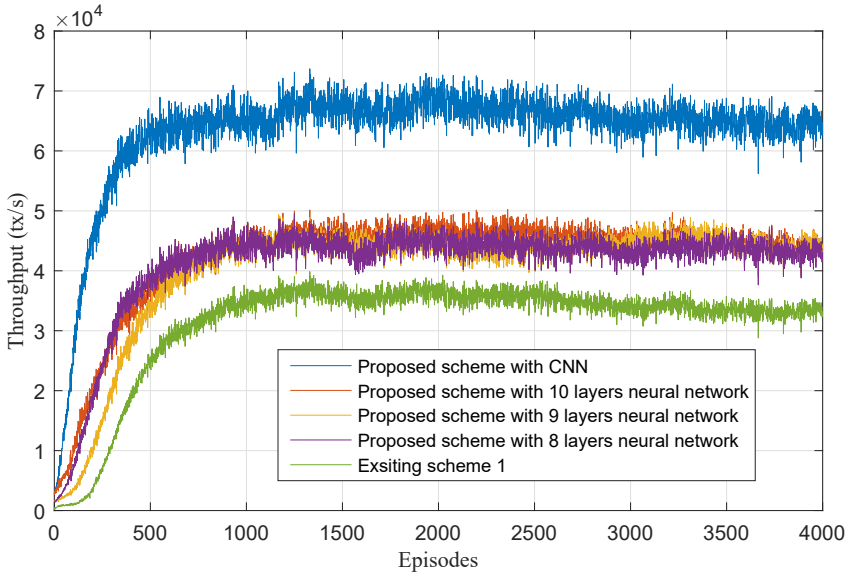

Figure 3.9: Training curves tracking the throughput of block-SDV under different schemes.

effectiveness of deep neural networks. Moreover, from Fig. 3.8, we can see that our proposed schemes are better than the existing scheme. This is because that as the number of deep neural network layers increases, the neural network has a higher degree of abstraction of features, which leads to better performance. Meanwhile, in our simulation, the importance of the samples is different. Some samples are important, and the priority is high. So we use prioritized experience replay method to improve the learning efficiency.

Fig. 3.9 shows the system throughput comparison under different schemes. In our simulation, each point is the average throughput per episode, and the agent uses Adamoptimizer [72]. As shown in this figure, we can also conclude that the system throughput of our proposed schemes is better than the existing scheme. There are two reasons for the higher throughput: 1) our proposed schemes use RBFT consensus mechanism. Comparing with PBFT, RBFT has added important transaction verification links, ensuring a consensus on the order of transaction execution while

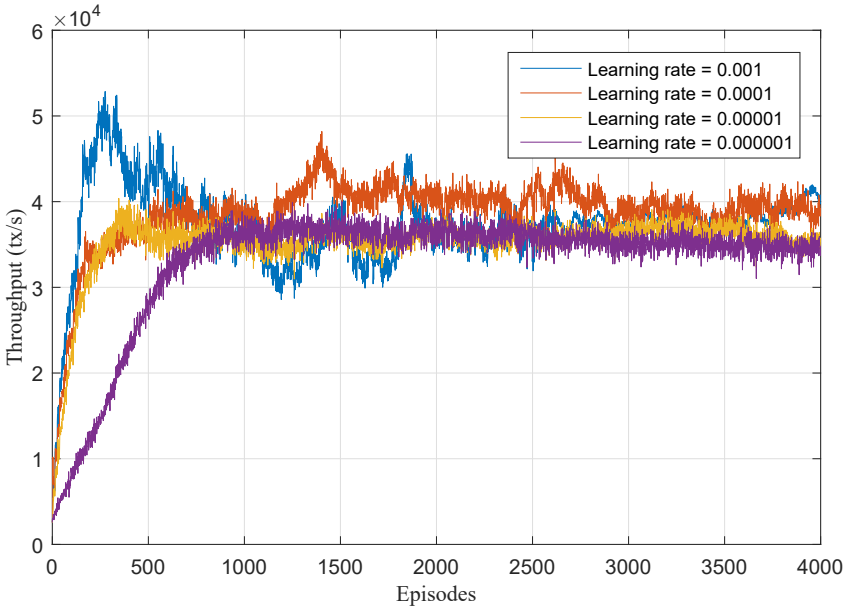

Figure 3.10: Training curves tracking the throughput of block-SDV under different learning rates.

ensuring consensus on the results of block verification, thus improving the consensus efficiency and system throughput; 2) since the existing scheme does not consider the trust feature of each vehicle, the malicious vehicles will degrade the system throughput. Meanwhile, this figure shows the convergence performance of DDQL with PER. At the beginning of training, DDQL with PER has some trials and errors. After increasing the training episodes, the system throughput becomes stable, which means that the agent has learned the optimal policies to maximize the long-term rewards.

Fig. 3.10 shows the throughput convergence performance of the proposed scheme using DDQL-based scheme using CNN with different learning rates. Learning rate determines the updating speed of weighted factor in deep Q-network. If the learning rate is too big, it will cause the result to exceed the optimal value. Otherwise, it will deeply interfere with the Q-network convergence. From this figure, when learning rates are 0.001 and 0.0001, the training curves are highly scaled, which lead to miss the global optimum. Compared with the two curves of yellow and purple, although the

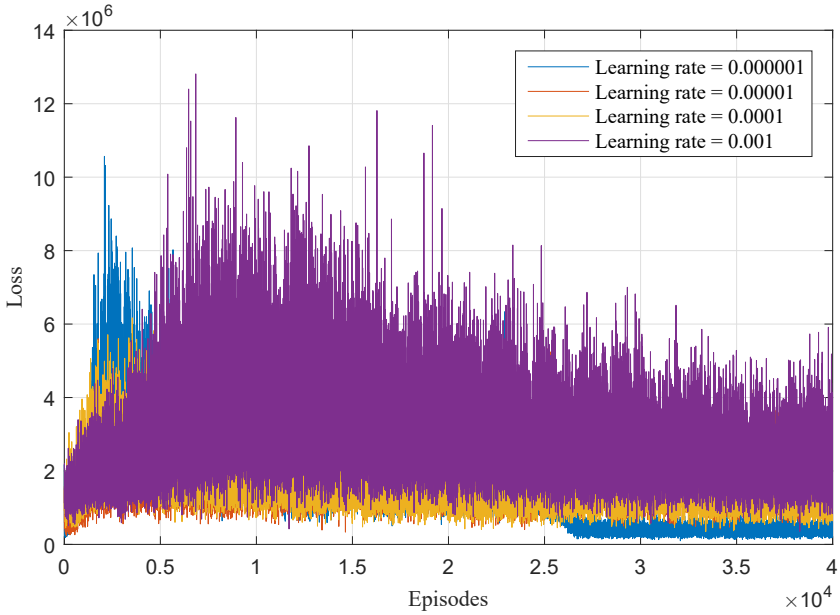

Figure 3.11: Training curves tracking the loss of block-SDV under different learning rates.

yellow curve has faster convergence speed, it does not very stable after convergence comparing with the purple curve. Therefore, we choose the learning rate as $1e^{-5}$ because of its acceptable convergence speed and better learning stability.

Fig. 3.11 shows the loss convergence performance of each proposed scheme using the deep neural network with ten hidden layers. Through this figure, we can see that the learning rate has effects on convergence performance. The learning rate means the stride to optimize the loss function. The bigger learning rate denotes longer learning steps and bigger vibration. When learning rate equal to 0.001, the loss has the biggest vibration. As the learning rate decreasing, the vibration of training curves decrease simultaneously. According to this figure, we choose the learning rate as $1e^{-5}$ in our simulation because it has the lowest vibration and final converge after 2500 episodes.

After the effective training using CNN and deep neural network, we use these results for further simulation. Fig. 3.12 shows the throughput comparison versus the number of controllers in the domain control layer. Specifically, this figure also

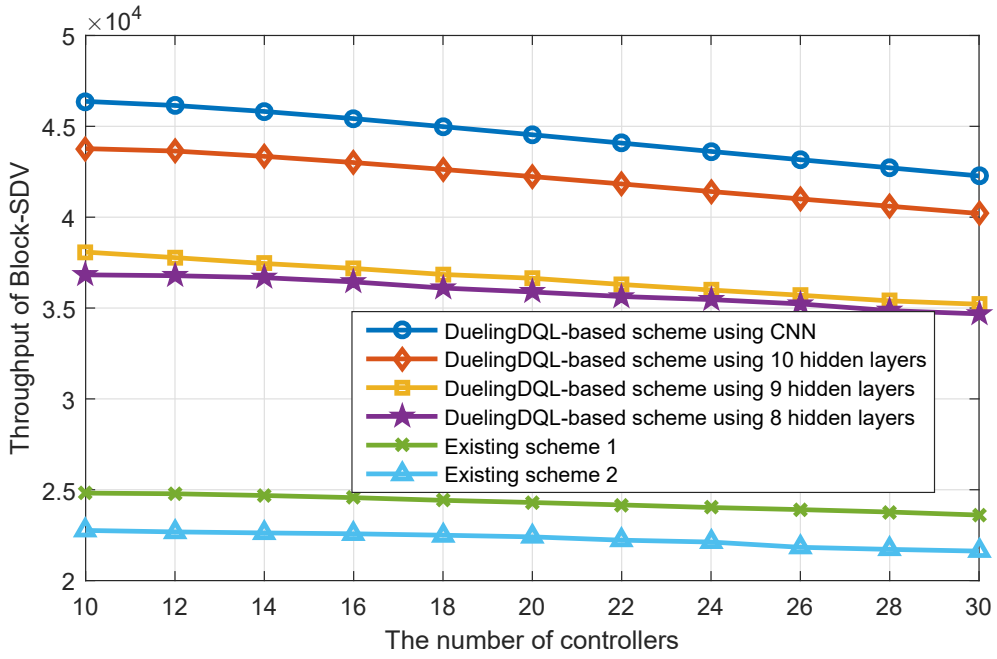

Figure 3.12: The throughput comparison versus the number of controllers.

can simulate the throughput performance in large realistic SDV environment because there are up to 30 domain controllers to interact with area control layer and the blockchain system. According to this figure, we can see that as the number of domain controllers increases, the system throughput decreases at the same time. This is because that more domain controllers need more computation resources to verify signatures and MACs. However, our proposed scheme, using hierarchical SDN control platform, RBFT consensus mechanism, and DDQL-based scheme with PER, has better performance as shown in the blue curve. Meahwhile, we can conclude that our proposed scheme has better performance in large SDV environment.

Fig. 3.13 shows the throughput comparison versus the number of consensus nodes in the blockchain. As shown in this figure, as the the number of consensus nodes increases, the system throughput decreases. This is because that more signatures and MACs are generated as the consensus nodes increase. These signatures and MACs need more CPUs to handle, resulting in decreased system throughput. However, our

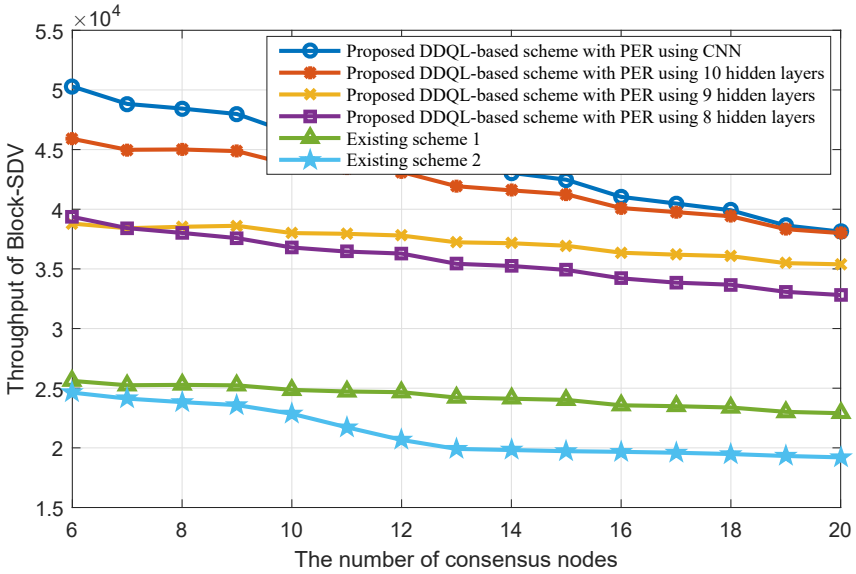

Figure 3.13: The throughput comparison versus the number of consensus nodes.

proposed scheme using CNN is still better than the existing schemes.

Fig. 3.14 shows the throughput comparison versus the batch size of each block. From this figure, we can see that the system throughput increases as the block enlarges to different sizes in our simulation. This is because the larger block size enables to contain more transactions to synchronize more local network events among domain controllers, which leads to the increase of system throughput. Meanwhile, our proposed scheme using CNN with PER is still better than the existing schemes.

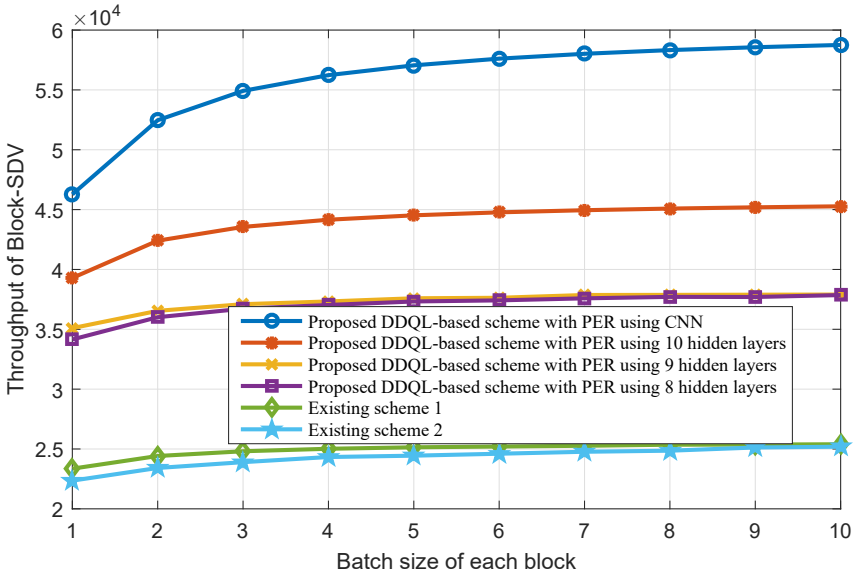

Figure 3.14: The throughput comparison versus the batch size of each block.

Chapter 4

Blockchain-based Mobile Edge Computing for Future Vehicular Networks: A Deep Compressed Neural Network Approach

In this chapter, we focus on distributed MEC servers and blockchain technology for VANETs. We propose a novel blockchain-based mobile edge computing for future VANETs (BMEC-FV). Considering the reliable features of VANET nodes, limited communication time, and high computing loads, we consider using a permissioned blockchain system in our proposed BMEC-FV framework. Specifically, the significance of the BMEC-FV architecture is the secure data storage and sharing in vehicular edge networks.

4.1 System Model

In this section, we introduce our blockchain-based MEC for VANETs. First, we present the network model and then illustrate the communication model and computing model. Here, a discrete-time slotted system is applied, where time is partitioned into discrete time periods $\mathcal{T} = \{1, 2, ..., t, ..., T\}$.

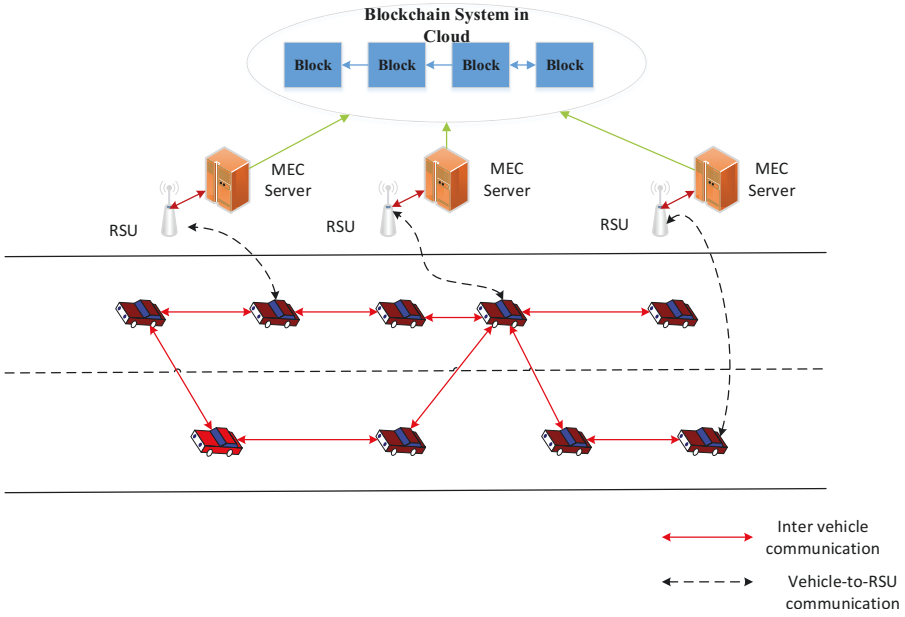

Figure 4.1: Blockchain-based MEC in VANETs.

4.1.1 Network Model

As shown in Fig. 4.1, we assume that there are R RSUs and C block producers in our BMEC-FV system. It should be specially pointed out that the block producers are selected by multiple RSUs, and each RSU is connected to the MEC server. Here, the block producer can be expressed as $\mathcal{B} = \{B_1, B_2, ..., B_c, ..., B_C\}$. In our proposed architecture, U vehicles around multiple RSUs communicate with each other, denoted by $\mathcal{V} = \{V_1, V_2, ..., V_u, ..., V_U\}$. In order to ensure the safety of communication between vehicles, the trust value of each vehicle is considered to be the computing task that needs to be calculated by each RSU. In the VANET environment, the trust calculation between vehicles needs to be offloaded to the adjacent RSU. In the BMEC-FV architecture, we do not consider the local computation of each vehicle.

The consensus mechanism is a very important part of the blockchain. In general, the consensus mechanism ensures that all blockchain nodes comply with the protocol rules and ensure that all transactions are performed in a reliable manner in our

proposed BMEC-FV system. In this chapter, we use redundant Byzantine fault tolerance (RBFT) as the consensus rule for the BMEC-FV system. In short, RBFT has a significant performance improvement over traditional PBFT. At the same time, RBFT ensures the security and liveness of the system when the number of malicious nodes is less than $\frac{C-1}{3}$. In BMEC-FV, there are C block producers that generate L blocks in sequence at time interval T (seconds). Meanwhile, we assume that the size of each block is S_b. In particular, the number of blocks L and the size S_b will change with time interval T, because the VANET environment is a dynamic time-varying system.

Overall, the entire BMEC-FV includes the vehicle networking communication model and the blockchain system (BCs) model, and we introduced MEC and computational offloading techniques to ensure the high throughput of the entire system. The MEC server is the core of the whole system, which contains a total of five computing offload tasks, 1) the inter-vehicle trust calculation is transferred to the RSUs; 2) the RSUs submit offloading requests to BCs; 3) the block producer performs the smart contracts; 4) the BCs reach consensus among multiple block producers, and 5) BCs sends the requests back to RSUs.

4.1.2 Communication Model

In BMEC-FV system, there are three wireless links: 1) vehicle-to-vehicle communication links; 2) the trust information from vehicles to RSUs, and 3) message delivery among multiple RSUs.

For the vehicle-to-vehicle communication links, we define the communication model in the VANET environment. Data transmission is affected by the communication quality of communication links on the Internet of Vehicles. Therefore, in order to measure the effectiveness of data transmission between vehicles, we have designed a

communication model between vehicles to estimate the wellness of a node's forwarding behavior. In order to check the behavior of the forwarding node (transmitting vehicle) to forward the data packet, the receiving node (receiving vehicle) needs to perform auxiliary communication with the help of the neighboring node or use the broadcast function of the wireless channel to passively monitor whether the transmitting node forwards its data packet. The former's auxiliary communication method will consume more resources. In contrast, the latter passive monitoring method consumes fewer resources and is less susceptible to incorrect or fake information sent by neighbors.

Consider a receiving node and observe how its sending node forwards packets sent and received between them in the window of P packets. Within each window, the receiving node calculates the percentage of packets in the P packets that are not forwarded by the sending node. This percentage is attributable to the wireless effect and malicious nodes effect. We use this to confirm the forwarding efficiency of each node and use it as a measure of the quality of the communication link between vehicles. Let A be the rate of network volatility and B be the malicious nodes drop rates. Wireless network fluctuations and malicious nodes are the culprits in network packet loss. If the communication link between the transmitting node and the receiving node has high volatility, the phenomenon actually observed by the receiving node will be a serious network packet loss [83].

Let us model the percentage of forwardable packets that can be observed as $(1-\gamma)$ and γ can be interpreted as the perceived packet loss rate. Note that $(1-\gamma)$ can be $(1-A)$ or $(1-A)(1-B)$ depending on whether the node is a malicious node. In order to estimate the reliability of the transmitting node, the number of valid data packets required by the transmitting node to forward a data packet can be quantified. This concept is the same as the general concept of ensuring that packets are received

by the receiving node, such as ETX [84] for collecting tree protocols. Let α denote the estimated node reliability metric, which is defined as follows according to our previous definition:

$$\alpha = \frac{1}{(1-A)(1-B)} = \frac{1}{1-\gamma} \tag{4.1}$$

In fact, the receiving node will need to measure α on the packet window. In this chapter, we record the moving average of α and we expect that there will be no large fluctuations. Therefore, we use an exponentially weighted moving average (EWMA) to track the α value, which is defined as follows:

$$\alpha_i = \beta \frac{1}{(1-\gamma_i)} + (1-\beta)\alpha_{i-1} \tag{4.2}$$

where the subscript i reflects the window for calculating the packet loss rate γ_i, and the coefficient β indicates the rate at which the weighting decreases. Now we can define the reliability estimates of conventional VANET nodes and malicious nodes as α_{A_i} and α_{AB_i}, respectively.

$$\alpha_{A_i} = \beta \frac{1}{(1-A_i)} + (1-\beta)\alpha_{A_{i-1}} \tag{4.3}$$

$$\alpha_{AB_i} = \beta \frac{1}{(1-A_i)(1-B_i)} + (1-\beta)\alpha_{AB_{i-1}} \tag{4.4}$$

After defining the VANET communication model, we need to define the communication model between the vehicles and the RSUs. As stated earlier, the reliability of each vehicle is transmitted to the RSUs. In this process, we need to consider how to

characterize the state of the time-varying channel. Therefore, we introduce a finite-state Markov channel model. Here, we consider that the channel state between the vehicle and the RSU depends on the received signal-to-noise ratio (SNR). We divide the received SNR into K levels $\mathcal{E} = \{E_0, E_1, ..., E_k, ..., E_{K-1}\}$, where E_0 and E_{K-1} represent the known minimum and maximum SNR values, respectively. At the same time, the finite state Markov chain can be expressed as $\mathcal{M} = \{M_0, M_1, ..., M_k, ..., M_{K-1}\}$. We assume that at time t, the channel state is δ_t and we have $\delta_t = M_k$.

Since the communication channel between the vehicle and each RSU is a time-varying channel, at each moment, the channel state may change over time. Therefore, we need to define the Markov state transition probability of the channel state δ_t. Let $p_{E_y,E_z} = Pr\{\delta_{t+1} = M_{E_z} | \delta_t = M_{E_y}\}$ be the state transition probability of the channel state M_{E_y} at time t hopping to the channel state M_{E_z} at time $t+1$, where $E_y, E_z \in \mathcal{E}$. Hence, the $K \times K$ state transition probability matrix can be depicted as $\mathcal{P} = [p_{E_y,E_z}]_{K \times K}$.

Based on the previously defined finite Markov channel model, we define the SNR between the vehicle and the RSUs as δ^{V_u,B_c}, and the SNR of the data transmission between the RSUs as $\delta^{B_c,B_{c'}}$.

In this chapter, we use the Orthogonal Frequency Division Multiple Access (OFDMA) channel access technology. We divide the access channel S sub-channels, each of which can be used by VANET nodes or RSUs to transmit data. We assume that the total bandwidth of the channel is H and the bandwidth of each sub-channel is H'. Each VANET node can occupy S_{V_n} sub-channels to transmit reliable messages to the RSUs. Similarly, each RSU can occupy S_{B_c} sub-channels to transmit information to each other. It is worth noting that the sum of the channels allocated by all the VANET nodes and RSUs cannot exceed the total S sub-channels. Therefore, we have

a constraint as follows:

$$\sum_{u=1}^{U} S_{V_u} + \sum_{c=1}^{C} S_{B_c} \leq S \tag{4.5}$$

According to Shannon's theorem, the information transmission rate $I_{V_n B_c}$ between a VANET node and a RSU can be expressed as:

$$I_{V_u,B_c} = S_{V_u} H' \log_2(1 + \delta^{V_u,B_c}) \tag{4.6}$$

Similarly, the information transmission rate $I_{B_c,B_{c'}}$ between RSUs can be expressed as:

$$I_{B_c,B_{c'}} = S_{B_c} H' \log_2(1 + \delta^{B_c,B_{c'}}) \tag{4.7}$$

As we know, the data transfer rate of each RSU is the sum of the rate at which the VANET node transmits to the RSU and the rate at which other RSUs transmit to the RSU. We assume that there are u VANET nodes transmitting reliable information to this RSU, and there are c' RSUs transmitting information to this RSU. The data transmission rate of this RSU should not exceed the receiving data transmission rate. Therefore, we have a second constraint as follows:

$$\begin{aligned} \sum_{u \in U} S_{V_u} H' \log_2(1 + \delta^{V_u,B_c}) + \sum_{c' \in C} S_{B_c} H' \log_2(1 + \delta^{B_{c'},B_c}) \\ \geq I_{B_c,B_{c''}} \end{aligned} \tag{4.8}$$

4.1.3 Computation Model of BCs

In BMEC-FV, BCs has many different computing tasks. For example, they may execute smart contracts, generate and verify block signatures and reach consensus

between blocks. When each RSU receives a message (a VANET node sends a reliable message to the RSU), it contains some specific computing tasks. Let $C_o = \{d_o, q_o\}$ denote a computing task related to message o, where d_o means the size of message o, and q_o means the required number of CPU cycles to complete this task. We do not know the computational resources for BMEC-FV at the next time slot. As mentioned earlier, each RSU is connected to the MEC server. We assume that at time t, the computing capacity of each MEC server is $\epsilon_{B_c}(t)$. There are T time slots, which start when each RSU issues un-validated block and terminates when the RSU receives a reply of a validated block.

For simplicity, at time t, we assume that the computing capacity assigned to the message o is customized. Since the computing capacity of each MEC server changes over time, we need to define a transition probability matrix for each MEC server's computing capacity. Let g_{xy} represent the transition probability of $\epsilon_{B_c}(t)$ moving from state x to next state y. Hence, the $X \times X$ transition probability matrix for g_{xy} is $[g_{xy}]_{X \times X}$, where $g_{xy} = Pr\{\epsilon_{B_c}(t+1) = y | \epsilon_{B_c}(t) = x\}$.

Based on the above model, the execution time of computing task C_o at RSU B_c can be calculated as:

$$T_{B_c} = \frac{q_o}{\epsilon_{B_c}(t)} \tag{4.9}$$

4.2 The Theoretical Computing Capability Analysis of the Blockchain System in BMEC-FV

In this section, we introduce the adaptive consensus mechanism of the blockchain system based on RBFT.

4.2.1 Overview of RBFT

The existing BFT algorithm (prime, Aardvark, Spinning) does not really perform Byzantine fault tolerance, mainly because there is a "primary" for sorting. If the primary becomes a malicious node, the performance of the entire system will drop significantly and will not be discovered. RBFT proposes a new mechanism. In RBFT, a multi-core machine parallel executes multiple PBFT protocol instances, only the results of the primary instance will be actually executed. Each protocol instance will be monitored for performance and will be compared with the primary instance. If the performance of the primary node degrades significantly, this node will be considered as a malicious node. RBFT will initiate the replacement process. According to [75], the performance degradation of RBFT compared with presence of BFT attacks is up to 3%, while the other protocols are: Prime (78%), Aardvark (87%), and Spinning (99%). RBFT has been used in some real scenarios, such as Hyperchain project [85]. It is hosted by the Linux Foundation, and develop applications with a modular architecture. Thus, we consider that the RBFT can be used in some real scenarios when deploying MEC servers.

4.2.2 Theoretical Analysis

As shown in Fig. 4.2, there are six steps when blockchain nodes try to achieve consensus with each other. In each step, the cost can be divided into two parts: the transmission latency of the primary node and the transmission latency of each replica. We consider that each consensus node executing smart contracts, generating one message authentication codes (MACs), verifying one MAC and one signature needs ν, ζ, ζ, and ξ cycles, respectively.

We assume that there are D transactions transferred to the blockchain system from each vehicle, and a fraction g of transactions sent by the RSUs are correct [80].

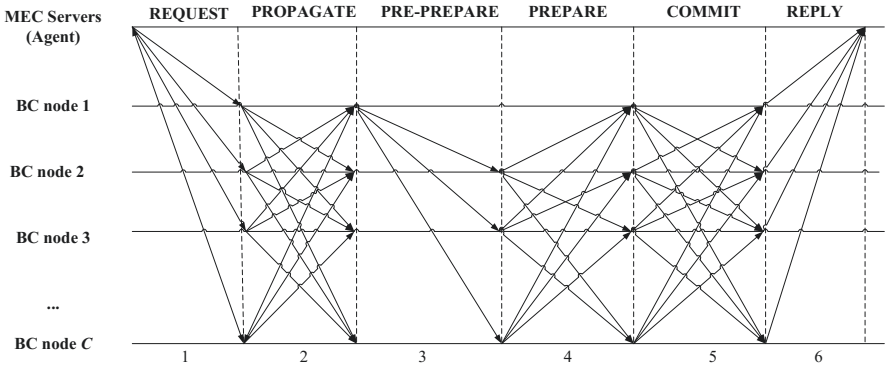

Figure 4.2: The consensus procedures inside of the blockchain.

Meanwhile, we assume that the average size of each transaction is σ.

1) **Theoretical analysis for all the MEC servers sending request messages to all the nodes**: In this phase, a vehicle sends its transactions (reliable request message) $\left\langle \langle transactions, m\rangle_{\chi_m}, m \right\rangle_{\chi_{\vec{m}}}$ to all the consensus nodes. The transmission latency from each vehicle to each replica in the BCs can be defined as:

$$T_{rel}^{L} = \max_{V_u \in V} \{ \frac{\sigma}{I_{V_u, B_r}} \} \tag{4.10}$$

where I_{V_u,B_r} denotes the transmission rate from the vehicle to each replica in the BCs.

Once each replica receives the message, a consensus node first verifies the MAC. If the MAC is valid, then the signature of the message is verified by this consensus node. If the MAC is invalid, further request messages will not be processed. Hence, the cost of the primary node is $(\frac{D}{g}+1)(\zeta+\xi)$. We assume that computation capacity assigned to one reliable message is ϵ_{B_p}. The computation latency of the primary node can be calculated as:

$$C_{rel}^{L} = \frac{(\frac{D}{g}+1)(\zeta+\xi)}{\epsilon_{B_p}} \tag{4.11}$$

and the computation latency of each replica is the same as C_{rel}^{L}.

2) **Theoretical analysis for the propagation procedure**: In this phase, each node propagates the receiving request message to all other consensus nodes once the request has been verified. This process ensures that every correct node will eventually receive a request as long as at least one correct node receives the request before. For example, once a node c receives the $PROPAGATE$ message $\left\langle PROPAGATE, \langle transactions, m\rangle_{\chi_m}, c\right\rangle_{\chi_{\vec{c'}}}$ from node c', node c first verifies the MAC authenticator. If the MAC is valid, then c verifies the signature of the receiving transactions. If the signature is valid, node c sends the $PROPAGATE$ message to all other nodes. The transmission latency in this phase can be defined as:

$$T_{pro}^{L} = \max_{B_p, B_r \in B} \{\frac{D \times \sigma}{I_{B_p, B_r}}\} \tag{4.12}$$

As defined above, a total of C block producers participate in the RBFT consensus process. RBFT can tolerate the Byzantine error of f nodes at most, where $f = (C-1)/3$. Meanwhile, according to [85], the number of nodes that can guarantee consensus is $e = (C+f+1)/2$. Hence, the cost of consensus node is $(\frac{D}{g} + \frac{C+2}{3})(\zeta + \xi)$. The computation latency in this phase is:

$$C_{pro}^{L} = \max_{B_p, B_r \in B} \{\frac{(\frac{D}{g} + \frac{C+2}{3})(\zeta + \xi)}{\epsilon_{B_p, B_r}}\} \tag{4.13}$$

3) **Theoretical analysis for the pre-prepare procedure**: when the $(e-1)$ replicas receive requests, the primary sends $PRE-PREPARE$ messages $\langle PRE-PREPARE, Pr, m, H(m)\rangle_{\chi_{\vec{Pr}}}$ authenticated by MAC for replicas, where Pr is the primary's ID and $H(m)$ is the hashed results of the issued block. The primary node generates $(e-1)$ MACs for all other replicas. In this phase, the primary node

packs the transactions into blocks according to the received time sequence, and performs verification, executes the smart contract, and finally writes the sequenced transaction information together with the verification results into the $PRE-PREPARE$ message and broadcasts it to all consensus nodes. We assume that the block size is S_b and the maximum number of transactions that can be stored in each block is S_b/σ. The transmission latency in this phase is:

$$T_{pre}^{L} = \max_{B_p, B_r \in B} \{ \frac{S_b}{I_{B_p, B_r}} \} \tag{4.14}$$

The cost of the primary node is $\frac{2C-2}{3}\zeta$. After each replica receiving the block, the computation cost of each replica is $\frac{S_b}{\sigma}(\nu+\zeta+\xi)+\zeta$. Hence, the computation latency of each replica is:

$$C_{pre}^{L} = \max_{B_r \in B} \{ \frac{\frac{S_b}{\sigma}(\nu+\zeta+\xi)+\zeta}{\epsilon_{B_r}} \} \tag{4.15}$$

4) **Theoretical analysis for the prepare procedure**: When a replica receives a $PRE-PREPARE$ message from the primary node, it verifies the validity of the MAC. It then replies a $PREPARE$ message $\langle PREPARE, Pr, m, H(m), r \rangle_{\chi_{\vec{r}}}$ to all other replicas, where r is the replica's ID and each replica generates $(e-1)$ MACs. Meanwhile, each replica needs to follow the reception of $2f$ matching $PREPARE$ messages from distinct replicas that are consistent with the proper $PRE-PREPARE$ message. In this phase, the transmission latency can be calculated as:

$$T_{par}^{L} = \max_{B_r, B_{r'} \in B} \{ \frac{S_b}{I_{B_r, B_{r'}}} \} \tag{4.16}$$

In this phase, the primary node needs to verify $2f$ matching $PREPARE$ messages,

and each replica generates $e-1$ MACs for all other consensus nodes as well as verifies $2f$ MACs. Thus, the cost of the primary node is $\frac{2C-2}{3}\zeta$. The cost of each replica can be denoted as $\frac{4C-4}{3}\zeta$. Hence, the computation latency in this phase is:

$$C_{par}^{L} = \max_{B_r \in B}\{\frac{\frac{4C-4}{3}\zeta}{\epsilon_{B_r}}\} \tag{4.17}$$

5) **Theoretical analysis for the commit procedure**: After finishing processing the $PREPARE$ message, each replica sends a commit message that is authenticated by a MAC authenticator. For example, a replica c generates $(C-1)$ MACs to all other replicas, and it verifies one MAC to check the validity of the incoming $COMMIT$ message $\langle COMMIT, Pr, m, H(m), r\rangle_{\chi_{\vec{r}}}$. After receiving incoming $2f+1$ matching $COMMIT$ messages from distinct replicas, c' will verify the smart contract. If valid, it will append this block to the blockchain.

In this phase, the primary node generates $C-1$ MACs and verifies $2f+1$ MACs. Each replica also needs to generate $e-1$ MACs for all other nodes and to verify $2f+1$ MACs. In this phase, the transmission latency is:

$$T_{com}^{L} = \max_{B_r, B_{r'} \in B}\{\frac{S_b}{I_{B_r, B_{r'}}}\} \tag{4.18}$$

Therefore, the costs of the primary node and each replica are $\frac{4C-1}{3}$. The computation latency in this phase is:

$$C_{com}^{L} = \max_{B_r \in B}\{\frac{\frac{4C-1}{3}\zeta}{\epsilon_{B_r}}\} \tag{4.19}$$

6) **Theoretical analysis for the reply procedure**: After the request message operation is executed, each node sends reply message $\langle REPLY, bl, c\rangle_{\chi_{c,m}}$ to the primary node, where bl is an ID of a valid block and c is a node's ID. After receiving

the reply messages, the primary node needs to verify $2f$ valid reply messages and $\frac{S_b}{\sigma}$ transactions in a block in order to confirm that the smart contract is executed successfully in each replica. If $\frac{S_b}{\sigma}$ transactions match with the $f+1$ reply messages, the primary node accepts this block and gets the information inside the block to update its network view. In this phase, the transmission latency can be defined as:

$$T_{rep}^{L} = \max_{B_r, B_p \in B} \{ \frac{S_b}{I_{B_r, B_p}} \} \tag{4.20}$$

In this phase, the primary and replicas need to generate $\frac{s_b}{\sigma}$ MACs for the transactions' operation. Therefore, the costs of the primary node and each replica are $\frac{s_b}{\sigma}\zeta$. Hence, the computation cost in this phase is:

$$C_{rep}^{L} = \max_{B_r \in B} \{ \frac{\frac{s_b}{\sigma}\zeta}{\epsilon_{B_r}} \} \tag{4.21}$$

Here, we need to point out that these variables (B_r, B_p) belong to the set $\mathcal{B}$. Meanwhile, these variables (ϵ_{B_p}, ϵ_{B_p,B_r}, ϵ_{B_r}) are all time-varying variables. As shown in equation 4.9, these variables follow the transition probability matrix $[g_{xy}]_{X \times X}$.

4.3 Performance Analysis of BMEC-FV

In this section, we analyze the performance of the BMEC-FV system. For the VANET MEC system, the QoS of the VANET node depends on the delay time that the VANET node sends the requests to RSUs until it receives the blockchain processing result. For the performance analysis of the blockchain system, we mainly discuss three aspects: 1) the throughput of the blockchain system; 2) the blockchain processing request delay, and 3) decentralization. In this section, we mainly analyze these aspects.

4.3.1 Performance of MEC in VANETs

For the VANET MEC system, its performance mainly depends on the three stages of message transmission: 1) the VANET node transmits the request to the RSU MEC server; 2) the selected block producer performs the calculation offload task (smart contract), and 3) the task processing result is sent back to the VANET node. As shown in Section 4.2.2, the transmission latency from the VANET node to each RSU is T_{rel}^{L}.

Based on the previous analysis, the primary node will package the offloading request into blocks in the pre-prepare phase, and perform verification to execute the smart contract. Therefore, the cost of the primary node handling a block request at this stage is $\frac{S_b}{\sigma}(\nu + \zeta + \xi)$. Furthermore, the processing delay of the primary node includes two parts: execution delay and queue delay. The execution delay can be defined as follows:

$$T_{ed} = \frac{S_b(\nu + \zeta + \xi)}{\epsilon_{B_p}\sigma} \tag{4.22}$$

As described before, there are L blocks in the RBFT consensus mechanism. The queuing delay is:

$$T_{qu} = \frac{1}{2}(L-1)\frac{S_b(\nu + \zeta + \xi)}{\epsilon_{B_p}\sigma} \tag{4.23}$$

In this analysis, we do not consider the sending back procedure as in [86], because the size of the output may be much smaller than the input data. Therefore, the processing delay of the primary node is $T_{pd} = T_{rel}^{L} + T_{ed} + T_{qu}$.

4.3.2 Performance Analysis of BCs

1) *Throughput of BCs*: Based on Section 4.2.2, we analyzed the cost of each primary node and each replica. For a block, the cost of each primary node is the sum of the costs of the primary node in each RBFT phase, which is defined as follows:

$$C_p = (\frac{2D}{g} + 3C + \frac{S_b}{\sigma})\zeta + (\frac{2D}{g} + \frac{C+5}{3} + \frac{S_b}{\sigma})\xi + \frac{S_b}{\sigma}\nu \tag{4.24}$$

Similarly, the cost of each replica is:

$$C_r = (\frac{2D}{g} + 3C - 1 + \frac{2S_b}{\sigma})\zeta + (\frac{2D}{g} + \frac{C+5}{3} + \frac{S_b}{\sigma})\xi + \frac{S_b}{\sigma}\nu \tag{4.25}$$

We assume that multi-core computation modules run in parallel on distinct cores. Each core has a computation speed of η Hz. Meanwhile, we consider that the probability of the primary node becoming a malicious node is h, $h \in (0, 1]$. The malicious node degrades the system performance because it reduces the system throughput. Therefore, the throughput $BT_{blockchain}$ of the blockchain system is at most:

$$BT_{blockchain} = \min\left[\frac{h\eta}{C_p}, \frac{h\eta}{C_r}\right] tx/s \tag{4.26}$$

Here, the $BT_{blockchain}$ is used to estimate the number of transactions that the blockchain system can process per second.

2) *The blockchain processing request delay*: In order to ensure the security of transactions in the block, it is critical to ensure that the transaction is not arbitrarily altered or revoked. Handling request latency is the time to ensure that transactions in the block are not maliciously tampering or revoked. In the actual VANET environment, if the node request is maliciously falsified or revoked, the entire processing

request delay will become longer, which cannot be tolerated by the low latency required in a VANET environment. Therefore, we consider the processing request delay as an important factor in measuring the performance of the BMEC-FV system.

We assume that the timeout of transmission latency in RBFT consensus is ϕ_{tl}. Hence, the transmission latency T_{tl} can be defined as follows:

$$\begin{aligned} T_{tl} = & \min\{T_{rel}^{L}, \phi_{tl}\} + \min\{T_{pro}^{L}, \phi_{tl}\} + \min\{T_{pre}^{L}, \phi_{tl}\} + \\ & \min\{T_{par}^{L}, \phi_{tl}\} + \min\{T_{com}^{L}, \phi_{tl}\} + \min\{T_{rep}^{L}, \phi_{tl}\} \end{aligned} \tag{4.27}$$

Similarly, we assume that the timeout of computation delay is ϕ_{cl}. The computation latency T_{cl} can be depicted as follows:

$$\begin{aligned} T_{cl} = & \min\{C_{rel}^{L}, \phi_{cl}\} + \min\{C_{pro}^{L}, \phi_{cl}\} + \min\{C_{pre}^{L}, \phi_{cl}\} + \\ & \min\{C_{par}^{L}, \phi_{cl}\} + \min\{C_{com}^{L}, \phi_{cl}\} + \min\{C_{rep}^{L}, \phi_{cl}\} \end{aligned} \tag{4.28}$$

The processing request delay T_{prd} includes two parts: the transmission latency T_{tl} and computation latency T_{cl} in the BCs.

$$T_{prd} = T_{tl} + T_{cl} \tag{4.29}$$

3) *Decentralization*: For BMEC-FV, we use the Gini coefficient [87] to characterize the degree of decentralization of the blockchain. The Gini coefficient refers to a common indicator used internationally to measure the income gap of a country or region. The Gini coefficient usually uses the Lorenz curve to measure inequality. In this chapter, the number of blocks produced by each block producer at time t is C, which is denoted as $C_{B_c}(t) = \{C_{B_c}(1), C_{B_c}(2), ..., C_{B_c}(t), ..., C_{B_c}(T)\}$. Hence, the Gini

coefficient of decentralization in BCs at time slot t can be defined as follows:

$$\begin{aligned} G(t) =& \frac{\sum_{B_c \in B} \sum_{B_{c'} \in B} |C_{B_c}(t) - C_{B_c}(t^{'})|}{2 \sum_{B_c \in B} \sum_{B_{c'} \in B} C_{B_c}(t)} \\ =& \frac{\sum_{B_c \in B} \sum_{B_{c'} \in B} |C_{B_c}(t) - C_{B_c}(t^{'})|}{2C \sum_{B_c \in B} C_{B_c}(t)} \end{aligned} \tag{4.30}$$

It is worth noting that the value range of $G(t)$ is $[0, 1]$. The smaller the value of $G(t)$, the better the decentralization of the blockchain nodes. Conversely, it represents the higher degree of centralization of the blockchain nodes. In particular, when the Gini coefficient of the blockchain system is equal to zero, it represents that the number of blocks produced by each block producer in the consensus phase is equal. In other words, this situation represents absolute fairness in the blockchain system. If the Gini coefficient is equal to 1, it means that only one block producer produces a block. In other words, the blockchain system is now a centralized system, which is contrary to the nature of the blockchain. Therefore, we need to avoid the Gini coefficient being too large. In order to ensure the decentralization of the blockchain system, we define the following restrictions:

$$G(t) \leq \kappa \tag{4.31}$$

where κ represents the threshold of decentralization in terms of $G(t)$.

4.4 Problem Formulation

In Section 4.3, we have already proposed the performance analysis of BMEC-FV. In order to optimize system performance, we jointly consider three different scenarios. In this section, we formulate this joint optimization problem as a Markov decision process, in which we define system state space, action space, and reward function.

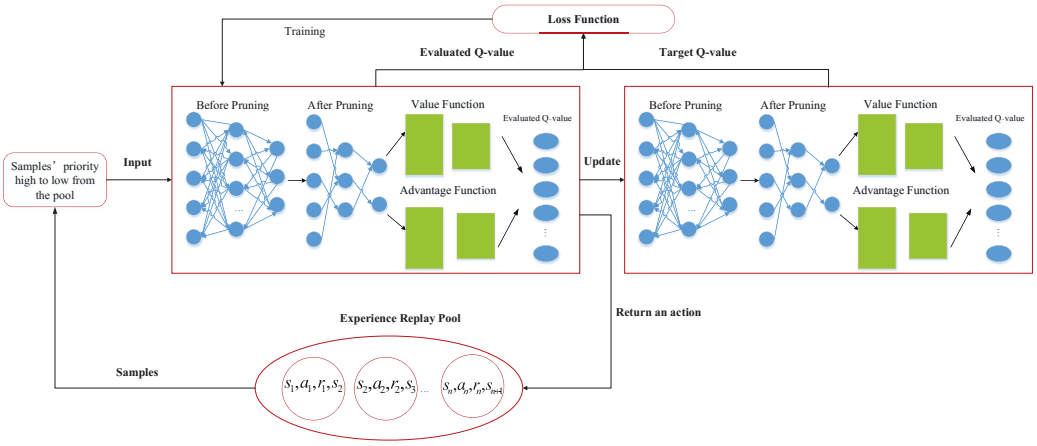

Figure 4.3: Blockchain-based MEC in VANETs.

4.4.1 System State

Let $\mathcal{S} = \{s(t), t \in \mathcal{T}\}$ represent the state space of BMEC-FV. Here, $s(t)$ denotes the system state at time slot t. The state of the system includes the reliability of each VANET node, the SNR between the VANET node and the RSU, the trust feature of each block producer, SNR between the RSUs, and the computing capacity assigned to different messages by RSUs. Hence, the system state space can be defined as follows:

$$\begin{aligned} S(t) = \{&\alpha^1_{AB_i}(t), \alpha^2_{AB_i}(t), ..., \alpha^u_{AB_i}(t), ..., \alpha^U_{AB_i}(t); \\ &\boldsymbol{\delta_{V_1}}(t), \boldsymbol{\delta_{V_2}}(t), ..., \boldsymbol{\delta_{V_u}}(t), ..., \boldsymbol{\delta_{V_U}}(t); \\ &\boldsymbol{\delta_{B_1}}(t), \boldsymbol{\delta_{B_2}}(t), ..., \boldsymbol{\delta_{B_c}}(t), ..., \boldsymbol{\delta_{B_C}}(t); \\ &\boldsymbol{\phi_{B_1}}(t), \boldsymbol{\phi_{B_2}}(t), ..., \boldsymbol{\phi_{B_c}}(t), ..., \boldsymbol{\phi_{B_C}}(t); \\ &\boldsymbol{\epsilon_{B_1}}(t), \boldsymbol{\epsilon_{B_2}}(t), ..., \boldsymbol{\epsilon_{B_c}}(t), ..., \boldsymbol{\epsilon_{B_C}}(t)\} \end{aligned} \tag{4.32}$$

where $\boldsymbol{\delta_{V_u}}(t) = \{\delta_{V_u,B_c}(t), B_c \in \mathcal{B}\}$, $\boldsymbol{\delta_{B_c}}(t) = \{\delta_{B_c,B'_c}(t), B_c \in \mathcal{B}\}$, and $\boldsymbol{\epsilon_{B_c}}(t) = \{\epsilon_{B_c}(t)\}$. Here, we need to point out that due to the fault tolerance of RBFT, each block producer participating in consensus may become a Byzantine node (notation

h in equation 4.26), so we consider the trust feature $\phi_{B_C}(t)$ of each block producer. Considering the time correlation of the trust feature of each block producer, we apply the Markov state transition matrix to model the trust of block producers. For any block producer c, let j_{xy} represents the state transition matrix of $\phi_{B_C}(t)$ from state x to the next state y. Therefore, the $J \times J$ state transition matrix of $\phi_{B_C}(t)$ can be expressed as $[j_{xy}]_{J\times J}$, where $j_{xy} = Pr\{\phi_{B_c}(t+1) = y | \phi_{B_c}(t) = x\}$.

4.4.2 System Action

In our proposed BMEC-FV, the action space of the system includes the selection of trusted VANET nodes, the allocation of subchannels, the block size, and the number of blocks successfully generated by the blockchain nodes. Let $\mathcal{A} = \{A(t), t \in \mathcal{T}\}$ denote the action space of BMEC-FV, which can be defined as follows:

$$\begin{aligned} A(t) = \{&A_{\alpha^1_{AB_i}}(t), A_{\alpha^2_{AB_i}}(t), ...A_{\alpha^U_{AB_i}}(t); \\ &S_{V_1}(t), S_{V_2}(t), ..., S_{V_u}(t), ..., S_{V_U}(t); \\ &S_{B_1}(t), S_{B_2}(t), ..., S_{B_c}(t), ..., S_{B_C}(t); S_b(t); L(t)\} \end{aligned} \tag{4.33}$$

where $A_{\alpha^u_{AB_i}}(t)$ represents the selection of a trusted VANET node. The reliability of each VANET node should be higher than the threshold ξ, which can ensure the trusted communication between the VANET node and the RSUs. After establishing trusted communication links, $S_{V_u}(t)$ and $S_{B_c}(t)$ represent sub-channel allocations between the trusted VANET nodes and the RSUs, respectively. In particular, equation (4.5) states that the sub-channels to which both are allocated cannot exceed the total channel capacity S. $S_b(t)$ represents the block size generated at time t, and $L(t)$ represents the number of successfully generated blocks at time t. In particular, the replicas generate blocks in order, and only one primary node exists while replicas spawn blocks. In other words, for the primary node $B_p(t)$ at time t, the number of blocks produced by

the replicas is $L(t)$.

4.4.3 Reward Function

In this research, we need to solve the joint optimization problem of RSU MEC and the blockchain system in the BMEC-FV system. We aim to maximize the performance of the entire system through decision making in action space and state space. Therefore, the reward function can be defined as:

$$
\begin{aligned}
&P1: \max_{\mathcal{A}} Q(\mathcal{S}, \mathcal{A}) \\
&C1: A_{\alpha^u_{AB_i}}(t) \geq \xi, \quad \forall u \in U \\
&C2: T^l_{prd} \leq T_{\max}, \quad \forall l \in L(t) \\
&C3: G(t) \leq \kappa \\
&C4: \sum_{u=1}^{U} S_{V_u} + \sum_{c=1}^{C} S_{B_c} \leq S \\
&C5: \sum_{u \in U} I_{V_u, B_c} + \sum_{c' \in C} I_{B_{c'}, B_c} \geq I_{B_c, B_{c''}}
\end{aligned}
\tag{4.34}
$$

where $Q(\mathcal{S}, \mathcal{A}) = \sum_{t=1}^{t=T-1} \gamma^t r(t)$ denotes the long-term reward of the BMEC-FV system. Here, γ^t is a discount factor that approaches zero when t is large enough. The learning agent senses the system state space $S(t)$ from the environment at time slot t, then the agent outputs a policy π that determines which action should be executed from the system action vector $A(t)$. The reward value will be returned to the learning agent at a time slot t. Then the system state space changes to the next state $S(t+1)$, and the learning agent outputs new policy and gets a new immediate reward $r(t+1)$.

According to (4.34), $A_{\alpha^u_{AB_i}}(t)$ in the constraint $C1$ represents the reliability of the VANET node. RSUs prefer to select VANET nodes above the threshold ξ for channel allocation. T^l_{prd} in the constraint $C2$ represents the processing request delay for the

lth block, which should not exceed the timeout $T_{\max}$. $C3$ determines the degree of decentralization of the blockchain nodes. $C4$ shows that the allocation of sub-channels should not exceed the total channel capacity S. $C5$ illustrates the qualification of the data transmission rate. We can see that the performance of the BMEC-FV system is very low when $C1 \sim C5$ is not satisfied. Therefore, we define the immediate reward function as follows:

$$r(t) = \begin{cases} \Upsilon BT_{blockchain} + \Omega \dfrac{1}{T_{pd}}, & C1 \sim C5 satisfied, \\ \\ 0, & otherwise \end{cases} \tag{4.35}$$

where Υ and Ω are two weighted factors ($\Upsilon, \Omega \in [0, 1]$) corresponding to the BCs and MEC servers in RSUs. And there is $\Upsilon + \Omega = 1$. It is worth noting that these two weight coefficients are dynamically changing, which means that the BMEC-FV system dynamically selects these two parts.

4.5 Deep Compressed Neural Network for BMEC-FV

In this section, we focus on the application of the deep compressed neural network in our proposed BMEC-FV system and the complexity analysis of the algorithm. In particular, we use the algorithm of Dueling Deep Q-learning (DDQL) to establish a deep convolutional neural network for the BMEC-FV system.

4.5.1 Background of Deep Compressed Neural Network

Traditional neural networks have a large number of hierarchies and nodes, so it is extremely important to consider how to reduce the amount of memory and computation required by them, especially for real-time applications such as online learning and incremental learning. In addition, the recent popularity of smart wearable devices has also provided researchers with the opportunity to deploy deep learning applications on portable devices with limited resources (memory, CPU, power, bandwidth, etc.). Efficient deep learning methods can significantly affect distributed systems, embedded devices, and Field Programmable Gate Array (FPGA) for artificial intelligence. If you pruned some redundant weights, it would save about 75% of the parameters and 50% of the calculation time [2].

We divide these methods into four categories: parametric pruning and sharing, low-rank decomposition, migration/compression convolution filters, and knowledge refinement. Methods based on parameter pruning and sharing focus on exploring redundant parts of the model parameters and attempting to remove redundant and unimportant parameters. The method based on the low-rank coefficient technique uses matrix/tensor decomposition to estimate the most informative parameters in the deep CNN. A special structure convolution filter is designed based on the method of transferred/compact convolutional filters to reduce the complexity of storage and calculation. Knowledge milling, on the other hand, learns a refinement model that trains a more compact neural network to reproduce the output of a large network.

In this chapter, we focus on the pruning method to prune the parameters of the BMEC-FV neural network system.

4.5.2 Dueling Deep Q-Learning Based on the Pruning Method For BMEC-FV

A deep Q-learning (DQL) concept introduced by [49] aims to solve the instability of a traditional Q-network. There are two important improvements compared with the traditional reinforcement learning method: *experience replay* and *target Q-network*. Experience replay stores trained data and then randomly samples from the pool. Therefore, it reduces the correlation of data and improves the performance compared with the previous reinforcement learning algorithms [81]. Meanwhile, *target Q-network* is another improvement in the deep Q-learning. That is, it calculates a target Q-value using a dedicated target Q-network, rather than directly using the pre-updated Q-network. The purpose of this is to reduce the relevance of the target calculation to the current value.

Although deep Q-learning makes big improvements compared with traditional reinforcement learning methods, many researchers still make great efforts for even greater performance and higher stability. Here, we introduce a recent improvement: dueling deep Q-learning [2]. The core idea of this approach is that it does not need to estimate the value of taking each available action. For some states, the choice of action makes no influence on these states themselves. Thus, the network architecture of DDQL can be divided into two main components: value function and advantage function. The value function is used to represent how good it is to be in a given state, and the advantage function can measure the relative importance of a certain action compared with other actions. After value function and advantage function are separately computed, their results are combined back to the final layer to calculate the final Q-value. The mechanism of DDQL would lead to better policy evaluation compared with DQL.

In the DDQN, the value function $A(S(t), A(t))$ and advantage function

$V(S(t), A(t))$ are aggregated as final output $Q(S(t), A(t))$. The dueling network architecture based on a pruning method is shown in Fig. 4.3. The final Q-value under policy π at time slot t can be denoted as:

$$Q_{\pi}\left(S(t), A(t)\right) = V_{\pi}\left(S(t)\right) + A_{\pi}\left(S(t), A(t)\right) \tag{4.36}$$

Algorithm 3 Dueling deep Q-network based on pruning method for BMEC-FV

Initialization
  Initialize experience replay memory O, memory size Z, and minibatch q.
  Initialize the evaluated deep Q-network with initial
  weights. ω
  Initialize the target deep Q-network with weights ω^{-}.
for episode k = *1, 2, 3,...,K* **do**:
  Initialize the beginning state $S(t)$
  Initialize the action set $A(t)$
  Initialize the time step t, exponents φ and τ
  for t = *1,2,3,...,T* **do**
    Choose a random probability p
    If $(p \geq \psi)$,
      $A^{*}(t) = \arg\max Q(S(t), A(t), \omega)$,
    Otherwise,
      Randomly choose an action$A(t) \neq A^{*}(t)$,
    obtain the immediate reward $r(t)$ and next state
    $S(t+1)$.
    Store experience $(S(t), A(t), r(t), S(t+1))$ into
    experience replay memory O
    for $j = 1$ to q **do**
      For each filter $\mathcal{F}_{x,y}$, calculate the sum of its absolute kernel weights
      $kw_y = \sum_{g=1}^{n_x} \sum |\mathcal{K}_g|$.
      Sort the filters by kw_y.
      Prune m filters with the smallest sum values and their corresponding
      feature maps. The kernels in the next convolutional layer corresponding to
      the pruned feature maps are also removed.
      A new kernel matrix is created for both the xth and $x+1th$ layers, and
      the remaining kernel weights are copied to the new model.
    end for
    Calculate two streams of evaluated deep networks
    including $V(S(t); \omega, \varrho)$ and $A(S(t), A(t); \omega, \iota)$ and combine
    them as $Q(S(t), A(t); \omega, \varrho, \iota)$ using (4.37).
    Calculate target Q-value $Q_{target}(s)$ in target deep
    networks:
    If $S(t+1) = S_{terminal}$
      $Q_{target}(s) = r_s$
    else
      The target Q-value is
      $Q_{target}(s) = r_s + \gamma \max_{A(t+1)} Q(S(t+1), A(t+1); \omega^{'}, \varrho^{'}, \iota^{'})$.
    Train evaluated deep networks to minimize the loss
    function $L(\omega)$.
    Every C steps copy weights into target network
    periodically $\omega^{-} \leftarrow \omega$, and update target deep
    networks.
  end for
end for

In Algorithm 3, $V(S(t);\omega,\varrho)$ is a scalar, and $A(S(t),A(t);\omega,\iota)$ is an $|\mathcal{A}|$ dimensional vector. ϱ and ι are the parameters of two different streams.

Specifically, the Q-function of DDQL based on the pruning method in BMEC-FV under policy π at time slot t can be defined as follows:

$$\begin{aligned} Q_{\pi}(S(t),A(t);\omega,\varrho,\iota) =& V_{\pi}(S(t);\omega,\varrho)+\Bigg(A_{\pi}(S(t),A(t);\omega,\iota) \\ &+\frac{1}{|A|}\sum_{A(t+1)}A_{\pi}(S(t),A(t+1);\omega,\iota)\Bigg) \end{aligned} \tag{4.37}$$

In Algorithm 3, the loss function $L(\omega)$ can be depicted as:

$$L(\omega)=E\left[Q_{target}(S(t))-Q(S(t),A(t);\omega,\varrho,\iota)\right] \tag{4.38}$$

4.5.3 Complexity Analysis

Next, we perform algorithm complexity analysis on the BMEC-FV system. In the practical VANET MEC system, BMEC-FV has a very complicated system state space and behavior space. Therefore, we perform a complexity analysis on the DDQL based on the pruning method applied to this system.

Theorem 1. *In a practical scenario, the computational complexity of the proposed DDQL based on the pruning algorithm is* $\mathcal{O}(S^{(u+C)})$ *or* $\mathcal{O}(u+C)^{S-u-C}$.

Proof of Theorem 1. In order to prove the above theorem and thus analyze the complexity of the proposed algorithm, the size of the system state function and the space of each state vector must be considered [88]. Thus, based on the definition of the preceding action space, BMEC-FV needs to update the spectrum allocation of each VANET node and RSU according to time changes. At the same time, the block size and number of blocks produced by each block producer are also associated with the functions of the channel vectors. This is because the transactions in each block

contain information about the assigned channel.

For each state, its associated actions are a function of vehicle confidence, channel association vector, block size, and a number of generated blocks. However, the reliable state of multiple VANET nodes of the BMEC-FV system, the number of possible channel associations between the VANET node and the RSUs is much larger than the possible block size levels and the number of blocks generated. Moreover, the RSU will preferentially select a highly trusted VANET node to allocate channels. Therefore, the channel correlation number of the trusted VANET node and the RSUs can be focused on analyzing the convergence complexity of the proposed training algorithm only by the law of large numbers. Therefore, when the BMEC-FV system updates C block producers and u trusted VANET nodes with S subchannels, the computational complexity of the proposed algorithm is $\mathcal{O}(S^{(u+C)})$. In this chapter, we assume that the number of subchannels is greater than the total number of trusted VANET nodes and RSUs, i.e., $S > u + C$. Therefore, from another perspective, the computational complexity can also be expressed by $\mathcal{O}(u + C)^{S-u-C}$. This completes the proof.

From Theorem 1, it can be concluded that the convergence speed of the proposed DDQL based on a pruning algorithm is closely related to the state space dimension. It is important to note here that there is a trade-off between the computational complexity of the proposed DDQL training algorithm and the resulting network performance [88]. It is worth mentioning that the complexity of the algorithm proposed in this chapter can be ignored in this research because the training process is the most expensive and is carried out off-line.

Theorem 2. *For the proposed DDQL based pruning algorithm, the algorithm complexity is significantly reduced because we remove the convolution kernel that has little impact on the accuracy of the volume and neural network.*

Proof of Theorem 2. The network overhead of CNN in different applications is accompanied by an increase in parameter storage costs and computational complexity. Recent efforts to reduce these costs involve pruning and compressing the weights of different layers, and without compromising accuracy. However, the amplitude-based weighted pruning mainly reduces the parameter amount of the fully connected layer and does not reduce the computational cost of the convolution layer due to the irregularity of the pruning. In the convolutional neural network of the BMEC-FV system, we directly remove the convolution kernel that has little impact on CNN accuracy. By removing these convolution kernels of the network and their feature maps, the computational cost and algorithm complexity can be greatly reduced. This method does not result in a sparse join mode compared to weight pruning, so it does not need to specifically support sparse convolution libraries.

As described in Algorithm 3, we assume that n_x is the number of input channels for the xth convolutional layer, and $\frac{h_x}{w_x}$ is the width and height of the input feature map. The convolution layer converts the feature map whose input dimension is $E_x \subset R^{n_x \times h_x \times w_x}$ into an output feature graph whose dimension is $E_{x+1} \subset R^{n_{x+1} \times h_{x+1} \times w_{x+1}}$, which can be directly used as the input of the next convolutional layer. Here, the dimension of the convolution kernel is $n_{x+1} \times n_x \times k \times k$, and the number of MAdds of the convolutional layer is $n_{x+1} \times n_x \times k^2 \times h_{x+1} \times w_{x+1}$. In the convolutional neural network of BMEC-FV, a convolution kernel is clipped, and a corresponding feature map is also clipped accordingly, reducing the $n_x \times k^2 \times h_x \times w_x$ sub-operation. At the same time, the reduction in the number of output channels leads to a reduction in the dimension of the convolution channel of the next layer, which reduces the $n_{x+2} \times k^2 \times h_{x+2} \times w_{x+2}$. We can see that after the convolution kernel of a current convolutional layer is clipped, the convolution kernel parameters retained by this layer are not only reduced in the dimension of the number of convolution kernels. At the same time, the

reserved convolution kernel is also associated with the number of convolution kernels from which the previous convolutional layer was clipped, thus indirectly causing the reduction of the layer convolution kernel in the channel dimension. In particular, $\mathcal{F}_{x,y}$ represents a convolution kernel of the xth layer, and we characterize the convolution kernel in each layer by $\sum|\mathcal{F}_{x,y}|$ (the sum of the absolute values of the weighted values in a convolution kernel - the L1 regular term) importance. In summary, the pruning method we use can reduce the complexity of the algorithm. Theorem 2 is proved.

4.6 Simulation Results and Discussions

In this section, we describe our simulation setup, configurations, and simulation results. We aim to evaluate the performance of our proposed scheme. First, we introduce our simulation setup, and then show the simulation results and corresponding discussions.

4.6.1 Simulation Setup

The simulation has been implemented in a GPU-based server with the processor of Intel(R) Core(TM) i7-6600 CPU and 16GB memory. The software environment that we use is TensorFlow 1.6.0 [71] with python 3.6 on Windows 10 64-bit operating system. These two simulation tools have been widely used in industry and academia. TensorFlow is an open-source software library for machine learning, and it has been widely used to deploy new machine learning algorithms and experiments. Therefore, we can confirm that the system performance of our proposed BMEC-FV can estimate and approximate the performance in realistic networks.

In the BMEC-FV simulation environment, there are 4 RSUs and the number of VANET nodes is 32. Each RSU is equipped with a MEC server. In the simulation,

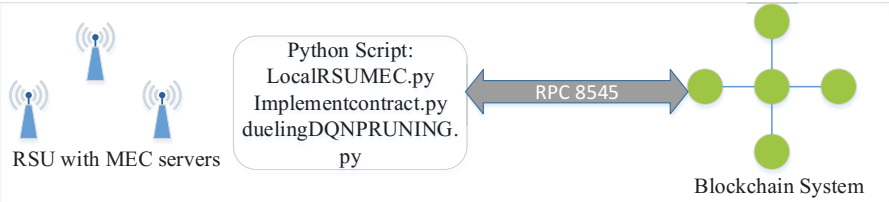

Figure 4.4: A simulation structure of proposed BMEC-FV.

we consider that these 4 RSUs are all block producers. For each VANET node, the reliability of each node changes with time. Therefore, the reliability of each node can be trustworthy (trust level higher than 0.6), suspect (trust level between $[\xi, 0.6]$), and un-reliable (trust level lower than ξ). In our simulation, we assume that the threshold of trust level ξ is 0.5. Trustworthy vehicles are trusted nodes that aim to establish a secure communication among multiple RSUs and VANET nodes. The proposed BMEC-FV provides a method for good entities to avoid working with suspected and malicious vehicles. Each vehicle has a possibility to remain its state unchanged or change its state at next time slot t. Therefore, we set the transition probability of each vehicle. An example of the transition probability matrix for each vehicle in an area can be set as $\Lambda = ((0.1, 0.3, 0.5, 0.1), (0.5, 0.2, 0.2, 0.1), (0.6, 0.2, 0.1, 0.1), (0.3, 0.6, 0.05, 0.05))$.

The computing capability of each RSU MEC server can be set as high level, medium level, and low level. the transition probability of each server can be defined as $\mathcal{Q} = g_{xy} = ((0.7, 0.1, 0.2), (0.65, 0.15, 0.2), (0.35, 0.5, 0.15))$.

In our simulation, the trust feature of each node in the blockchain can be illustrated as trustworthy, suspect and un-trusted. The most trusted node can be selected as the primary. The transition probability matrix of each node is $\Psi = ((0.6, 0.1, 0.2, 0.1), (0.65, 0.1, 0.15, 0.1), (0.5, 0.25, 0.15, 0.1), (0.6, 0.1, 0.1, 0.2))$

For channel state, the state of each channel follows the MDP. In the simulation, we assume four different scenarios, each with different levels of SNR values, labeled

Table 4.1: Simulation parameters in BMEC-FV

Simulation Parameter	Assigned value
VANET Topology	Random and grid
Packet size	1024 bytes
IEEE 802.11 MAC	802.11p
Topology covered area in device layer	$5 \times 5km^2$
Data rates	5.5 Mbps
Mobility	Static (none)
Number of vehicles	32
Total channel number	10
Weight for VANET environment Ω	0.5
Weight for VANET environment Υ	0.5
The block size	1Mb
Mini-batch size	32

SNR^1, SNR^2, SNR^3, and SNR^4. In the first scenario, SNR^1 can be divided into three levels: low ($Z_l = 1$), medium ($Z_l = 5$), and high ($Z_l = 9$). The state transition matrix can be expressed as $\mathcal{P}_1 = ((0.1, 0.4, 0.5), (0.3, 0.5, 0.2), (0.4, 0.2, 0.4))$. In the second scenario, the value of SNR^2 is $\{1, 5, 9, 15\}$ and the state transition matrix is $\mathcal{P}_2 = ((0.15, 0.25, 0.3, 0.3), (0.25, 0.35, 0.2, 0.2), (0.4, 0.15, 0.25, 0.2),$
$(0.3, 0.2, 0.25, 0.25))$. In the third scenario, the value of SNR^3 is $\{2, 7, 14, 20, 26\}$ and transition probability matrix is $\mathcal{P}_3 = ((0.1, 0.2, 0.4, 0.15, 0.15), (0.3, 0.2, 0.1, 0.2, 0.2),$
$(0.25, 0.25, 0.2, 0.15, 0.15), (0.2, 0.1, 0.1, 0.25, 0.25),$
$(0.1, 0.35, 0.25, 0.2, 0.1))$. In the fourth scenario, the value of SNR^4 is $\{2, 7, 14, 20, 26, 33\}$ and transition probability matrix is $\mathcal{P}_4 = ((0.2, 0.1, 0.4, 0.15, 0.15), (0.1, 0.2, 0.3, 0.2, 0.2),$
$(0.25, 0.25, 0.15, 0.2, 0.15), (0.1, 0.1, 0.2, 0.25, 0.25),$
$(0.1, 0.35, 0.25, 0.2, 0.1))$. Meanwhile, the other related parameters in our simulation are shown in Table 4.1.

The deep Q-network used in this chapter is the normal neural network with 10 hidden layers and the convolution neural network (CNN). Specifically, we using the pruning method on CNN in order to improve the training efficiency. In our simulation,

We use three convolution layers, three max-pooling layers, and two fully connected layers in an evaluation network and a target network. Specifically, the plan for pruning in CNN is to start pruning in 2000 steps, stop at 4000 steps, and once every 100 steps. In our simulation, the pruning method is to add a binary mask variable to each selected layer for pruning. Specifically, the shape and the weight tensor shape of the layer are exactly the same. The mask variable determines which weights participate in the weight update. The mask update algorithm needs to inject a special operator for the TensorFlow training calculation graph and sort the current layer weights by the absolute value. For the weights whose amplitude is less than a certain threshold, the corresponding mask value is set to 0. The back-propagation gradient also passes through the mask variable, and the masked weight (the mask is 0) does not get the updated value in the back-propagation step. We use this method to achieve the pruning of the deep Q-network parameters. In our simulations, the architecture of the evaluation network is the same as the target Q-network. However, only the evaluation network is trained depending on the gradient descent method, and we replace the target Q-network by trained Q-value every 5 steps (after 200 steps).

As shown in Fig. 4.4, each RSU with the MEC server will run a blockchain node and will form the network. Meanwhile, the communication between each RSU and blockchain system is done via *RPC* (remote procedure call). In our simulation, we need to establish our smart contract logic to the blockchain system. It contains a mapping for every account (inside the local MEC servers) to message, transactions and tags. Specifically, *implementcontract.py* is a Python script for sending our contract to the blockchain system; *localRSUMEC.py* is used to store transactions and to send these transactions to the blockchain system; *duelingDQNpruning.py* aims to use dueling DQN with pruning method to get the BMEC-FV system state space and to learn an optimal policy in order to maximize the system performance.

In our simulation, there are four schemes simulated for the performance comparison:

- Proposed DDQL-based scheme with pruning method using CNN. In this scheme, each learning agent (RSU with MEC server) selects the most reliable vehicles, an appropriate number of sub-channels allocation, the optimal block size, and the number of consecutively produced blocks for each replica. The architecture of deep Q-network using in this scheme is CNN with pruning method. Comparing with traditional deep Q-network, CNN with pruning can perform more efficient feature extraction, which improves training efficiency. Therefore, this scheme should have the best performance.

- Proposed DDQL-based scheme without pruning method using CNN and 10 hidden layers neural network. This scheme also selects the most reliable vehicles, an appropriate number of sub-channels allocation, the optimal block size, and the number of consecutively produced blocks for each replica. Since this scheme does not use the pruning method, the performance should be worse than the proposed scheme using CNN.

- Existing scheme 1 [73] with local computational capabilities, without the selection of reliable vehicles, random block size, and a fixed block number. Meanwhile, the DDQL approach without the pruning method using traditional deep neural network is also employed in this scheme. This scheme allows the connected vehicles to communicate with each RSU randomly without considering the reliable features. Compared with the DDQL-based scheme with pruning method, the advantage of using DDQL with pruning can be shown.

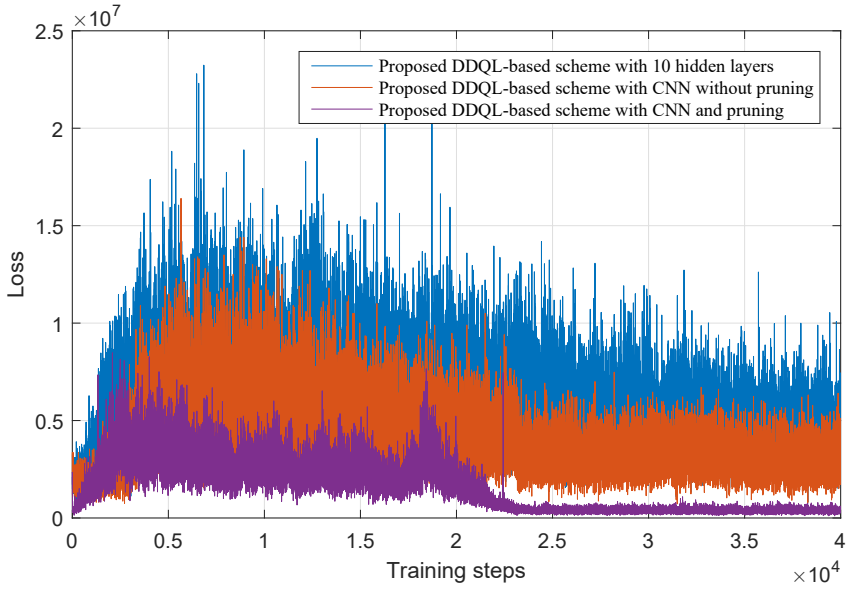

Figure 4.5: Training curves tracking the loss of BMEC-FV under different schemes.

- Existing scheme 2 [89] with local computational capabilities, without the selection of reliable vehicles, random block size, and a fixed block number. Meanwhile, the DDQL approach is not employed in this scheme. This scheme uses the random selection algorithm (RSA) [89] for resource allocation. Compared with the DDQL-based scheme with pruning, the advantage of using DDQL with pruning method can be shown.

4.6.2 Simulation Results

Fig. 4.5 shows the convergence performance comparison of different scenarios in the proposed schemes using dueling deep Q-learning learning. The value of loss function $L(\omega)$ reflects the degree of Q-value fitting. At the beginning of training, deep neural networks (10 hidden layers and CNN) do not have enough knowledge of the uncertain environment, and with the increasing experiences from the experience replay pool, the training loss is higher. When the agent learns enough experiences, the

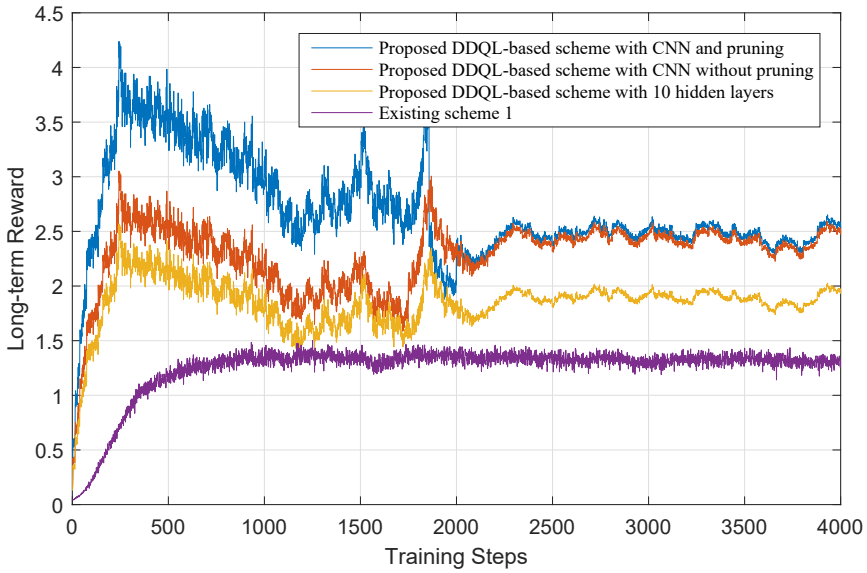

Figure 4.6: Training curves tracking the long-term reward of BMEC-FV under different schemes.

training loss decreases at the same time. Such increasing and decreasing performance of the training loss illustrates the effectiveness of deep neural networks. Moreover, from Fig. 4.5, we can see that our proposed schemes with pruning method are better than the schemes without pruning. This is because the pruning method eliminates unnecessary values in the weight tensor of the CNN network, reduces the number of connections between the neural network layers, and reduces the parameters involved in the calculation, thereby reducing the number of operations.

Fig. 4.6 shows the system performance comparison under different schemes. In our simulation, each point is the average long-term reward per episode, and the agent uses Adamoptimizer [72]. At the beginning of training, the proposed schemes have some trials and errors. After increasing the training episodes, the system long-term reward becomes stable, which means that the agent has learned the optimal policies to maximize the long-term rewards. As shown in this figure, we can also conclude that the system reward of our proposed schemes is better than the existing

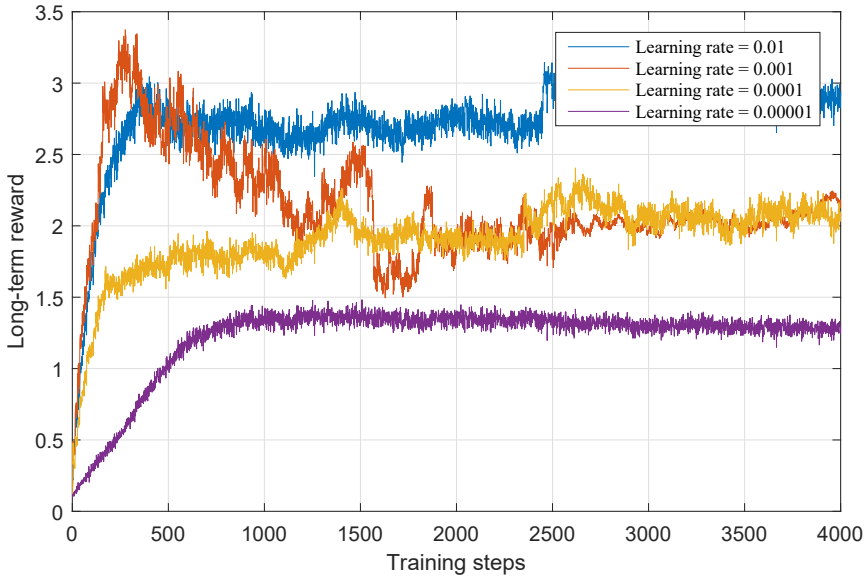

Figure 4.7: Training curves tracking the long-term reward of BMEC-FV under different learning rates.

scheme 1. There are two reasons for the higher system performance: 1) our proposed schemes use the RBFT consensus mechanism. Comparing with PBFT, RBFT has added important transaction verification links, ensuring a consensus on the order of transaction execution while ensuring consensus on the results of block verification, thus improving the consensus efficiency and system throughput; 2) since the existing scheme does not consider the reliable feature of each vehicle, the malicious vehicles will degrade the system throughput. Meanwhile, the proposed scheme uses dynamic adaptive subchannel allocation, which can effectively reduce the communication delay. Moreover, the proposed scheme can adaptively adjust the block size and the number of generated blocks, thereby improving the performance of the entire system.

Fig. 4.7 shows the reward convergence performance of the proposed DDQL-based scheme using the pruning method with different learning rates. The learning rate determines the updating speed of the weighted factor in deep Q-network. If the learning rate is too big, it will cause the result to exceed the optimal value. Otherwise,

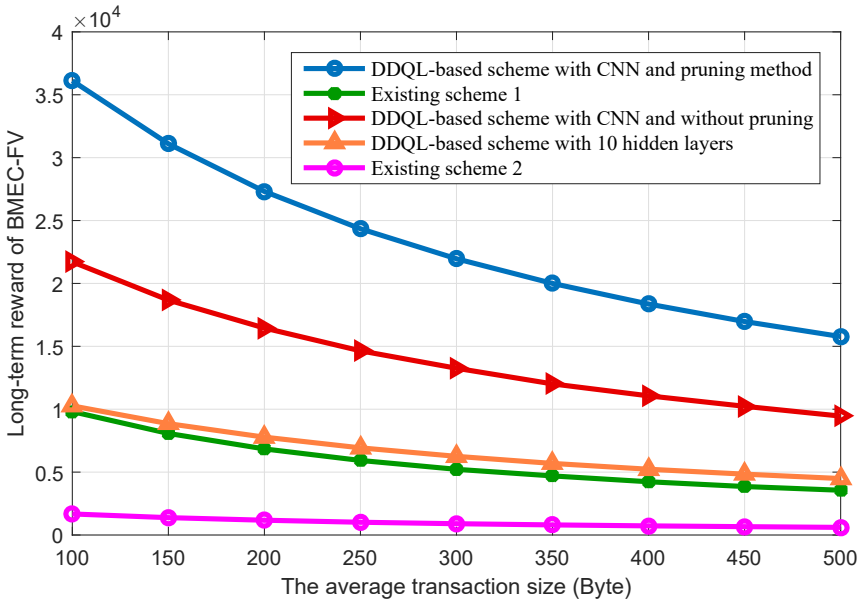

Figure 4.8: The long-term reward comparison of BMEC-FV versus the average transaction size.

it will deeply interfere with the Q-network convergence. From this figure, when learning rates are 0.01 and 0.001, the training curves are highly scaled, which leads to miss the global optimum. Compared with the two curves of yellow and purple, although the yellow curve has faster convergence speed, it does not very stable after convergence comparing with the purple curve. Therefore, we choose the learning rate as $1e^{-4}$ because of its acceptable convergence speed and better learning stability.

Figure. 4.8 depicts the relationship between BMEC-FV system performance and average transaction size. This figure can be used to show the performance of the proposed method with different transaction sizes that correspond to the offloading tasks in BMEC-FV systems. From this figure, we can see that as the average transaction size increases, the system performance drops significantly. There are two reasons for this: 1) When the sizes of transactions increases, a block can contain fewer transactions. In other words, the number of blocks in the system increases significantly. When the system processes more blocks, the performance of the system will decrease.

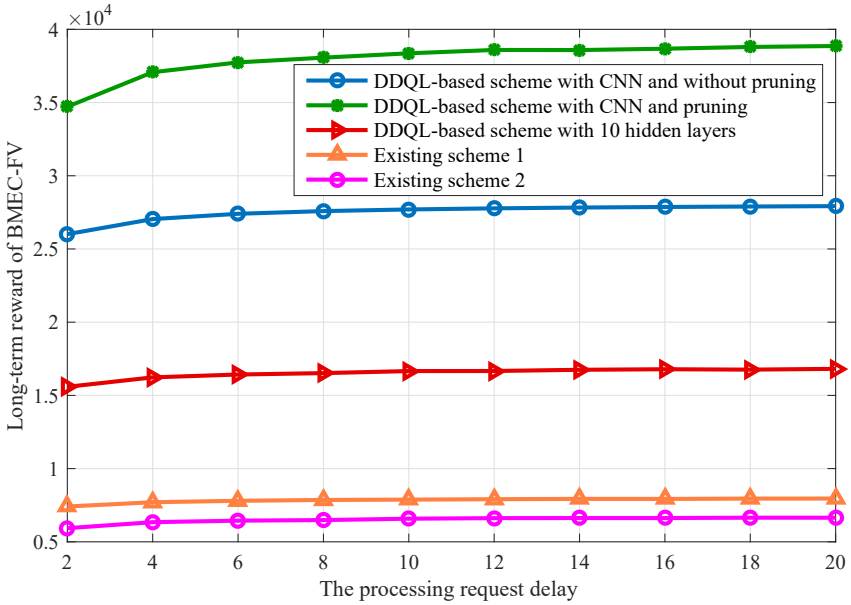

Figure 4.9: The long-term reward comparison of BMEC-FV versus the processing request delay.

2) The transmission delay from VANET nodes to RSUs increases, which reduces the performance of the MEC system. From the long-term benefits of BMEV-FV system, our proposed scheme achieves the highest long-term reward. Existing schemes have the worst performance due to the inability to ensure the security of the communication link, the uncertainty of the transaction size, and the number of fixed blocks.

Fig. 4.9 shows the relationship between the performance of the BMEC-FV and the processing request delay of the blockchain. From this figure, we can see that the long-term reward of the system increases with the increase of the processing request delay and eventually remains stable. We specify delay constraints for processing request delays in the constraints of the reward function, in order to add fewer penalties to the rewards, which naturally leads to higher performance of the BMEC-FV system. However, when the threshold for processing the request delay is large enough, the impact on the reward is small. Meanwhile, we can see that our proposed pruning-based scheme is better than existing solutions in performance. The reason is that it

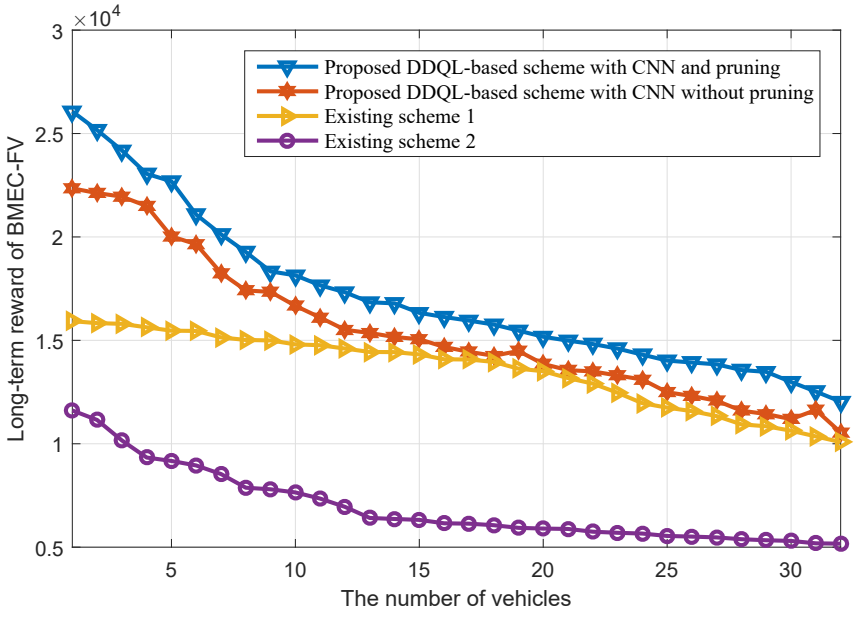

Figure 4.10: The long-term reward comparison of BMEC-FV versus the processing request delay.

not only ensures the secure communication and channel allocation between the car networking node and the RSU but also dynamically adjusts the block size and the number of generated blocks, thereby adjusting the block interval to strictly define the threshold for processing the request delay and avoiding the loss of the block. In addition, we found that existing schemes have the worst performance, which reveals the superiority of the DDQL-based method. The proposed DDQL-based scheme with CNN and pruning also maintains all the best performance.

Fig. 4.10 plots the rewards to show the scalability of the BMEC-FV system and the ability of the system to handle dynamic changes in the number of VANET nodes. In our simulation, we first train the BMEC-FV framework with 32 VANET nodes and 4 RSUs in an off-line mode. This off-line training framework is then used to process the VANET environment online, where the VANET node U changes from 2 to 32 and the number of RSUs C is fixed at 4. In other words, the curve obtained by this figure is obtained online. From this figure, we can conclude that even with a large number

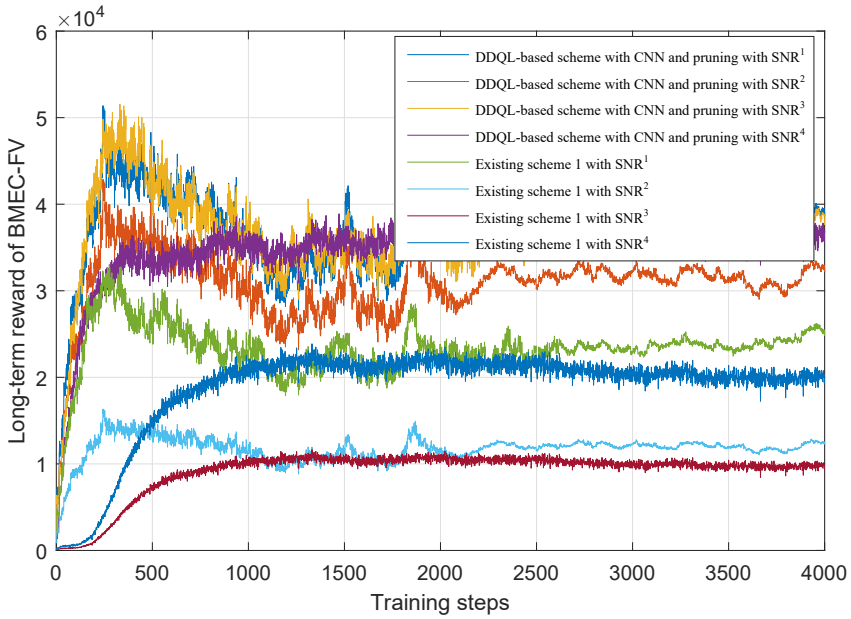

Figure 4.11: The long-term reward comparison of BMEC-FV versus the different SNR.

of VANET nodes, the proposed methods can get the highest long-term reward, while RSA can get the lowest return. The performance of the existing 1 is better than the former but still lower than our proposed methods. This figure shows the superiority of the proposed schemes and also shows the scalability of the proposed framework. Meanwhile, as the number of VANET nodes increases, the reward from these schemes is reduced. The reasons are: 1) the average resource is reduced, resulting in lower latency, and 2) more blocks may be ignored, resulting in smaller throughput for the entire system. Therefore, long-term rewards are reduced. Finally, this figure shows that the off-line training method can actually be used online, so it can be applied to actual VANET scenarios.

In Fig. 4.11, the performance of the proposed scheme is verified by comparing the proposed scheme with the existing algorithm 1 in three cases where the SNR state setting is different. In the simulation, once the SNR changes dynamically, we need to train the proposed BMEC-FV architecture in order to verify the adaptability of the

proposed scheme to SNR dynamics. First, the scene with SNR^4 gets the best return. This is because in this case, the system can obtain a better SNR state, which leads to greater throughput and lower latency, resulting in a greater long-term reward. Secondly, it can be seen that when the number of SNR states increases, the proposed scheme has different convergence conditions. The reason is that because the state space dimension rises, the agent needs to spend a longer learning process to get the optimal policy. Third, the convergence speed of these algorithms is different. This is because the state, action, and number of steps in an episode are much larger than the dimensions of the state set, which can be ignored by the law of large numbers. Fourth, due to the superiority of DRL, the proposed scheme achieves the best performance, which also indicates the ability of the proposed scheme to adapt to different network dynamics. It should be pointed out that since the existing scheme 2 does not apply the DDQL method, it does not compare its performance here.

Chapter 5

Conclusions and Future Works

5.1 Conclusions

In this dissertation, we investigated performance improvements in software-defined and MEC based VANETs with advanced optimization techniques.

- In Chapter 2, we proposed a deep Q-learning approach used in connected vehicle communication in VANETs. We designed a novel framework named SD-TDQL, which is based on a novel deep Q-learning mechanism. The software-defined controller in the control layer acts as a learning agent to interact with the VANET environment. In the trust derivation phase, a trust model was proposed for each vehicle to determine the ETX delay of the communication link for the long-term reward (Q-value). Meanwhile, a deep Q-learning algorithm was introduced to obtain the optimal communication link quality policy. The simulation results were presented to illustrate that the deep Q-learning is a feasible and effective approach to solve the joint optimization problem in software-defined VANETs.

- In Chapter 3, we presented a novel blockchain-based dueling deep reinforcement learning approach used in a VANET environment. We designed a

blockchain-based hierarchical distributed software-defined VANET framework (block-SDV). In the dueling deep reinforcement learning model of block-SDV, we used dueling deep Q-learning algorithm to determine the best policy for maximizing the system throughput. In order to improve the throughput of block-SDV, we jointly considered the trust feature of each vehicle in the device layer, the trust feature of each consensus node in the blockchain system, the number of consensus nodes, and computing capabilities of the edge computing servers. Due to the complexity of the considered system, we proposed to use a novel dueling deep reinforcement learning with a prioritized experience replay method to optimize this problem. The simulation results show the effectiveness of our proposed method.

- In Chapter 4, we presented a novel blockchain-based dueling deep reinforcement learning approach with the pruning method used in a VANET environment. We designed a novel framework of blockchain-based mobile edge computing for a future VANET ecosystem (BMEC-FV). In the dueling deep reinforcement learning model of BMEC-FV, we used dueling deep Q-learning algorithm to determine the best policy for maximizing the system performance. In order to improve the performance of BMEC-FV, we jointly considered the reliable feature of each vehicle in the VANET environment, the subchannel allocation for each RSU and each vehicle, the block size, and the appropriate number of block number of each block producer. Due to the complexity of the considered system, we proposed to use a novel dueling deep reinforcement learning with the pruning method to optimize this problem. The simulation results show the effectiveness of our proposed method. In the future, we hope to use a multi-agent deep reinforcement learning approach to enhance the performance.

In summary, first, we applied the DQL method to the centralized SDV, aiming to use the DQL method to obtain the best ETX link strategy. However, due to the shortcomings of the centralized SDV (the specific defects have been clarified in the first sheet), we further adapted the distributed SDV method to make up for the shortcomings of the centralized SDV. At the same time, we introduced blockchain technology to ensure the interaction of distributed SDV controllers. Finally, we added MEC on the basis of distributed SDV and blockchain to complete more complex tasks and give a new and more reliable trust calculation method through computing offloading to improve system performance.

5.2 Future Works

A number of interesting research problems arise during the course of the investigations reported in this dissertation.

- Although the application of DRL to the connected vehicle environment is a relatively innovative idea, due to some problems in DRL itself, we can still do a lot of work in future research. Since DRL is still very unstable, this has also become a bottleneck restricting its application in the connected car environment. For the same DRL training task, even if all the same hyperparameters and algorithms are used, the performance will be different when solving the same task. Reinforcement learning is very sensitive to changes that occur during initialization and training, because all data is collected in real-time, and the only indicator we can monitor is the reward, which is a scalar. A strategy that performs well at random will be activated faster than a strategy that does not apply randomly, and if a good strategy fails to give an excellent case in time, reinforcement learning will rashly conclude that all its performance is

poor. Therefore, in future research, we will focus on making breakthroughs in the stability of the DRL algorithm.

- In our research, we have introduced blockchain technology into the current connected car system. However, due to some problems in the blockchain technology itself, there are still many directions worth studying in the future. The first is the storage of blockchain data. There is cloud storage in the internet era. However, where does the massive data in the blockchain era exist? At present, the IPFS protocol and FileCoin project that solve the blockchain storage problem have attracted much attention. The second is the transaction per second (TPS) problem of the blockchain. We all know that the traditional centralized system has a very high transaction processing speed, which is tens of thousands of transactions per second, but the blockchain is different. Single-digit Bitcoin and double-digit Ethereum. Although in the process of demonstration, consensus security and transaction processing speed in everyone's blockchain system are contrary, it is necessary to sacrifice one party. The DPOS mechanism seems to solve the problem of transaction processing speed, but in practice, whether it can run safely for a long time and receive any from the public, it will take a long time to verify. Consensus resolution speed of transaction processing is also a direction, but the market still needs to verify the feasibility. Therefore, in future research, storing and sharing the data of the Internet of Vehicles in the blockchain system will still need to improve TPS and better blockchain data storage technology.

List of References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, Dec. 2013.

[2] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," *arXiv preprint arXiv:1511.06581*, Apr. 2015.

[3] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, and H. Wang, "Blockchain Challenges and Opportunities: A Survey," *International Journal of Web and Grid Services*, vol. 14, no. 4, pp. 352–375, Jan. 2018.

[4] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, "Integrated Blockchain and Edge Computing Systems: A Survey, Some Research Issues and Challenges," *IEEE Comm. Survey and Tutorials*, vol. 21, no. 2, pp. 1508–1532, Jan. 2019.

[5] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Comm. Survey and Tutorials*, vol. 19, no. 3, pp. 1628–1656, Mar. 2017.

[6] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Comm. Survey and Tutorials*, vol. 19, no. 4, pp. 2322–2358, Aug. 2017.

[7] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile Edge Computing: A Survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Sept. 2017.

[8] F. R. Yu, "Connected Vehicles for Intelligent Transportation Systems [Guest Editorial]," *IEEE Trans. Veh. Tech.*, vol. 65, no. 6, pp. 3843–3844, June 2016.

[9] K. Abboud and W. Zhuang, "Impact of Node Mobility on Single-Hop Cluster Overlap in Vehicular Ad Hoc Networks," in *Proc. ACM MSWiM'14*, (Montreal, Canada), Sept. 2014.

[10] A. Grzybek, G. Danoy, M. Seredynski, and P. Bouvry, "Evaluation of Dynamic Communities in Large-Scale Vehicular Networks," in *Proc. ACM DIVANet'13*, (Barcelona, Spain), Nov. 2013.

[11] P. Vijayakumar, M. Azees, A. Kannan, and L. J. Deborah, "Dual Authentication and Key Management Mechniques for Secure Data Transmission in Vehicular Ad Hoc Networks," *IEEE Trans. Intell. Transp. Sys.*, vol. 17, no. 4, pp. 1015–1028, Apr. 2016.

[12] H. Shafiq, R. A. Rehman, and B.-S. Kim, "Services and Security Threats in SDN Based VANETs: A Survey," *Wireless Comm. and Mobile Computing*, vol. 124, pp. 1–14, Apr. 2018.

[13] A. Alnasser, H. Sun, and J. Jiang, "Cyber Security Challenges and Solutions for V2X Communications: A Survey," *Elsevier Computer Networks*, vol. 151, pp. 52–67, Mar. 2019.

[14] Z. Lu, G. Qu, and Z. Liu, "A Survey on Recent Advances in Vehicular Network Security, Trust, and Privacy," *IEEE Trans. Intell. Transp. Sys.*, no. 99, pp. 1–17, Apr. 2018.

[15] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges," *IEEE Comm. Survey and Tutorials*, vol. 18, no. 1, pp. 602–622, Oct. 2015.

[16] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Comm. Survey and Tutorials*, vol. 17, no. 1, pp. 27–51, First Quarter 2015.

[17] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[18] F. Bannour, S. Souihi, and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy, and Challenges," *IEEE Comm. Survey and Tutorials*, vol. 20, no. 1, pp. 333–354, Dec. 2017.

[19] A. Tootoonchian and Y. Ganjali, "Hyperflow: A Distributed Control Plane for Openflow," in *Proc. NSDI'10*, (San Jose, CA), Apr. 2010.

[20] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, *et al.*, "Onix: A Distributed Control Platform for Large-Scale Production Networks.," in *Proc. OSDI'10*, (Vancouver, Canada), Oct. 2010.

[21] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications," in *Proc. ACM HotSDN'12*, (Helsinki, Finland), Aug. 2012.

[22] F. R. Yu, *Blockchain Technology and Applications - From Theory to Practice*. Kindle Direct Publishing, 2019. https://www.amazon.com/dp/1729142591.

[23] P. K. Sharma, M.-Y. Chen, and J. H. Park, "A Software Defined Fog Node Based Distributed Blockchain Cloud Architecture for IoT," *IEEE Access*, vol. 6, pp. 115–124, Sept. 2018.

[24] P. K. Sharma, S. Singh, Y.-S. Jeong, and J. H. Park, "Distblocknet: A Distributed Blockchains-Based Secure SDN Architecture for IoT Networks," *IEEE Comm. Mag.*, vol. 55, no. 9, pp. 78–85, Sept. 2017.

[25] Z. Lu, Q. Wang, G. Qu, and Z. Liu, "Bars: A Blockchain-Based Anonymous Reputation System for Trust Management in VANETs," in *Proc. IEEE TrustCom/BigDataSE*, (New York, USA), Aug. 2018.

[26] P. K. Sharma, S. Y. Moon, and J. H. Park, "Block-VN: A Distributed Blockchain Based Vehicular Network Architecture in Smart City," *Journal of Information Processing Systems*, vol. 13, no. 1, pp. 184–195, Feb. 2017.

[27] M. Singh and S. Kim, "Blockchain Based Intelligent Vehicle Data Sharing Framework," *arXiv preprint arXiv:1708.09721*, June 2017.

[28] B. Rashid and M. H. Rehmani, "Applications of Wireless Sensor Networks for Urban Areas: A Survey," *Journal of Network and Computer Applications*, vol. 60, pp. 192–219, Jan. 2016.

[29] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance Guaranteed Computation Offloading for Mobile-Edge Cloud Computing," *IEEE Wireless Comm. Letters*, vol. 6, no. 6, pp. 774–777, Dec. 2017.

[30] Y. He, F. R. Yu, N. Zhao, V. C. Leung, and H. Yin, "Software-Defined Networks with Mobile Edge Computing and Caching for Smart Cities: A Big Data Deep

Reinforcement Learning Approach," *IEEE Comm. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[31] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Computation Offloading and Resource Allocation in Wireless Cellular Networks with Mobile Edge Computing," *IEEE Trans. Wireless Comm.*, vol. 16, no. 8, pp. 4924–4938, May. 2017.

[32] Z. Yang, K. Yang, L. Lei, K. Zheng, and V. C. Leung, "Blockchain-Based Decentralized Trust Management in Vehicular Networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1495–1505, May. 2018.

[33] T. Salman, M. Zolanvari, A. Erbad, R. Jain, and M. Samaka, "Security Services Using Blockchains: A State of the Art Survey," *IEEE Comm. Survey and Tutorials*, vol. 21, no. 1, pp. 858–880, Aug. 2018.

[34] K. Rabieh, M. M. Mahmoud, and M. Younis, "Privacy-Preserving Route Reporting Schemes for Traffic Management Systems," *IEEE Trans. Veh. Tech.*, vol. 66, no. 3, pp. 2703–2713, Mar. 2017.

[35] Y. Xiao and Y. Liu, "BayesTrust and VehicleRank: Constructing an Implicit Web of Trust in VANET," *IEEE Trans. Veh. Tech.*, vol. 68, no. 3, pp. 2850 – 2864, Mar. 2019.

[36] H. Hu, R. Lu, Z. Zhang, and J. Shao, "REPLACE: A Reliable Trust-Based Platoon Service Recommendation Scheme in VANET," *IEEE Trans. Veh. Tech.*, vol. 66, no. 2, pp. 1786–1797, Feb. 2017.

[37] S. Tan, X. Li, and Q. Dong, "A Trust Management System for Securing Data Plane of Ad-Hoc Networks," *IEEE Trans. Veh. Tech.*, vol. 65, no. 9, pp. 7579–7592, Sept. 2016.

[38] Y. He, F. R. Yu, Z. Wei, and V. Leung, "Trust Management for Secure Cognitive Radio Vehicular Ad Hoc Networks," *Elsevier Ad Hoc Networks*, vol. 86, pp. 154–165, Apr. 2019.

[39] S. Guleng, C. Wu, X. Chen, X. Wang, T. Yoshinaga, and Y. Ji, "Decentralized Trust Evaluation in Vehicular Internet of Things," *IEEE Access*, vol. 7, pp. 15980–15988, Jan. 2019.

[40] G. S. Aujla, A. Jindal, and N. Kumar, "EVaaS: Electric Vehicle-as-a-Service for Energy Trading in SDN-Enabled Smart Transportation System," *Elsevier Computer Networks*, vol. 143, pp. 247–262, Oct. 2018.

[41] T. T. Nguyen and G. Armitage, "A Survey of Techniques for Internet Traffic Classification Using Machine Learning," *IEEE Comm. Survey and Tutorials*, vol. 10, no. 4, pp. 56–76, Fourth Quarter, 2008.

[42] A. Gosavi, "Reinforcement Learning: A Tutorial Survey and Recent Advances," *INFORMS Journal on Computing*, vol. 21, no. 2, pp. 178–192, Dec. 2009.

[43] S. Chettibi and S. Chikhi, "Dynamic Fuzzy Logic and Reinforcement Learning for Adaptive Energy Efficient Routing in Mobile Ad-Hoc Networks," *Applied Soft Computing*, vol. 38, pp. 321–328, Jan. 2016.

[44] A. A. Bhorkar, M. Naghshvar, T. Javidi, and B. D. Rao, "Adaptive Opportunistic Routing for Wireless Ad Hoc Networks," *IEEE/ACM Transactions On Networking*, vol. 20, no. 1, pp. 243–256, July. 2012.

[45] G. M. Borkar and A. Mahajan, "A Secure and Trust Based On-Demand Multipath Routing Scheme for Self-Organized Mobile Ad-Hoc Networks," *Wireless Networks*, vol. 23, no. 8, pp. 2455–2472, May. 2017.

[46] L. Xiao, X. Lu, D. Xu, Y. Tang, L. Wang, and W. Zhuang, "UAV Relay in VANETs Against Smart Jamming with Reinforcement Learning," *IEEE Trans. Veh. Tech.*, vol. 67, no. 5, pp. 4087–4097, May. 2018.

[47] A. Jindal, G. S. Aujla, N. Kumar, R. Chaudhary, M. S. Obaidat, and I. You, "SeDaTiVe: SDN-Enabled Deep Learning Architecture for Network Traffic Control in Vehicular Cyber-Physical Systems," *IEEE Network*, vol. 32, no. 6, pp. 66–73, Nov. 2018.

[48] S. S. Mousavi, M. Schukat, and E. Howley, "Traffic Light Control Using Deep Policy-Gradient and Value-Function-Based Reinforcement Learning," *IET Intell. Transp. Sys.*, vol. 11, no. 7, pp. 417–423, Sept. 2017.

[49] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human Level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[50] T. Y. He, N. Zhao, and H. Yin, "Integrated Networking, Caching and Computing for Connected Vehicles: A Deep Reinforcement Learning Approach," *IEEE Trans. Veh. Tech.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

[51] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al.*, "ONOS: Towards an Open, Distributed SDN OS," in *Proc. Conf. Hot Topics in Software Defined Net.*, pp. 1–6, Aug. 2014.

[52] C. Qiu, C. Zhao, F. Xu, and T. Yang, "Sleeping Mode of Multi-Controller in Green Software-Defined Networking," *EURASIP Journal on Wireless Commu. and Net.*, vol. 2016, no. 1, p. 282, Dec. 2016.

[53] H. Yao, C. Qiu, C. Zhao, and L. Shi, "A Multicontroller Load Balancing Approach in Software-Defined Wireless Networks," *International Journal of Distributed Sensor Net.*, vol. 11, no. 10, p. 454159, Oct. 2015.

[54] A. Lei, H. Cruickshank, Y. Cao, P. Asuquo, C. P. A. Ogah, and Z. Sun, "Blockchain-Based Dynamic Key Management for Heterogeneous Intelligent Transportation Systems," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1832–1843, Aug. 2017.

[55] J. Liu, J. Wan, B. Zeng, Q. Wang, H. Song, and M. Qiu, "A Scalable and Quick-Response Software Defined Vehicular Network Assisted by Mobile Edge Computing," *IEEE Comm. Mag.*, vol. 55, no. 7, pp. 94–100, Jul. 2017.

[56] C.-M. Huang, M.-S. Chiang, D.-T. Dao, W.-L. Su, S. Xu, and H. Zhou, "V2V Data Offloading for Cellular Network Based on the Software Defined Network (SDN) Inside Mobile Edge Computing (MEC) Architecture," *IEEE Access*, vol. 6, pp. 17741–17755, Mar. 2018.

[57] G. Luo, Q. Yuan, H. Zhou, N. Cheng, Z. Liu, F. Yang, and X. S. Shen, "Cooperative Vehicular Content Distribution in Edge Computing Assisted 5G-VANET," *China Communications*, vol. 15, no. 7, pp. 1–17, Jul. 2018.

[58] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. Zhang, "Mobile-Edge Computing for Vehicular Networks: A Promising Network Paradigm with Predictive Off-Loading," *IEEE Veh. Tech. Mag.*, vol. 12, no. 2, pp. 36–44, Apr. 2017.

[59] C. A. Kerrache, A. Lakas, N. Lagraa, and E. Barka, "UAV-Assisted Technique for the Detection of Malicious and Selfish Nodes in VANETs," *Vehicular Communications*, vol. 11, pp. 1–11, Jan. 2018.

[60] M. Nazari, A. Oroojlooy, L. V. Snyder, and M. Takáč, "Reinforcement Learning for Solving the Vehicle Routing Problem," *arXiv preprint arXiv:1802.04240*, May. 2018.

[61] X. Chen, J. Guo, Z. Zhu, R. Proietti, A. Castro, and S. Yoo, "Deep-RMSA: A Deep-Reinforcement-Learning Routing, Modulation and Spectrum Assignment Agent for Elastic Optical Networks," in *Proc. IEEE OFC*, (San Diego, USA), Mar. 2018.

[62] S. Chinchali, P. Hu, T. Chu, M. Sharma, M. Bansal, R. Misra, M. Pavone, and S. Katti, "Cellular Network Traffic Scheduling with Deep Reinforcement Learning," in *Proc. Int'l Conf. AAAI'18*, (New Orleans, USA).

[63] X. Li, Z. Jia, P. Zhang, R. Zhang, and H. Wang, "Trust-Based On-Demand Multipath Routing in Mobile Ad Hoc Networks," *IET Information Security*, vol. 4, no. 4, pp. 212–232, Dec. 2010.

[64] S. Marti, T. J. Giuli, K. Lai, and M. Baker, "Mitigating Routing Misbehavior in Mobile Ad Hoc Networks," in *Proc. MobiCom'00*, (New York, United States), Aug. 2000.

[65] A. A. Pirzada, A. Datta, and C. McDonald, "Propagating Trust in Ad-Hoc Networks for Reliable Routing," in *Proc. International Workshop on Wireless Ad-Hoc Networks*, (Oulu, Finland), June. 2004.

[66] Y. Fu, J. Bi, Z. Chen, K. Gao, B. Zhang, G. Chen, and J. Wu, "A Hybrid Hierarchical Control Plane for Flow-Based Large-Scale Software-Defined Networks," *IEEE Trans. Network and Service Management*, vol. 12, no. 2, pp. 117–131, June. 2015.

[67] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[68] D. S. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A High-Throughput Path Metric for Multi-Hop Wireless Routing," in *Proc. ACM MobiCom'03*, (San Diego, USA), Sept. 2003.

[69] O. Alzamzami and I. Mahgoub, "Fuzzy Logic-Based Geographic Routing for Urban Vehicular Networks Using Link Quality and Achievable Throughput Estimations," *IEEE Trans. Intell. Transp. Sys.*, vol. 20, no. 8, pp. 1–12, Sept. 2018.

[70] C. J. Watkins and P. Dayan, "Q-Learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, May. 1992.

[71] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," *arXiv preprint arXiv:1603.04467*, Mar. 2016.

[72] T. M. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman, "Project Adam: Building an Efficient and Scalable Deep Learning Training System.," in *Proc. OSDI'14*, (Broomfield, CO), Oct. 2014.

[73] C. Qiu, F. R. Yu, F. Xu, H. Yao, and C. Zhao, "Blockchain-Based Distributed Software-defined Vehicular Networks via Deep Q-Learning," in *Proc. ACM DIVANet'18*, (Montreal, Canada), Nov. 2018.

[74] F. R. Yu, J. Liu, Y. He, P. Si, and Y. Zhang, "Virtualization for Distributed Ledger Technology (vDLT)," *IEEE Access*, vol. 6, pp. 25019–25028, Apr. 2018.

[75] P.-L. Aublin, S. B. Mokhtar, and V. Quéma, "RBFT: Redundant Byzantine Fault Tolerance," in *Proc. IEEE Int'l Conf. Distributed Computing Systems (ICDCS)'13*, (Philadelphia, USA), July 2013.

[76] M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance and Proactive Recovery," *ACM/Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[77] G. Rong, "Hypechain," *GitHub repository*, 2018.

[78] C. Weifang, L. Xiangke, S. Changxiang, L. Shanshan, and P. Shaoliang, "A Trust-Based Routing Framework in Energy-Constrained Wireless Sensor Networks," in *Proc. Int'l Conf. Wireless Algorithms, Systems, and Applications (WASA)'06*, (Xian, China), Aug. 2006.

[79] H. Wu, X. Wang, Q. Zhang, and X. Shen, "IEEE 802.11e Enhanced Distributed Channel Access (EDCA) Throughput Analysis," in *Proc. IEEE ICC'06*, (Istanbul, Turkey), June. 2006.

[80] A. Clement, E. L. Wong, L. Alvisi, M. Dahlin, and M. Marchetti, "Making Byzantine Fault Tolerant Systems Tolerate Byzantine Faults.," in *Proc. ACM NSDI'09*, (Boston, Massachusetts), Apr. 2009.

[81] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška, "The Importance of Experience Replay Database Composition in Deep Reinforcement Learning," in *Proc. NIPS'15 Workshops*, (Montreal Canada), Dec. 2015.

[82] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," *arXiv preprint arXiv:1511.05952*, Feb. 2015.

[83] B. Cui and S. J. Yang, "NRE: Suppress Selective Forwarding Attacks in Wireless Sensor Networks," in *Proc. IEEE Communications and Network Security*, (San Francisco, USA), Oct. 2014.

[84] S. M. Das, H. Pucha, K. Papagiannaki, and Y. C. Hu, "Studying Wireless Routing Link Metric Dynamics," in *Proc. ACM IMC'07*, (San Diego, USA), Oct. 2007.

[85] G. Rong, "Hypechain." https://github.com/hyperchain/hyperchain.git, 2018.

[86] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," *IEEE/ACM Trans. Networking*, vol. 24, no. 5, pp. 2795–2808, Oct. 2015.

[87] M. Liu, Y. Teng, F. R. Yu, V. C. Leung, and M. Song, "Deep Reinforcement Learning Based Performance Optimization in Blockchain-Enabled Internet of Vehicle," in *Proc. IEEE ICC'19*, (Shanghai, China), Jul. 2019.

[88] U. Challita, W. Saad, and C. Bettstetter, "Interference Management for Cellular-Connected UAVs: A Deep Reinforcement Learning Approach," *IEEE Trans. Wireless Comm.*, vol. 18, no. 4, pp. 2125–2140, Apr. 2019.

[89] J. Zhu, Y. Song, D. Jiang, and H. Song, "A New Deep-Q-Learning-Based Transmission Scheduling Mechanism for the Cognitive Internet of Things," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 2375–2385, Oct. 2017.