Received May 24, 2019, accepted June 11, 2019, date of publication June 14, 2019, date of current version July 15, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2923021

Autonomous Cache Resource Slicing and Content Placement at Virtualized Mobile Edge Network

GUOLIN SUN[1], HISHAM AL-WARD[1], GORDON OWUSU BOATENG[1], AND GUISONG LIU[1,2]
[1]School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China
[2]School of Computer Science, Zhongshan Institute, University of Electronic Science and Technology of China, Zhongshan 528400, China

Corresponding author: Guisong Liu (lgs@uestc.edu.cn)

This work was supported in part by the National Natural Science Research Foundation of China under Grant 61771098, in part by the Fundamental Research Funds for the Central Universities under Grant ZYGX2018J068, in part by the fund from the Department of Science and Technology of Sichuan Province under Grant 2017GFW0128, Grant 8ZDYF2268, Grant 2018JY0578, and Grant 2017JY0007, and in part by the ZTE Innovation Research Fund for Universities Program 2016.

ABSTRACT Nowadays, the unprecedented growth of mobile traffic is only rivaled by the demands for a better quality of service and faster data delivery. Caching the content closer to the end users is one solution to cope with these demands, but when service providers share a common physical infrastructure, managing the limited cache resources becomes more challenging. With this increased complexity, the deep reinforcement learning (DRL) presents itself as an optimal solution. In this work, we propose a dynamic DRL-based management framework for virtual cache slicing and customized content placement. The virtualization problem is formulated as a Markov decision process, and it utilizes a deep Q-network to autonomously make network-wide slicing decisions and optimize the cache resource allocation. These global decisions are then mapped to the small cell stations with a dynamic resource provisioning algorithm. Lastly, a customized content placement algorithm is used to find the optimal content placement policy for each service provider. The content placement problem is first formulated as a convex optimization problem and is then solved with a distributed alternating direction method of multipliers (ADMM). Finally, the simulation results are presented to show the effectiveness of the proposed solution.

INDEX TERMS Content placement, deep reinforcement learning, deep Q-networks, network virtualization, resource provisioning, resource slicing.

I. INTRODUCTION

The world today is witnessing a skyrocketing growth of mobile traffic. This unprecedented development is accompanied by an ever-increasing demand from the end user for an improved quality of experience (QoE) and faster data delivery. The Cisco visual networking index in [1] reports that mobile data traffic has grown 18-fold between 2011 and 2016 and predicts the global mobile data traffic to increase sevenfold and reach 49.0 exabytes per month by 2021. This dramatic rise in mobile traffic inevitably leads to congestion of backhaul networks, which in turn results in higher costs of operation and maintenance, lower quality of service (QoS), and slower data delivery. The gaps between the expected and the feasible performances are too big for the fourth generation (4G) technologies and the traditional internet pconrotocol (IP) networks to fill on their own, and thus new solutions need to be sought.

One major challenge facing traditional IP networks is the congestion of backhaul links due to the redundant transmissions traveling for the original content provider. This problem can be effectively solved by caching the content. When content replicas are distributed the network, the amount of data traveling through the core network is reduced, and the visiting time is proportionally decreased as well. And while caching is sure to improve the QoS, placing content replicas around the network is wasteful from a resource utilization perspective. As such, the content placement problem should be approached carefully as to balance between QoS satisfaction and resource utilization. In this work, we consider caching at the edge of the mobile network, within the radio access network (RAN) and in close proximity to mobile subscribers [2].

However, it is very difficult to implement a caching solution in traditional IP networks because each network provider

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

has to own its dedicated physical infrastructure (i.e. caching storage). These hefty expenses make it difficult for network providers to cache their content or scale their services [3]. A popular solution to this problem is found by adopting network function virtualization (NFV). By means of NFV, the physical infrastructure can be divided and abstracted into isolated substrate resources called virtual network slices [4]. In a virtualized network, an infrastructure provider (InP) owns the infrastructure physical equipment and leases them out to mobile virtual network providers (MVNOs). Similarly, if the InP owns a wireless network infrastructure, the scheme can be referred to as wireless network virtualization [5]. With network virtualization, differentiated services thrive since the MVNOs can serve diverse applications according to their own QoS requirements. In addition, the InPs can maximize their resource utilization by leasing their physical resources to multiple MVNOs. In this paper, our objective is to develop a framework to provide a balance between the QoS satisfaction of MVNOs and the resource utilization of InPs.

The resource slicing and content placement problems are very promising in the field of wireless mobile telecommunication, and they have received considerable attention in recent years. However, most current research efforts fall short in the following accounts. First, most of the current works that address both the cache slicing and content placement problems fail to support MVNO customization. Authors in [6], for example, formulate the cache resource slicing and in-network caching strategy as a cost-based optimization problem and solve it with a distributed ADMM, but they assume that all the MVNOs have one objective function, which is not quite realistic. Second, most reinforcement learning (RL)-based caching solutions focus only on the content placement problem. For example, authors in [7] develop a Q-learning algorithm to find the optimal caching policy with unknown transitional probabilities by considering both the global and local popularity demands, but they do not consider virtualization or cache slicing. Third, RL-based solutions that address both resource slicing and support MVNO customization choose a different resource than cache. Authors in [8], for example, develop a novel radio resource slicing framework for 5G networks. They find the optimal slicing strategy with an RL method, then they customize the radio resource for haptic communications using a low-complexity heuristic algorithm. Our motivation in this paper is to develop a cache management framework for virtualized wireless networks that optimizes both the slicing of the cache resource and the actual content placement for providers with differentiated services.

Considering differentiated services makes developing a model and handcrafting clever heuristics considerably more complicated. As such, we propose a DRL-based cache management framework. In a DRL system, an agent observes the environment and accordingly makes decisions. The agent learns from experience and continues to improve its decision-making policy by reinforcement- a reward that reflects how good its decisions are [9]. The deep Q-network (DQN) algorithm is chosen in our solution for the following reasons: 1) it is a model-free algorithm, which allows us to model such a complex problem that would otherwise be very challenging to build with handcrafted features, 2) DQN can be adaptive to varying conditions as long as it is allowed to continue to learn [10], and 3) in a real-time scenario, the changing demand of the users with the large state space will make finding an optimal solution incredibly difficult. In this context, a DQN algorithm has lower complexity than other traditional approaches such as dynamic programming. Our contributions in this paper can be summarized as follows:

- *We propose a novel cache resource management framework that handles both cache resource slicing and the content placement for MVNOs with differentiated services in virtualized wireless networks. This framework incorporates both DRL-based and distributed convex optimization methods [11].*
- *We first introduce a DRL-based virtual cache resource slicing strategy with two components: a DQN algorithm is responsible for making network-wide cache slicing decisions, and a dynamic resource provisioning algorithm to update the allocation and map the physical resources at the small cell stations to the MVNOs.*
- *Using the resource allocation from the DRL solution as an input, we use a distributed ADMM to find the optimal cache placement for the MVNOs. Unlike previous schemes where all the virtual slices are assumed to have the same objective function, our proposal allows for the customization of virtual slices: that is, each operator can optimize their cache placement independently and according to their own unique QoS metrics and objectives.*
- *Extensive simulations are conducted to verify the significance of the proposed work. Our results are compared to two algorithms: ADMM and popularity-aware cache provisioning (PACP).*

II. RELATED WORKS

Content placement and cache resource slicing have been investigated in the recent past. In [12], the authors aim at maximizing the effective capacity for the end-users by utilizing information-centric networks (ICN) to cache popular contents near mobile users rather than the remote, original providers. That is, when requests arrive at the network controller, it finds where the closest copy of the content is and creates a delivery path for the requesting user. The delay-bounded QoS for the multimedia transmissions is calculated with the effective capacity theory, while the delivery path is created on flexible network architecture. The work in [13] presents a novel information-centric HetNets framework aiming at enabling content caching and computing in a virtualized architecture. The virtual resource allocation strategy is formulated as a joint optimization problem where the gains of virtualization, caching, and computing are considered, and a distributed ADMM is used to reduce complexities and signaling. The authors in [14] solve the difficult problem jointly optimizing resource allocation and content caching by proposing a bisection-search-based algorithm that iteratively optimizes the resource allocation and content caching placement. They further develop a low-complexity heuristic algorithm which achieves moderate performance loss compared to the bisection-search based algorithm. Kyi *et al.* [15] apply the hierarchical business model for resource allocation and hybrid model for the physical resources sharing and formulate the virtual resource allocation and in-network caching strategy as an optimization problem, which maximizes the profit of MVNOs and improves the user's QoS.

Furthermore, Jia *et al.* [16] maximize the utilization of physical caching resource for network slicing via price mechanism, then formulate the caching resource allocation issue as an integer linear programming (ILP) model. Particularly, they also take the caching energy consumption into account and propose a chemical reaction optimization (CRO)-based algorithm to maximize caching resource utilization. The authors in [17] propose a joint user association, caching, spectrum, and power problem in an effort to maximize the utility for all network operators. The problem is formulated as a cost-driven virtual user association and resource allocation problem with stringent energy harvesting constraints. The problem is later reformulated as two sub-problems and solved with an ADMM-based distributed algorithm to reduce complexity and improve efficiency.

Some other works have been done in the virtualization of the wireless network while imploring learning techniques to help in the optimization problem. Authors in [7] introduce a novel RL framework for finding the optimal caching policy with unknown transition probability. This caching approach considers content popularity as well as cache-refreshing costs and is based on Q-learning implemented in an online fashion. A linear function approximation that allowed better scalability, faster convergence, and reduced complexity is also proposed. The work in [18] deals with an information-centric virtualized network for smart cities with a deep Q-learning approach. He *et al.* [19] formulate an optimization problem to maximize the network operator's utility while considering the trust-based social networks specifically with mobile edge computing (MEC), in-network caching, and device-to-device (D2D) communications using a DRL approach. Finally, an integrated framework that can enable dynamic orchestration of networking, caching and computing resources to improve the performance of next generation vehicular networks is studied in [20]. In this framework, the resource allocation strategy is formulated as a joint optimization problem and DRL is used to solve it.

Our work differs from [12]–[17] in that we implore DRL strategies to solve the problem of cache resource slicing and allow the system to support differentiated services. We believe that a DRL-based solution, in principle, is a more effective approach to solve the slicing problem, and we put this assumption to test in section V.D. The authors in [7] use Q-leaning in their solution, but in section IV.B of this paper, we explain the shortcomings of this method and how they are overcome with DQN. Lastly, our work differs from [18]–[20] in that it considers the content placement problem and also supports differentiated services.

III. SYSTEM MODEL

The system model is described in four entities: business model, virtualization model, network model, and utility models. In the business model, we explain the business entities governing our network and how they interact with one another as well as the objective from a business point of view. In the virtualization model, we provide a brief explanation of how the slicing decisions in our solution are executed and updated, and we also define a crucial element of our virtualization scheme: cache resource reservation. In the network model, we mathematically define every variable in our network and establish the user association and content caching principles. We also provide the calculations for a user's transmission rate in our network model, which are essential for developing the utility models. In the utility models sub-section, we formulate two utility functions: one for the QoS satisfaction, and one for the resource utilization. The QoS satisfaction calculation differs according to the type of the slice, whereas the resource utilization calculation is universal across the different slices.

A. BUSINESS MODEL

In this work, we adopt a two-tier business model consisting of InPs and MVNOs. The mobile edge network in our scenario consists of macro base stations (BSs), small cell stations (SBS's) and cache storage at the SBS's. An SDN controller separates the control and the data planes in our network. The SBS's and their caching capabilities form our data layer, while the macro base station will have control capabilities. As we can see in the system framework in Fig. 1, our management system uses two control units: A DRL-based controller for cache slicing, and an ADMM controller for customized content placement.

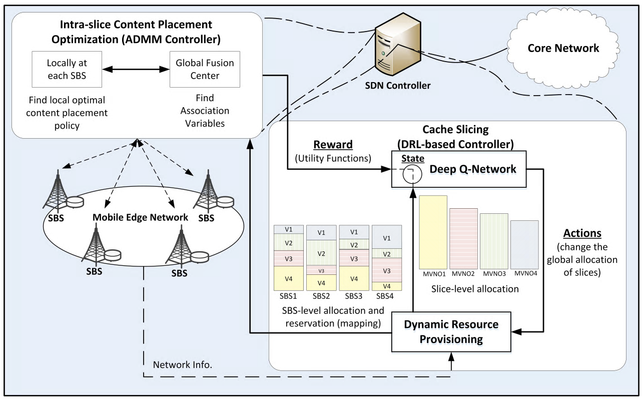
Intra-slice Content Placement Optimization (ADMM Controller)
Locally at each SBS
Global Fusion Center
Find local optimal content placement policy
Find Association Variables
SDN Controller
Core Network
Cache Slicing (DRL-based Controller)
Reward
(Utility Functions)
State
Deep Q-Network
SBS
Mobile Edge Network
SBS
SBS
SBS
V1
V2
V3
V4
SBS1
SBS2
SBS3
SBS4
SBS-level allocation and reservation (mapping)
MVNO1
MVNO2
MVNO3
MVNO4
Slice-level allocation
Actions
(change the global allocation of slices)
Dynamic Resource Provisioning
Network Info.

FIGURE 1. System framework.

In our business model, the network resources owned by the InPs are the macro base stations, small cell stations, and the caching storage. The MVNOs, on the other hand, own and manage unique types of content and/or have different QoS requirements. In this work, we consider four MVNOs with two objective functions. All the MVNOs are delay-constrained, but the first two provide guaranteed bit rate (GBR) services (i.e. live video streaming), while the last two provide non-GBR services (i.e. buffered video streaming) [21].

An MVNO leases network and cache infrastructure from the InP and makes a profit by charging the end users on a QoE-conditioned charge-by-use basis. The InP's income comes from the MVNO; however, as an MVNO only pays for the resources it uses, the InP needs to maximize the utilization of its resource pool. After all, idol resources mean a loss of potential profit. The ability to deliver the agreed upon service level agreement (SLA) is how we measure the success of this scheme.

B. VIRTUALIZATION MODEL

Network virtualization is the partitioning of the physical network into virtual networks. This means that the physical resource belonging to one InP can be shared by a number of MVNOs offering different services with different QoS requirements. The virtualization needs to a) *fulfill the QoS requirements of the MVNOs*, b) *ensure the maximum utilization of the InP physical resources*, and c) *provide complete isolation between MVNOs*. As such, our aim is to develop a dynamic slicing strategy that provides the optimal slicing, isolation, and mapping of physical cache resources in different small cells to MVNOs.

Resource Reservation: Resource reservation is essential in our work because it guarantees the isolation between MNVOs and swift adaptation to sudden increases in demand from the end users. To start, each SBS b has a defined cache capacity of C_b. In the initial resource provisioning, a portion of each SBS is **reserved** for each MVNO and is denoted as $C_b^{res,v}$. These portions differ from one SBS to another based on the slice's unique QoS requirement and user demand for this slice at the SBS. However, the total reserved resource on each SBS must always be equal to the total capacity of the small cell.

$$\sum_{v \in \mathbf{V}} C_b^{res,v} = C_b, \quad \forall b \tag{1}$$

Once the DQN algorithm runs, a portion of the SBS cache resource is **allocated**, $C_b^{alloc,v}$, to the MVNO. This is the portion of caching resource that the DQN algorithm thinks is needed for a given MVNO. As with the reserved cache, the portions differ on each SBS, but the total allocated cache for each SBS should be equal to or less than the actual capacity of the small cell.

$$\sum_{v \in \mathbf{V}} C_b^{alloc,v} \leq C_b, \quad \forall b \tag{2}$$

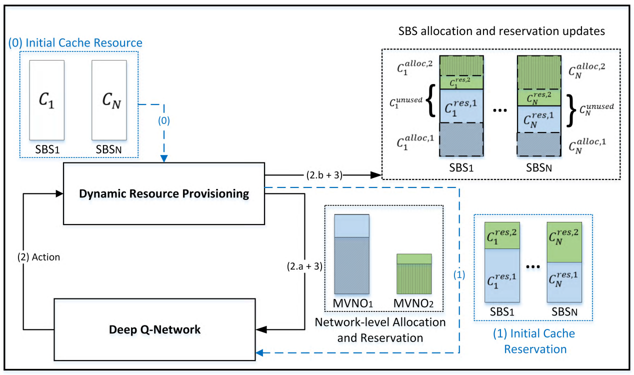

FIGURE 2. Virtualization model.

If the total allocated storage is less than the total caching capacity, then the **unused** cache resource C_b^{unused} is re-calculated and re-reserved for the MVNOs in accordance with their weights, which we cover in details in subsection IV.B.

The benefits of this scheme are as follow: 1) it guarantees performance isolation as each MVNO has its own reserved storage space that cannot be used by any other slice, and 2) if the user demand for one slice suddenly increases, the MVNO can use more than its allocated resource as long as it has a dedicated reserved resource. This results in faster adaptability to sudden environmental changes.

In this work, we propose a DRL-based solution implemented at the SDN controller at the mobile edge network and managed by the InP. Our virtualization model, in brief, consists of two algorithms: DRL and dynamic resource provisioning. These two algorithms cooperate to perform reservation and allocation duties, as illustrated in Fig. 2.

1) *Initial cache resource reservation:* By considering the QoS requirement of each slice and request demand for each MVNO, the dynamic resource provisioning algorithm develops certain weights and uses them to calculate the initial cache reservation; that is to reserve a certain percentage of the small cells' caching storage for each MVNO.
2) *Cache resource allocation:* The DRL algorithm makes the slicing decisions (actions) for all MVNOs. These actions either increase or decrease the total allocation for a slice. The resource provisioning algorithm takes these actions to (a) calculate the global allocation for each slice $C^{alloc,v}$, which is then fed back to the DRL algorithm, and (b) map these global allocation values to the SBS-level, making each SBS know how much of its resource to allocate to each slice $C_b^{alloc,v}$.
3) *Reservation refinement:* If the total allocation for all the slices at one SBS is less than its full capacity, the dynamic resource provisioning algorithm will recalculate its unused cache resource and reserve it back for the different slices. This also happens when the statistical information scouted from the network changes.

C. NETWORK MODEL

In our scenario, we consider a set of virtual slices (MVNOs) $\mathbf{V} = \{1, .., v, .., V\}$ and a set of small cell stations $\mathbf{B} = \{1, .., b, .., B\}$ in a given geographical area. Please note that an MVNO and a virtual slice in this work mean the same thing: the entity that owns and manages the content, and for the rest of this work, we will use these two terms interchangeably. In this paper, we assume that the entire infrastructure is owned by a single InP; however, the principles can be extended and applied to scenarios with multiple InPs. Each MVNO has its own unique set of content objects $\mathbf{O^v} = \{1, .., o^v, .., O^v\}$ with a content library size of L^v. Objects across all MVNOs have the same size O_s. Small cell stations, on the other hand, have the same caching capacity of C_b. This capacity is divided into equal-sized segments called slots. The size of a slot is equal to the size of an object. The aggregated sum of all the cache resources in the network is referred to as C_{total} (slots). Users in our scenario can only request content from one MVNO, so we define the users in relation to the MVNO they belong to as $\mathbf{U^v} = \{1, .., u^v, .., U^v\}$. The slice users $\mathbf{U^v}$ in the coverage area of a small cell station b are assumed to request the content following a Poisson process with an average rate of μ_{bv} (requests/s). The Poisson process is commonly used in the field of telecommunications and computer systems to model random arrival rates.

Content placement (aka *caching*) occurs when a content object of an MVNO occupies a slot in a small cell. The caching can be controlled by a binary parameter, Z^v_{ob}, such that $Z^v_{ob} = 1$ indicates that object o of virtual slice v is cached on small station b, and 0 otherwise. This variable can be used to mathematically formulate the following constraints:

$$\sum_{o \in \mathbf{O^v}} Z^v_{ob} \cdot O_s \leq C^{res,v}_b, \quad \forall b, v \tag{3}$$

$$\sum_{v \in \mathbf{V}} \sum_{o \in \mathbf{O^v}} Z^v_{ob} \cdot O_s \leq C_b, \quad \forall b \tag{4}$$

The constraint in (3) ensures that the total content cached at any small cell b for any MVNO v does not exceed its reserved caching space, while the constraint in (4) guarantees that the total content cached at any one small cell b for all MVNOs does not exceed its total physical caching storage.

Let $X^v_{ub} \in \{0, 1\}$ be the association binary variable, where $X^v_{ub} = 1$ denotes that the user u of MVNO v is associated with the small cell b, and 0 otherwise. We also assume that each small cell has a nominated spectrum bandwidth allocation of W_b (Hz) and the total spectrum bandwidth for the cellular network is noted as W_B. Let $Y^v_{ub} \in [0, 1]$ be the fraction of bandwidth allocated to the requesting user u of MVNO v that is associated with small cell b. The constraint in (5) ensures that the total channel bandwidth used by users to connect to one small cell b does not exceed the nominal spectrum allocation for said small cell

$$\sum_{v \in \mathbf{V}} \sum_{u \in \mathbf{U^v}} X^v_{ub} Y^v_{ub} W_b \leq W_b, \quad \forall b \tag{5}$$

A user's effective transmission rate R can be expressed as

$$R_{ub} = \sum_{b \in \mathbf{B}} X^v_{ub} Y^v_{ub} W_b sh_{ub} \tag{6}$$

where sh_{ub} is Shannon's achievable spectrum efficiency, which can be calculated as $sh_{ub} = \log_2(1 + SINR_{ub})$, and $SINR_{ub}$ is the SINR model which can be obtained from the expression

$$SINR_{ub} = \frac{g_{ub} P_b}{\sum_{k,k \neq b} g_{uk} P_k + \sigma} \tag{7}$$

where g_{ub} is the large-scale channel gain, σ is the power spectrum density of AWGN and P_b is the received signal power of user u from small cell b. The channel gain can be obtained by considering the path loss as

$$PL = 20 \log_{10}(dist) + 20 \log_{10}(F) + 32.4 \tag{8}$$

where $dist$ is the distance between the receiver and transmitter in (km) and F is the frequency band in (MHz). The shadowing small-scale fading effect is assumed to be a Gaussian random variable with zero mean and a standard deviation equal to $8dB$.

D. UTILITY MODELS

In this sub-section, we formulate two utility models, one for QoS satisfaction and one for resource utilization. This is the case because our objective in this work is to maximize a balance between satisfaction and resource utilization. The QoS utility model measures the level of satisfaction in an MVNO, which obviously depends on the slice's unique QoS requirement. The resource utilization model, on the other hand, measures the extent to which the physical cache resources are utilized. One assumption we are operating under is that the MVNOs will have different objective functions and/or QoS requirements, so in developing these models, we make sure that all the results are normalized and in the range of [0,1]. This normalization allows for a more fair comparison between the MVNOs.

1) QOS SATISFACTION

The overall QoS satisfaction utility function of one MVNO, Sat^v is the average sum of satisfaction at all the SBS's, which can be calculated as

$$Sat^v = \frac{1}{B} \sum_{b \in \mathbf{B}} Sat^v_b \tag{9}$$

where B is the total number of small cell stations and Sat^v_b is the satisfaction at one small cell, which is defined as the averaged satisfaction of all the users belonging to the slice v and requesting content from small cell b

$$Sat^v_b = \frac{1}{U^v_b \times O^v_b} \sum_{u \in \mathbf{U^v}} \sum_{o \in \mathbf{O^v}} Sat^v_{uo} \tag{10}$$

where U^v_b is the total number of users in slice v that are associated with small cell b, O^v_b is the total number of objects that are cached in small cell b, and Sat^v_{uo} is the satisfaction of user u requesting content object o. The calculations of Sat_{uo}^{v} vary depending on the type of the MVNO.

a: SLICES THAT PROVIDE NON-GUARANTEED RATE SERVICES

As MVNO3 and MVNO4 provide non-GBR, delay-sensitive services, their QoS requirement in terms of delay is denoted as q_d, which indicates the maximum allowed delay.

The download time depends on whether the content object is cached at a small cell b. So, let's first break down the download time needed for the content to reach the requesting user into two phases: *1) In-network downloading delay:* the time it takes for the data to travel through the backhaul network before reaching the small cell closest to the requesting user, and *2) wireless content delivery delay:* the time needed to offload the content from the closest small cell to the requesting user. We tackle each type of delay and finally formulate a universal delay expression.

The in-network download delay is a function proportionally related to the number of wired backhaul network hops the data travels through. Assuming the number of hops to be h and the delay time for each hop to be fixed as T, we obtain the in-network delay as $d_I = (T \times h_{uo})$; where h_{uo} is the number of hops between the user u and the original object o.

As for the wireless content delivery time, the user transmission rate we found in (6) can be translated into an equivalent downloading time metric, $d_{ub} = O_s/R_{ub}$, where d_{ub} is the time for user u to download content from small cell b. When an object o is cached in small cell b (that is $Z_{ob}^{v} = 1$), d_{uo} which is the time for user u to download content object o is identical to d_{ub}. Finally, the overall download delay time is formally calculated as follows:

$$d_{uo} = \begin{cases} O_s/R_{ub}, & \textit{if } Z_{ob}^{v} = 1 \\ (O_s/R_{ub}) + (T \times h_{uo}), & \textit{otherwise} \end{cases} \tag{11}$$

Now that we have the download time, we formulate the delay satisfaction function, $\xi(d_{uo})$, as

$$\xi(d_{uo}) = \frac{1}{1 + \exp\left(-\beta(q_d - d_{uo})\right)} \tag{12}$$

where β is the steepness constant. As such, the delay-sensitive QoS satisfaction function that will be used as our primary QoE metric for this type of slice is

$$Sat_{uo}^{v} = \xi(d_{uo}) \tag{13}$$

b: SLICES THAT PROVIDE GUARANTEED RATE SERVICES

As MVNO1 and MVNO2 provide GBR, delay-sensitive services, their QoS requirement in terms of delay is denoted as q_d as well, while their QoS requirement in terms of transmission rate is denoted as q_r, which indicates the minimum required transmission rate.

The delay satisfaction $\xi(d_{uo})$ is calculated with (12), while the rate satisfaction $\xi(R_{uo})$ is found by considering the user transmission rate R_{uo} in (6) in a sigmoid function as follows:

$$\xi(R_{uo}) = \frac{1}{1 + \exp\left(-\beta(R_{uo} - q_r)\right)} \tag{14}$$

where β is the steepness constant. Now that we have both elements, we create a combined rate and delay-sensitive QoS satisfaction function as our primary QoE metric, which is defined as the average satisfaction achieved for both rate and delay requirements, or as mathematically expressed as

$$Sat_{uo}^{v} = \frac{\xi(d_{uo}) + \xi(R_{uo})}{2} \tag{15}$$

2) RESOURCE UTILIZATION

Resource utilization is independent from the slice's QoS requirement, so its calculations for the different MVNOs are the same. The overall resource utilization of a slice, denoted as φ^{v}, is defined as the average sum of the utilization at each small cell, φ_b^{v}, which can be expressed as follows:

$$\varphi^{v} = \frac{1}{B} \sum_{b \in \mathbf{B}} \varphi_b^{v} \tag{16}$$

$$\varphi_b^{v} = \frac{1}{C_b^{alloc,v}} \sum_{o \in \mathbf{O^v}} Z_{ob}^{v} \tag{17}$$

where $C_b^{alloc,v}$ is the cache resource at small cell b that is allocated to slice v, and Z_{ob}^{v} is the binary caching variable.

IV. PROBLEM FORMULATION

A. DYNAMIC RESOURCE PROVISIONING

The dynamic resource provisioning algorithm is at the heart of our virtualization scheme, and it handles two important tasks: *1) cache resource reservation,* and *2) cache allocation updates*. In this subsection, we first develop two unique weights then delve into how each task is executed in details.

Weights are calculated based on the number of users connecting to the small cell and the MVNO's minimum required resource for a single user. The **minimum required resource** is referred to as $Req_{ub}^{min,v}$. This value is calculated for the non-guaranteed rate MVNOs as

$$Req_{ub}^{min,v} = \frac{O_s}{q_d} \tag{18}$$

where O_s is the object size and q_d is the MVNO's delay requirement. For the guaranteed rate MVNOs, it is calculated as

$$Req_{ub}^{min,v} = \frac{O_s}{q_d} + q_r \tag{19}$$

where q_r is the MVNO's rate requirement. It is important to note that $Req_{ub}^{min,v}$ does not estimate how much cache resource is needed for each MVNO, and it does not need to. This value is developed for weight calculations only to give a normalized indication of the relative importance of MVNOs and SBS to one another given the MVNO unique characteristics.

The first weight we develop is called *a slice weight*, denoted as I_b^{Slice}, and it defines the importance of a slice to a given SBS. That is to say, for all the MVNOs in the network, the SBS will dedicate more of its cache resources to the slices with higher I_b^{slice}. This weight is used to determine how much cache resource an SBS will *reserve* for each respective MVNO. For an SBS, this **weight** is the total minimum resource required by one slice's users to the total minimum resource required by all the users connected to this SBS. This can be expressed as:

$$I_b^{Slice} = \frac{\sum_{u \in \mathbf{U}^v} Req_{ub}^{min,v}}{\sum_{v \in \mathbf{V}} \sum_{u \in \mathbf{U}^v} Req_{ub}^{min,v}} \tag{20}$$

The second weight is *an SBS weight*, denoted as I_v^{SBS}, and it defines the importance of the local SBS's to any given slice. That is to say, considering all the SBS's that can serve the slice, the ones with bigger weights are required to *allocate* more of their cache resources to this slice in comparison to other SBS's. This weight is used to map the global allocation decisions of a slice to the local SBS's. For one SBS, this **weight** is calculated by aggregating the minimum required resource for the slice's users that are connected to this SBS by the total minimum required resource by all users of this nominated slice in all the SBS's. Mathematically, this can be expressed as:

$$I_v^{SBS} = \frac{\sum_{u \in \mathbf{U}^v} Req_{ub}^{min,v}}{\sum_{b \in \mathbf{B}} \sum_{u \in \mathbf{U}^v} Req_{ub}^{min,v}} \tag{21}$$

1) CACHE RESOURCE RESERVATION

Throughout the management period, the **unused** cache at any SBS will be reserved for different slices based on various network info and QoS demands. The dynamic resource provisioning algorithm collects network data and calculates the minimum required resource $Req_{ub}^{min,v}$ and slice weight I_b^{Slice}. Using these weights, the cache resource is reserved in each small cell with this expression:

$$C_b^{res,v} = I_b^{Slice} C_b^{unused} \tag{22}$$

At the very beginning of the running time, we call this *initial cache resource reservation*. At this stage, the DRL algorithm has not run yet, so no slicing decisions have been made and caching resources at the SBS's are untouched. As such, the unused cache resource is $C_b^{unused} = C_b$.

At any other time, we can refer to the same task as *resource reservation refinement*. This can be triggered by an environmental change (*e.g. a change in user distribution or request demands*) or when the total allocation at one SBS is less than its full capacity. Either way, the remaining unused cache resource at each SBS is calculated as $C_b^{unused} = C_b - \sum_v C_b^{alloc,v}$.

2) CACHE ALLOCATION UPDATES

When the DRL algorithm makes a slicing decision, it basically takes an action, a^v, that changes the overall amount of cache resource that is allocated to the MVNO. The dynamic resource provisioning takes this action and translates it into two allocation values, one for the slice-level, and one for the SBS-level. Let the overall allocation of cache resource to a slice v before taking an action be $C^{alloc,v}$, while after it is $C^{*alloc,v}$. The slice-level resource allocation is updated as follows:

$$C^{*alloc,v} = \begin{cases} C^{alloc,v}, & \text{if } a^v = 0 \\ (1 + a^v) \times C^{alloc,v}, & \text{if } a^v < 0 \\ C^{alloc,v} + (a^v \times C^{res,v}), & \text{if } a^v > 0 \end{cases} \tag{23}$$

Now, this global allocation needs to be mapped to the SBS. Each small cell needs to know how to allocate its cache resources to the slices accordingly, and this is done by applying the SBS weight I_v^{SBS} in the following equation:

$$C_b^{alloc,v} = I_v^{SBS} C^{alloc,v} \tag{24}$$

The algorithm is summarized in **Algorithm 1**.

Algorithm 1 Dynamic Cache Resource Provisioning

```
1:  Initialize resource reservation and allocation to zeros
2:  if $t = 0$ then
3:      Calculate $Req_{ub}^{min,v}$
4:      Calculate the slice weight $I_b^{Slice}$ with (20)
5:      Reserve resource at local SBS's with (22)
6:  else
7:      Import the slice action from Algorithm 2
8:      Calculate $Req_{ub}^{min,v}$
9:      Calculate the slice weight $I_b^{Slice}$ with (20)
10:     Calculate the SBS weight $I_b^{SBS}$ with (21)
11:     Update the slice's overall resource allocation
        $C^{alloc,v}(t+1)$ with (23)
12:     Map the overall resource allocation to the local SBS's
        with (24)
13:     Update $C_b^{unused}$ on each SBS and calculate new reser-
        vation with (22)
14: end if
```

B. CACHE RESOURCE SLICING (DQN)

Right after the initial resource reservation, the DRL controller dynamically and autonomously attempts to optimize the cache resource allocation in order to maximize the QoS satisfaction and the resource utilization for all the MVNOs.

1) DEEP Q-LEARNING

Reinforcement Learning is a very successful branch of machine learning where an agent interacts with an environment and independently learns how to find the best actions to perform through trial and error. The agent observes the state of the environment (s) and takes an action (a) to change the environment to its advantage. If successful, it earns a reward (r), but if not, it will be penalized (*in the form of a negative reward*). The objective of the agent is to find the optimal policy $\pi^*(s, a)$, which defines what action should be taken in any given state.

The **objective function** of our slicing controller is to find the optimal policy that maximizes the expected cumulative reward.

$$\pi^* = argmax_\pi E[\sum_{t=0}^{\infty} \gamma^t r(t)] \tag{25}$$

where γ is the reward discount function. Of the many available RL algorithms, we find Q-learning the most appealing because 1) it is a model-free algorithm that does not require known transition probabilities, which is perfect for real-life complex problems like ours [22], and 2) it is proven that with the use of a proper learning rate, the Q-function will always converge [23]. According to the Bellman equation, there exits at least one strategy that satisfies the optimality conditions, and it is defined as

$$Q(s, a) = r + \gamma max_{a'}(s', a') \tag{26}$$

where a' and s' are future actions and states, respectively. However, since the state-action space in our case is large, estimating each Q value is computationally infeasible, so we use function approximators such that $Q(s, a; \theta) \approx Q^*(s, a)$, where θ is a function parameter.

While there are quite a few options for a function approximator, we use deep neural networks (DNNs) [24] as they have the advantage of not requiring hand-crafted features. They also have proven successful in solving large RL tasks [22], [25]. We use the Least Square algorithm to optimize the function parameter by minimizing the sum-squared error between the approximation (prediction) and target value.

$$loss = (r + \gamma max_{a'}(s', a') - Q(s, a)) \tag{27}$$

When choosing a neural network as an estimator, the new network is called Q-network. In a Q-network, the parameter θ is a network parameter, and the network is trained by adjusting these parameters.

That said, the Q-network is not perfect. First, it was observed that the Q-network quickly learns the patterns of a training period set, and as a result, has a tendency to reach a local minimum instead of a global one. Additionally, the Q-network is quite unstable because the same neural network is used to calculate the prediction and target values.

Fortunately, these shortcomings can be overcome with the use of a DQN. A DQN significantly improves the performance of a Q-network by using experience replay and adding a target network. First, with experience replay, random sets of the training data are chosen to train the network every once in a while. This breaks the patterns in a limited data set and prevents the network from reaching a local minimum. Second, by adding a second NN to calculate the target Q-values with independent network parameters θ that are only updated every few steps instead of every single step, the network becomes more stable. Fig. 3 illustrates how our proposed DQN works.

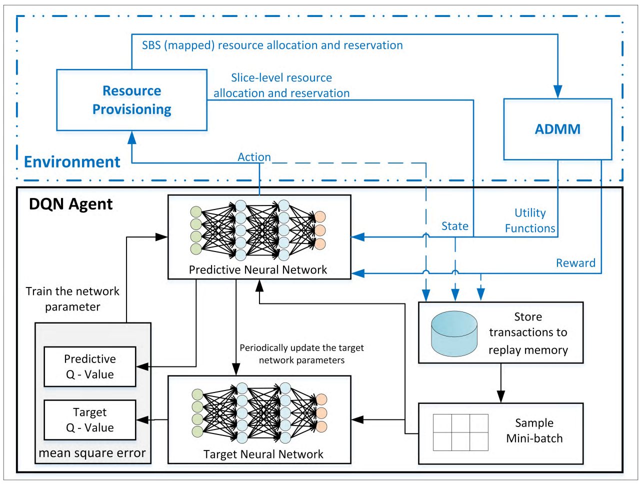
SBS (mapped) resource allocation and reservation
Resource Provisioning
Slice-level resource allocation and reservation
ADMM
Environment
Action
DQN Agent
State
Utility Functions
Predictive Neural Network
Reward
Train the network parameter
Periodically update the target network parameters
Store transactions to replay memory
Predictive Q - Value
Target Q - Value
mean square error
Target Neural Network
Sample Mini-batch

FIGURE 3. Deep Q-Network.

2) OUR MDP MODEL

The environment in our work is defined using Markov decision process (MDP) as follows:

a: STATES (s)

Let the state of the system for a particular MVNO v at time t be represented with

$$s^v(t) = [\varphi^v(t), Sat^v(t), C^{res,v}(t), C^{alloc,v}(t)] \tag{28}$$

where $\varphi^v(t)$ is the resource utilization of a slice v at time t, $Sat^v(t)$ is the QoS satisfaction utility function, $C^{res,v}(t)$ is the total reserved cache resource for the slice, and $C^{alloc,v}(t)$ is the total allocated cache resource.

b: ACTIONS (a)

At each time step, the learning agent needs to take an action to win a reward. The actions that the system as a whole takes at time t can be represented as

$$a(t) = \{a^{v1}(t), a^{v2}(t), a^{v3}(t), a^{v4}(t)\} \tag{29}$$

Which means that our DQN algorithm makes a total of four actions at each time step, one action per MVNO. The action for each slice is either to increase or decrease the current total cache resource allocation. We define the set of possible actions for any slice as $a^v(t) = \{-50\%, -40\%, -30\%, -20\%, -10\%, 0\%, 10\%, 20\%, 30\%, 40\%, 50\%\}$. As the neural network does not understand percentages, we change this set into fractional values as $a^v(t) = \{-0.5, -0.4, -0.3, -0.2, -0.1, 0, 0.1, 0.2, 0.3, 0.4, 0.5\}$, and each action is identified by its index in the NN as $\{a_{-5}, a_{-4}, a_{-3}, a_{-2}, a_{-1}, a_0, a_1, a_2, a_3, a_4, a_5\}$, where $a_{-5} = -0.5$ and $a_5 = 0.5$. There are two conditions here though:

1) *Physical cache limitation:* The total cache resource allocated for all the MVNOs in one small cell must not exceed its actual cache capacity. That is $\sum_{v \in \mathbf{V}} C_b^{alloc,v} \le C_b$.

2) *System resource limitation:* The total cache allocated for all the MVNOs in all the small cells must not exceed the total capacity of the system. That is $\sum_{b\in\mathbf{B}}\sum_{v\in\mathbf{V}} C_b^{alloc,v} \leq C_{total}$.

c: REWARDS (r)

As the objective of the system is to maximize a balance between the QoS satisfaction and the utilization of resources, the overall reward function for the system is the accumulative reward achieved for each MVNO, and it is formulated as follows

$$r(t) = \sum_{v\in\mathbf{V}} \omega Sat^v(t) + (\omega - 1)\varphi^v(t) \tag{30}$$

where ω is the satisfaction weight, and it controls which utility function is given more priority. We prioritize satisfaction in this work, and as such $\omega > 0.5$ is always used.

d: NEXT STATE (s′)

After the reward is gained from taking a particular action, the agent enters into the next state, and the original state information tuple is changed. The new QoS satisfaction and resource utilization that were computed to get the reward will be used in the new state's tuple as well as the updated cache allocation and reservation values.

e: NEURAL NETWORK CONFIGURATION

We configure a feed-forward neural network with 4 inputs and 11 outputs because the state has 4 variables and there is a total of 11 actions. There is no known rule for determining the number of hidden layers and neurons, and it is far more efficient to choose the size by trial and error while considering the computational cost against the performance. We use two hidden layers with 18 neurons each. We arbitrarily set the activation function in the hidden layers to be a sigmoid function. **Algorithm 2** illustrates how our DQN-based cache slicing scheme works. The computational complexity of Algorithm 2 is of $O(HN)$, where H denotes the number of hidden layers and N represents the number of neurons in the artificial neural network.

Algorithm 2 Cache Resource Slicing With DQN Algorithm

```
1:  Initialize learning parameters
2:  Initialize replay memory $D$
3:  Initialize the Q-network with random network parameters $\theta$
4:  Initialize the target Q-network with network parameters $\theta^- = \theta$
5:  for episode = 1, ..., E do
6:      while learning period do
7:          Observe state $s(t)$
8:          With a probability $\epsilon$, select a random action $a(t)$
9:          Otherwise, use Q-network and select $a(t) = max_a Q(s, a; \theta)$
10:         Export $a(t)$ to Algorithm 1 and update resource allocation and reservation
11:         Observe the reward $r(t)$ and next state $s(t+1)$
12:         Store transaction $< (s(t), a(t), r(t), s(t+1) >$ in $D$
13:         if (timer activated) then
14:             Sample random mini-batch of transactions $< (s(j), a(j), r(j), s(j+1) >$ from memory $D$
15:             $y_j = r + \gamma max_{a(j+1)}(s(j+1), a(j+1))$
16:             Train the function parameter with a loss function as the mean square error of the output and target, $loss = (y_j - Q(s, a))$
17:             Update target parameters every C steps $\theta^- \leftarrow \theta$
18:         end if
19:     end while
20: end for
```

C. CUSTOMIZED CONTENT PLACEMENT (ADMM)

After the cache resource has been sliced and allocated to the different MVNOs, a customized content placement algorithm is carried out for each MVNO based on its QoS requirements. This intra-slice content placement is formulated as a convex optimization problem to maximize the QoS satisfaction of each MVNO and is solved with a distributed ADMM. ADMM is commonly used in the field of wireless communications because it reaches decent optimality in a reasonable number of iterations [11], [26], [27].

In this work, we make the assumption that one user can associate with multiple small cells. We assume that the popularity profile of the files is characterized by a Zipf-like distribution with a parameter δ which is commonly used to model content popularity in networks. Hence, the popularity of a content object of a certain MVNO's object, o^v, in period t, is $p_o = 1/(o^\delta \sum_{t=1}^{O} 1/t^\delta)$. To formulate the optimization problem, the general objective function for any MVNO is written as follows

$$max \sum_{b\in\mathbf{B}}\sum_{u\in\mathbf{U^v}}\sum_{o\in\mathbf{O^v}} p_o \cdot Sat_{uo}^v \cdot X_{ub}^v \cdot Z_{ob}^v \tag{31}$$

1) MVNO CUSTOMIZATION

We use (13) in (31) to express the customized objective function for non-guaranteed rate slices (*MVNO3-4*) as

$$max \sum_{b\in\mathbf{B}}\sum_{u\in\mathbf{U^v}}\sum_{o\in\mathbf{O^v}} p_o \cdot \xi(d_{uo}) \cdot X_{ub}^v \cdot Z_{ob}^v \tag{32}$$

$$\text{s.t.} \quad \sum_{o\in\mathbf{O^v}} Z_{ob}^v \cdot O_s \leq C_b^{res}, \quad \forall b \tag{32.a}$$

$$\sum_{b\in\mathbf{B}}\sum_{u\in\mathbf{U^v}}\sum_{o\in\mathbf{O^v}} Z_{ob}^v \cdot d_{uo} \leq q_d, \quad \forall b, u, o \tag{32.b}$$

$$\sum_{b\in\mathbf{B}} Z_{ob}^v \leq \Upsilon, \quad \forall o \tag{32.c}$$

where constraint (32.a) ensures that the number of placed objects of an MVNO does not exceed its reserved value in a small cell, constraint (32.b) ensures that the delay requirements are met, and constraint (32.c) allows the network to make more than one copy of each content object.

As for the guaranteed rate slices (*MVNO1-2*), we use (15) in (31), and write their customized objective function as

$$max \sum_{b\in\mathbf{B}}\sum_{u\in\mathbf{U^v}}\sum_{o\in\mathbf{O^v}} p_o \cdot \frac{\xi(d_{uo})+\xi(R_{uo})}{2}\cdot X^v_{ub}\cdot Z^v_{ob} \tag{33}$$

$$\text{s.t.} \quad \sum_{o\in\mathbf{O^v}} Z^v_{ob}\times O_s \le C^{res}_b, \quad \forall b \tag{33.a}$$

$$\sum_{b\in\mathbf{B}}\sum_{u\in\mathbf{U^v}}\sum_{o\in\mathbf{O^v}} Z^v_{ob}\cdot d_{uo}\le q_d, \quad \forall b,u,o \tag{33.b}$$

$$\sum_{b\in\mathbf{B}}\sum_{u\in\mathbf{U^v}}\sum_{o\in\mathbf{O^v}} Z^v_{ob}\cdot R_{uo}\le q_r, \quad \forall b,u,o \tag{33.c}$$

$$\sum_{b\in\mathbf{B}} Z^v_{ob}\le \Upsilon, \quad \forall o \tag{33.d}$$

where constraint (33.a) ensures that the number of placed objects of an MVNO does not exceed its reserved value in a small cell, constraints (33.b) and (33.c) ensure that the delay and rate requirements are met, and constraint (33.d) allows the network to make more than a copy of each content object.

2) PROBLEM REFORMULATION

We look to implement a distributed solution with a global consensus problem, so we note that $\{Y^v_{ub}\}$ and $\{Z^v_{ob}\}$ are separable and thus can be solved locally at different small base stations, but the association variable $\{X^v_{ub}\}$ needs to use a central controller. So, we introduce the local variables $\{y^v_{ub}\}$ and $\{z^v_{ob}\}$ as direct maps from the global ones. These two variables are not only separable across each base station, but also only affect said node. As for the association variable, we introduce $\mathbf{X}$ as the matrix of association variables, where the local opinion of the association indicator between the local small cell l and user u is represented as x^l_{ub}.

For the lack of space, we reformulate the problem for the guaranteed rate slices only. The same can be applied for the non-guaranteed rate slices by simply removing the rate-related terms, when applicable.

First, the global objective and constraints are split into B terms, each term can also encode constraints by assigning $+\infty$ when a constraint is violated.

$$Sat^l_b = \begin{cases} \sum_U\sum_O p_o\cdot x^l_{ub}\cdot z_{ob}, & x_b,y_b,z_b\in\Theta_b \\ \infty, & otherwise \end{cases} \tag{34}$$

where Sat^l_b is the local utility function at small cell b, $\{x_b, y_b, z_b\}$ are the local opinion of a small cell on the association, the friction of radio resource, and caching, respectively, and Θ_b is the feasible local set variable at each SBS.

The reformulated problem can now be written as

$$min \sum_{b\in\mathbf{B}} Sat^l_b(x_b,y_b,z_b)$$
$$\text{Subject to} \quad x^l_{ub}=\chi^l_u, \quad \forall b,u,l \tag{35}$$

with the constraint imposed to allow parallelism, where each local opinion of the association variable needs to agree with the noted χ^l_u.

The ADMM for the reformulated problem can be derived directly from the augmented Lagrangian

$$\mathcal{L}_\rho\big(\{x_b,y_b,z_b\}_{b\in B},\{\mathbf{X}\}\{\lambda^b\}\big) = \sum_{b\in\mathbf{B}} Sat^l_b(x_b,y_b,z_b) + \sum_{\substack{l\in\mathbf{B}\\u\in\mathbf{U}}}\lambda^{b[t]T}_{ul}(x^l_{ub}-\chi^{l[t]}_u)+\frac{1}{B}\sum_{\substack{l\in\mathbf{B}\\u\in\mathbf{U}}}(x^l_{ub}-\chi^{l[t]}_u) \tag{36}$$

where λ^b_{ul} is the Lagrange multiplier corresponding to the consensus constraint and ρ is a constant penalty parameter. The distributed ADMM iterations are as follows. First, local variables are updated at each small cell station:

$$\{x_b,y_b,z_b\}^{[t+1]}_{b\in B} = argmax\Big\{Sat^l_b(x_b,y_b,z_b) + \sum_{\substack{l\in\mathbf{B}\\u\in\mathbf{U}}}\lambda^{b[t]T}_{ul}(x^l_{ub}-\chi^{l[t]}_u)+\frac{\rho}{2}\sum_{\substack{l\in\mathbf{B}\\u\in\mathbf{U}}}(x^l_{ub}-\chi^{l[t]}_u)^2\Big\} \tag{37}$$

Second, the global association matrix is aggregated and updated at the ADMM controller:

$$\{\mathbf{X}\}^{[t+1]}=\frac{1}{B}\sum_{b\in\mathbf{B}}\sum_{\substack{l\in\mathbf{B}\\u\in\mathbf{U}}}\Big((x^{l[t+1]}_{ub}+\lambda^{b[t]}_{ul}/\rho)\Big) \tag{38}$$

Third, the Lagrangian multipliers are updated on each SBS:

$$\{\lambda^{b[t+1]}_{ul}\}_{\substack{l\in\mathbf{B}\\b\in\mathbf{B}}}=\lambda^{b[t]}_{ul}+\rho(x^{l[t+1]}_{ub}-\lambda^{l[t+1]}_u) \tag{39}$$

The first and third Iterations in (37) and (39), respectively, can be done separately on each SBS while the second iteration in (38) takes place at the ADMM controller. Iteration (37) yields the optimal cache placement, and it also gives the SBS's opinion of the global association variable. At the controller, the local opinions of x are collected, and then the global association variable is retrieved. After the controller updates the x values, it broadcasts them back to base stations, which then will update their Lagrange multipliers. These iterations keep happening until the stoppage criteria are met, which is decided by the controller.

Termination Criteria: The algorithm terminates when the primal residual $\|pri^{[t]}\|_2$ and the dual residual $\|dual^{[t]}\|_2$ are smaller than τ values. This value determines the optimality of the solution as well as the number of iterations it will take to find it. The termination criteria are formally expressed as:

$$\|pri^{[t]}\|_2\le\tau^{pri} \quad \text{and} \quad \|dual^{[t]}\|_2\le\tau^{dual} \tag{40}$$

where

$$\|pri^{[t]}\|^2_2=\sum_{\substack{b\in\mathbf{B}\\l\in\mathbf{B}}}\|x^{l[t]}_{ub}-\chi^{l[t]}_u\|^2_2 \tag{41a}$$

$$\|dual^{[t]}\|^2_2=B\rho^2\sum_{l\in\mathbf{B}}\|\chi^{l[t]}_u-\chi^{l[t-1]}_u\|^2_2 \tag{41b}$$

The customized intra-slice optimization algorithm is outlined in **Algorithm 3**. Since each SBS only solves its local optimization subproblem, the computational complexity at each SBS is $O(U^k)$, where $k = 1$ denotes a linear algorithm and $k > 1$ denotes a polynomial time algorithm. The ADMM controller finds the global solutions with a computational complexity of $O((B+1)U)$ and updates the dual-variables with a complexity of $O((B+1)U)$ as well. As a result, for a J number of iterations, the total computational complexity for Algorithm 3 is $J(O((B+1) \cdot U^k) + 2(B+1)U) = J(O((B+1) \cdot U^k))$.

Algorithm 3 Distributed ADMM for Intra-Slice Content Placement

```
1:  Initialize $\mathbf{X}^0 \in \chi$, $\lambda^0 > 0$, $\rho^0 > 0$, and $\tau$
2:  Import $C_b^{alloc,v}$ from the **Algorithm 1**
3:  Broadcast $\mathbf{X}^{[t]}$, $\lambda^{[t]}$, $\rho^{[t]}$, and $C_b^{alloc,v}$ to each SBS;
4:  **for** t = 0,1,2,... **do**
5:      Each SBS $b$ solves problem (37) to update $(x_b, y_b, z_b)_{b \in B}^{[t+1]}$ and sends results to the controller
6:      The controller updates $\mathbf{X}^{[t+1]}$ by computing (38)
7:      **if** termination criteria (40) are met, **do** go to step 10
8:      The controller broadcasts $\mathbf{X}^{[t+1]}$ to all SBSs
9:      Each SBS $b$ updates $\lambda^{[t+1]}$ with (39)
10: **end for**
11: Calculate the slice's QoS satisfaction with (9)
12: Output the optimal content allocation policy and QoS Utility Function $Sat^v$
```

V. RESULTS AND EVALUATION

In this sub-section, we first detail the settings of our experiment and then discuss the results we obtained. We evaluate four types of results: The convergence of the DQN algorithm, the cache resource provisioning, algorithm comparison, and performance isolation.

A. SCENARIO CONFIGURATION

We use computer simulations to show the performance of the proposed scheme. We run the simulations in an X86 GPU-based server with 4 Nvidia GPUs. The CPU is Intel Xeno E5-2630 v4 with 32G RAM, and the operating system is Microsoft Windows 8.1. The software environment is TensorFlow 1.13.1 with Python 3.7. The mobile network scenario is based on previous work in the field of cache allocation in a wireless network [17]. In a given area of *1000m*1000m*, 4 SBS's belonging to one InP are uniformly scattered. Each SBS has a transmit power of 30dBm and is equipped with a caching capacity of $C_b = 3000$ slots. These slots are equal in size and can host one content object. By using a channel frequency of 2.4 GHz in (8), the path-loss (PL) is then defined as $PL = 100 + 20\log_{10}(dist)$, with $dist$ (*in km*) denoting the distance between the user and the SBS. In our simulation, we consider one InP and four MVNOs sharing

TABLE 1. Simulation parameters.

Parameter	Setting
System bandwidth	20 MHz
Frequency	2.4 GHz
coverage area	1000m x 1000m
Number of SBS's, B	4
SBS's caching capacity, C_b	3,000 slots
Transmission Power, Tx	30dBm
Path loss	$100 + 20\log_{10}(dist(km))$
Noise spectral density	-174 dBm/Hz
Shadowing	Log-normal distribution with a standard deviation of 8dB
Number of slices, V	4
Delay requirement, q_d	[120; 90; 250; 200](ms)
Rate requirement, q_r	[100; 150; N/A; N/A](kpbs)
Number of users, $\mathbf{U}^v$	[5-50; 5-50; 10-100; 10-100]
Content Library size, L^v	[500; 500; 1500; 1500]
Object size, O_s	0.5 Mbyte
Request arrival rate, μ_{bv}	[1, 15]
Zipf parameter, δ	[0.3, 1.2]
ADMM penalty parameter, ρ	1
Learning rate, α	0.01
Exploration Rate, ϵ	0.93
Discount Factor, γ	0.9
Size of replay memory, D	10,000
Size of mini-batch, D'	64
Slicing period	200ms
Episode duration	2s
Simulation duration	2000s
Training function	Bayesian

the common physical infrastructure. The requirements of the different slices in our scenario are derived from the QCI index in [21]. The first and second slices provide guaranteed rate, delay-constrained services. According to the index, the budget delay for such services is 150ms, so for our experiment, we define the delay requirements as 120ms and 90ms and the rate requirements as 100kpbs and 150kpbs for MVNO1 and MVNO2, respectively. The third and fourth slices provide non-guaranteed rate, delay-constrained services. According to the index, the budget delay for such services is 300ms, so we set the maximum allowed delay as 250ms and 200ms for MVNO3 and MVNO4, respectively. The first two slices will have a content library size of 500 objects each, while the last two will have a content library size of 1,500 objects each. Each object is 0.5 Mbytes in size. The first two slices will also have fewer requesting users than the last two, with the range of requesting users set as (5–50) for MVNO1 and MVNO2 and (10–100) for MVNO3 and MVNO4. The average request rates for each MVNO are randomly chosen in the range of [1, 15], and content popularity follows a Zipf distribution with the Zipf parameter set as $\delta \in [0.3, 1.2]$. The ADMM algorithm uses a constant penalty parameter $\rho = 1$. As for the DQN algorithm, we define a learning rate of 0.01, a discount factor of 0.9, an epsilon-greedy exploration rate of 0.93, a replay memory of size 10,000, and a mini batch size of 64. The simulation parameters are summarized in Table. 1. Finally, unless explicitly stated otherwise, these simulation parameters are used to obtain all the results in this section.

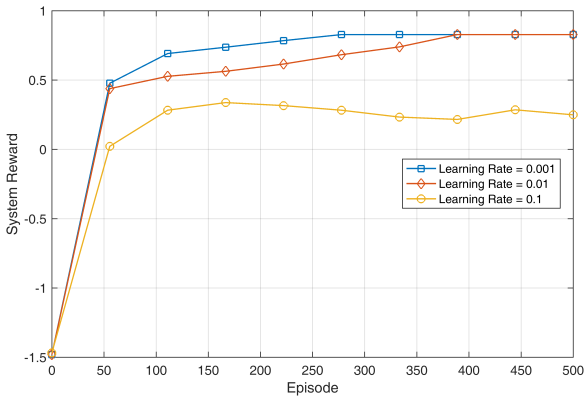

FIGURE 4. DQN reward convergence.

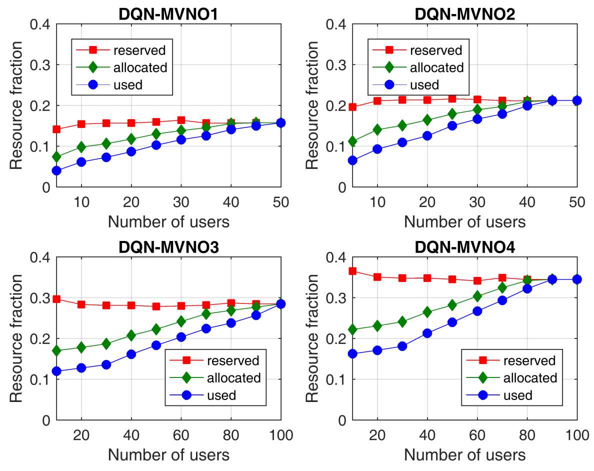

FIGURE 5. The cache resource provisioning.

B. THE CONVERGENCE OF THE DQN

In this experiment, we study the convergence of the proposed DQN scheme and the effect of the learning rate parameter, α, on it. We run the experiment for 500 episodes, and then the results are smoothed by taking the maximum result from every 50 episodes. The performance of the DQN algorithm is measured in reward, which is computed for the entire system by aggregating a balanced summation of resource utilization and QoS utility function for all MVNOs as in (30). We can see in Fig. 4 that the total rewards for all the learning rate parameters are very low at the beginning of the experiment. However, as the training continues and the number of episodes increases, the total rewards increase until they saturate at a relatively stable values. These values differ according to the used learning rate parameter. When using a learning rate of 0.1, the total reward stabilizes at around 0.25, which is the lowest reward among the three parameters we tested. The remaining two learning rates coverage to a total reward of 0.8, but they do so at different speeds. A learning rate of 0.001 reaches this value after only 270 training episodes, but this learning rate is prohibitively costly to run. A learning rate of 0.01 reaches the same saturation value of 0.8 after 370 episodes and is far more affordable to run.

Taking the convergence and the computational cost into consideration, we use the learning rate of 0.01 for the rest of this simulation. Additionally, in order to draw more sensible conclusions from our experiments and to better compare our results to other benchmarks, we smooth our findings by reporting the averaged values of the converged results returned only in the 400-499th episodes.

C. THE CACHE RESOURCE PROVISIONING

In this subsection, we investigate the resource provisioning of our proposed solution. To smooth the results of this experiment, we only consider the averaged results from the last 100 episodes of training. We look at the cache resource that the system reserves, how much of it is allocated by the DQN-based solution, and how much is used by the slice for an increasing number of requesting users. Initially, the cache resource is reserved for each slice based on the user demand and its unique QoS requirements. Then, the DQN algorithm makes allocation decisions, and these decisions are used to update the system-level and SBS-level allocation as well as to adjust the reservation values as discussed in subsection IV.A. The sum of reserved resources for the four MVNOs is always equal to 1, while the sum of allocated resource fractions is less than or equal to 1. For MVNO1 and MVNO2, the number of users increases from 5 to 50, and for MVNO3 and MVNO4, the number of users increases from 10 to 100 users.

The first observation we can make by looking at Fig. 5 is that our DQN-based solution is able to closely follow the increase in the demand for resources in each slice and allocate cache resources efficiently. At the beginning of the experiment, and when the number of requesting users is small (i.e. 5 requesting users for MVNO1-2 and 10 requesting users for MVNO3-4), the algorithm allocates a small but sufficient amount of cache resources to the MVNOs. The allocation values in this instance are [MVNO1: 0.0742, MVNO2: 0.1121, MVNO3: 0.1704, and MVNO4: 0.2223], and this takes the total resource allocation for the system to 0.5790 of the available cache resource. The remaining resource is reserved for the different slices in accordance with (22). As the number of requesting users increases, our DQN-based solution continues to follow this upward trend and allocates more resources to meet the increasing demand. This continues until the system allocates all of its available physical resources. For MVNO1 and MVNO2, this happens with 50 and 45 users, respectively, while for MVNO3 and MVNO4, it happens with 100 and 90 users, respectively. The values of the allocated resources at these instances are [MVNO1: 0.1576, MVNO2: 0.2123, MVNO3: 0.2850, and MVNO4: 0.3451]. This totals to 1, which is the total capacity of the system. This highlights the fact that the allocated resource determined by our solution is always able to adapt to and meet the increasing demands of the users until all physical resources are depleted. We also note that the gap between the reserved and allocated resource gets smaller with the increasing number of users until the two completely merge when demand is in its highest points by the end of the experiment.

This experiment also gives us insight as to how the cache resources were distributed among the four slices. We can easily see in Fig. 5 that the non-guaranteed rate slices (MVNO3-4) get a larger share of the available cache resources. For example, in the last training instance, MVNO3-4 use 0.6301 of the system's resources, while MVNO1-2 only use 0.3699. This is not surprising considering that they have a content library triple the size of their counterparts in the first two slices and twice as many requesting users. Furthermore, we note that among the slices of the same type, MVNO 2 uses more resources than MVNO1 and MVNO4 uses more resources than MVNO3. This is the case because even though each pair has the same objective function, MVNO2 and MVNO4 have more stringent QoS requirements as illustrated in Table. 1.

D. ALGORITHM COMPARISON

In this section, we evaluate the performance of our proposed DQN-based solution against two benchmarks: ADMM and PACP. This performance comparison is carried out in two parts; first on a slice-level and then on a system-level.

1) ALTERNATING DIRECTION METHOD OF MULTIPLIERS (ADMM)

For the first benchmark, we use a joint resource slicing and in-network caching system with a distributed ADMM as introduced in [6]. In this solution, the cache slicing and content placement problems are intertwined, and the system caches the content immediately into the sliced resource. The objective function developed in the referenced work is the maximization of a cost utility function. This objective function, without loss of generality, is defined as $max \sum Xu(a^{-}Y + b^{-}Z)$ where a^{-} is the net revenue of allocating radio resource, while b^{-} is the expected saved cost from the alleviation of backhaul bandwidth by caching the content. The four slices we use in this experiment will retain their different QoS requirements, but when solving for this benchmark, this one cost function will apply to all of them.

2) POPULARITY-AWARE CACHE RESOURCE PROVISIONING (PACP)

For this benchmark, we used a modified version of the algorithm introduced in [28]. In this method, the ranks of slices on each SBS is determined by calculating weights that consider the popularity of the slice's content objects and the number of its users associated with each SBS. The rank of a slice at one small cell is derived as $rk_b^v = \sum_{U^v} \sum_{O^v} p_o Z_{ob}^v / L^v$ where p_o is the popularity of the object, Z_{ob}^v is the binary caching variable, and L^v is the content library size of slice v. The portion of resources allocated to a slice on one SBS is then found as $C_b^{alloc,v} = (rk_b^v / \sum_v rk_b^v) * C_b$, and the total allocated resources in this case will always be equal to the full capacity of a small cell. This method overwhelms the network with the

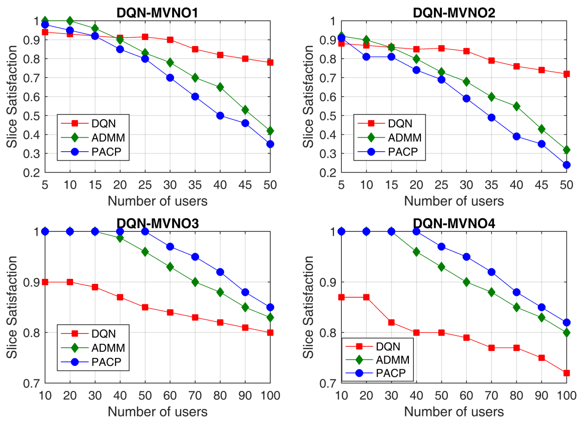

FIGURE 6. Slice QoS satisfaction.

most popular and in-demand objects in order to maximize the user satisfaction, but it does not consider resource utilization.

3) PERFORMANCE ANALYSIS ON THE SLICE LEVEL

The performance is analyzed in terms of slice satisfaction and slice resource utilization. Slice satisfaction is the average sum of the satisfaction of all the users belonging to the slice that request content objects from any SBS, and it is calculated with (9) and (10). In Fig. 6, the y-axis represents the satisfaction level, while the x-axis is the number of requesting users. Slice resource utilization, on the other hand, is calculated by considering how much of the allocated resource is actually used by the slice with (16) and (17). In Fig. 7, the y-axis, which is the resource utilization axis, represents the total allocation of the cache resource for the respective MVNO, while the x-axis is the number of requesting users. The two performance metrics are evaluated for an increasing number of users. In this subsection, we first look at each performance metric independently before discussing how they impact one another.

The first observation we can make from Fig. 6 is that our DQN solution is the only one that is able to provide

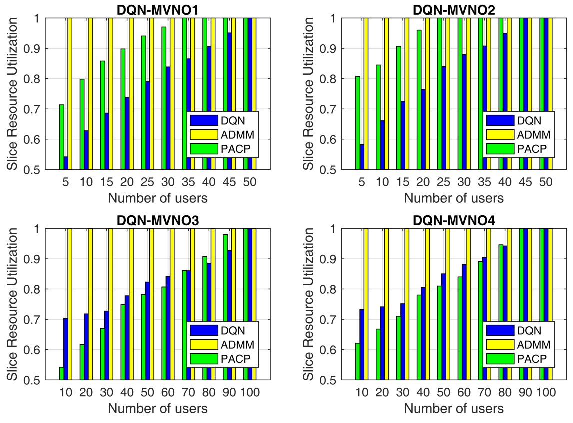

FIGURE 7. Slice resource utilization.

acceptable satisfaction levels for the four slices. The satisfaction level for the DQN algorithm never drops below 0.72; however, both ADMM and PACP satisfaction results plummet below 0.5 for both MVNO1 and MVNO2. This demonstrates our system's ability to account for different QoS requirement and to react accordingly. We can also clearly see that MVNO1 and MVNO3 have an overall higher satisfaction than MVNO2 and MVNO4, which is due to the fact that they have less demanding QoS requirements and thus are easier to satisfy.

Taking a closer look at the satisfaction results for MVNO1 and MVNO2 in Fig. 6, we also observe that the DQN algorithm is obviously the best solution. And while both ADMM and PACP have higher satisfaction results for the first 2-3 instances (i.e. 5 – 15 users), their performance sharply declines with the increase of demand, while the DQN remains satisfactory. For MVNO1, ADMM goes below the 0.5 mark with 50 users, and PACP crosses this line with 40 users. For MVNO2, the ADMM and PACP fall below the 0.5 mark with 45 users and 35 users, respectively. On the other hand, the DQN algorithm starts with satisfaction rates of 0.94 and 0.88 with 5 users and ends with solid satisfaction rates of 0.78 and 0.72 for MVNO1 and MVNO2, respectively.

As for MVNO3 and MVNO4, we see in Fig. 6 that the three algorithms are able to maintain satisfactory levels for any number of users. PACP is the one algorithm that performs best because PACP is a popularity-oriented algorithm, and the huge library size and a big number of requesting users for these two slices lead it to prioritize them and subsequently overwhelm the network with their content objects. The DQN still maintains commendable satisfaction levels with a minimum satisfaction of 0.8 for MVNO3 and 0.72 for MVNO4.

Looking at the four MVNOs together, we find that our algorithm is the only one to register good satisfaction across the different slices. It is not the best solution for every individual slice, but it provides the best balance in terms of slice satisfaction for all the slices, which satisfies our objective of serving differentiated services.

Moving on to the slice resource utilization, the first thing we see in Fig. 7 is that the ADMM algorithm always has a resource utilization of 1 under all circumstances and for all slices. This is the case because resource allocation and cache placement are inseparable in the devised ADMM [6]. That is to say, the algorithm immediately and fully places content objects into whatever resource that is sliced. With this algorithm, there is no distinction between the allocated and used resources as the resource slicing and cache placement problems are intertwined. This makes it hard to evaluate the efficiency of the slicing solution of this solution. Looking at the PACP solution, we clearly see that it attempts to use as much resource as possible. This is very typical of the PACP algorithm since its main idea is to flood the network by caching as many copies of the content objects as possible with no consideration for resource utilization. This results in the algorithm quickly using all of its allocated resources as the number of users increases. This is most obvious with MVNO1 and MVNO2 where it starts by using 0.71 and 0.8 of the slices' resources, and the available cache resources are fully utilized at 35 and 25 users, respectively. MVNO3 and MVNO4 witness a similar trend but on a slightly lighter level; it initially uses 0.54 and 0.62 for 10 users, and the allocated resources are fully used with 100 users for MVNO3 and with 95 users for MVNO4.

In contrast, our DQN-based solution demonstrates a more balanced utilization of resources; it gradually increases the utilization as demand requires it. For MVNO1 and MVNO2, it is the one algorithm with the lowest resource utilization and is the last one to reach full utilization. It starts by using 0.54 and 0.58 of the allocated resource with 5 users, and full utilization is realized with 50 users and 45 users, respectively. Tying this back to Fig. 6, this means that the DQN algorithm is able to achieve satisfying service levels for MVNO1 and MVNO2 with the lowest resource utilization rates. For MVNO3 and MVNO4, the DQN algorithm initially uses 0.7 and 0.73 of its allocated resource with 10 users and reaches full utilization of allocated resources with 100 and 90 users, respectively. This gradual increase in resource utilization proves that our DQN-based solution balances the ratio between the allocated to used resources quite well; it matches the increase in demand, while still maintaining low resource utilization.

Looking at the two performance metrics together, we notice that rising resource utilization corresponds to a falling rate of satisfaction. A low resource utilization level means that the allocated resource is enough to comfortably meet the demands of the requesting users, while a high resource utilization rate means that the allocated resource is not sufficient and that the system is struggling to satisfy the demands of the users. This contrasting pattern between slice satisfaction and resource utilization is most obvious with the PACP algorithm serving MVNO2. We can see that for 5 users, the system uses 0.8 of this slice's dedicated resource and registers a staggering 0.9 satisfaction score. As the number of requesting users increases to 20, the utilization steadily grows to 0.96 and the satisfaction gradually drops to 0.74. Starting from the next instance (i.e. 25 users), the PACP algorithm uses all of the resources dedicated to MVNO2. This means that despite the continuous increase in the number of requesting users in the next 5 instances from 25 to 50, this algorithm cannot allocate any more resources to meet this demand, and as a result, the satisfaction plunges dramatically to a measly 0.24 for the 10th instance (i.e. 50 users). The same pattern can be seen with other algorithms and MVNOs, albeit not with the same sharp contrast. For example, in the case of our DQN-based solution serving MVNO2, resource utilization gradually increases from 0.58 with 5 users to full utilization with 45 and 50 users. The slice satisfaction, conversely, drops from 0.88 to 0.72. This proves that our algorithm is able to strike a fine balance between slice satisfaction and slice resource utilization, and it is thanks to the ability of our algorithm to accurately gauge the needs of a slice and allocate resources accordingly.

4) PERFORMANCE ANALYSIS ON THE SYSTEM LEVEL

In this subsection, we investigate the system-level performance of our solution and the introduced benchmarks. First, we develop system-level performance metrics that mirror the ones used in the slice-level subsection. For the satisfaction case, we select the minimum satisfaction achieved by each slice in the slice-level satisfaction experiment. Fortunately, in the slice-level satisfaction experiment, the three benchmarks tested for the same number of users, therefore we continue with this normalized satisfaction value in our system comparison. However, this is not the case with resource utilization. In the slice-level experiment, each algorithm had its considerations for allocating cache resources to the slices, and these allocation values differed quite significantly. In other words, while it is still correct to say that the three algorithms utilized 100% of their allocated resources with 50 users for MVNO1 in Fig. 7, this translates to the following values in terms system's cache resources: [ADMM = 0.1246, PACP = 0.0740, DQN = 0.1576]. Therefore, for the purposes of this experiment, we take maximum resource utilization of the slice-level experiment and multiply that with the actual resource provisioning of each slice for that specific instance. This gives us the maximum used resource, which is considered the system-level equivalent of the slice-level resource utilization. The total used resource by each algorithm for all the slices is equal to the total capacity of the system (=1).

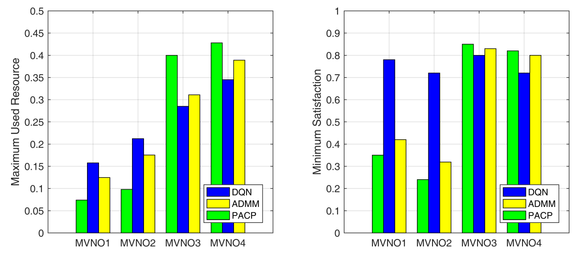

FIGURE 8. System level performance.

Looking at the maximum used resource first, we observe in Fig. 8 that the three algorithms use more resources with MVNO3-4 than with MVNO1-2. This is expected given that MVNO3-4 have more requesting users and content objects. As the PACP algorithm is popularity-oriented, it uses about than 0.82 of the system resources to serve the last two slices. ADMM has a cost objective function to consider, so it follows a more moderate distribution of the system resources, but it still uses 0.7 of the system resources to serve the non-guaranteed rate slices. However, what the PACP and ADMM seem to overlook is the more stringent requirements of MVNO1-2. Unlike MVNO3-4, the first two slices provide guaranteed rate services and are more delay-sensitive. As our DQN-based solution offers MVNO customization, it is able to better understand the varying demands of the slices, and it is the one algorithm that uses the most amount of its resources (about 0.37) attending to the needs of MVNO1-2. As for the satisfaction, we can also see in Fig. 8 that the ADMM and PACP algorithm achieve very unsatisfying results for MVNO1-2. This is a direct consequence of not being able to consider its more demanding requirements accurately. They do however outperform our solution for MVNO1-2 as they obviously overwhelmed the network with their content objects, but they do use more resources for this.

Looking at both the used resource and satisfaction, we can make two observations. First, satisfaction is proportionately related to the system's used resources. This is pretty evident for all the algorithms and all the slices; the algorithm with the highest slice satisfaction level is the one that invests the most resources into the slice. For example, the PACP algorithm uses a maximum of 0.428 to serve MVNO4, and in return, it towers over the rest of the algorithms with a minimum satisfaction of 0.82. Second, in the realistic situation of limited physical capacities and differentiated services, it is imperative that the management framework is able to accurately gauge the requirements of the slices and assign resources accurately. In Fig. 8, the PACP and ADMM algorithms fail to properly address the challenging requirements of MVNO1-2, and this results in very low satisfaction levels. Our solution, in contrast, provides a balance between slice satisfaction and resource utilization for a number of MVNOs offering differentiated services. The satisfaction levels it achieves are not the highest for every slice, but this is a compromise that must be made to guarantee the overall satisfaction of the entire network.

E. PERFORMANCE ISOLATION

In this experiment, we try to verify the performance isolation of our proposed solution. To that end, we initially run the experiment for the user ranges specified in Table. 1 and get the averaged satisfaction and resource utilization results as usual. However, at the 300th episode, we increase the range of requesting users for MVNO2 from 5 – 50 users to 5 – 100 users. As we saw in Fig. 7, the slice's allocated resources are fully utilized starting from the 45th user. As such, the slice's dedicated cache resources are not going to be enough to satisfy this scheduled increase in demand, and the slice will be desperate to use cache resources that belong to other MVNOs. In order for our isolation scheme to succeed, the other three MVNOs must not witness any disturbance to their normal performance.

In Fig. 9, we can see the four MVNOs and their respective satisfaction and resource utilization levels returned from the DQN algorithm from the 200th to the 400th episode. Starting from the 200th episode, MVNO1 achieves an average satisfaction of 0.87 and average resource utilization of 0.79. MVNO3, on the other hand, has a satisfaction rate of 0.85 and resource utilization of 0.82. In line with the previous results in the algorithm comparison subsection, MVNO4 has a lower average satisfaction of 0.79 and higher resource utilization of 0.86 in comparison to MVNO3. Finally, from the 200th to the 300th episode, MVNO2 return an average satisfaction 0.81 and resource utilization of 0.83. After increasing the demand, however, the satisfaction plummets to around

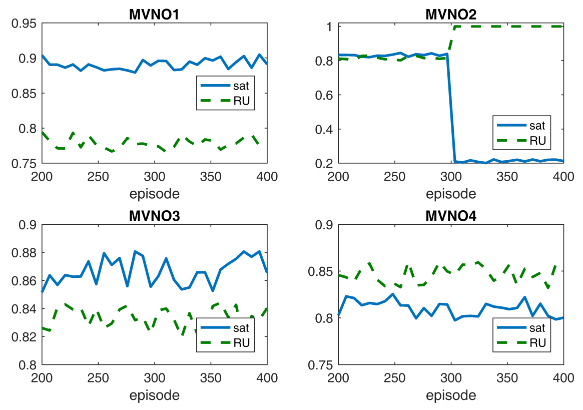

FIGURE 9. Performance isolation.

0.2 and the resource is fully utilized. Despite this disturbance to the performance of MVNO2, the remaining MVNOs seem unaffected by this increase in demand, and they continue to perform normally. As such, it can be concluded that the resources of each slice are isolated and that an increase in the demand for one specific slice does not affect the satisfaction or utilization of other slices.

VI. CONCLUSION

In this work, we proposed a dynamic cache resource management framework that considers cache resource virtualization and content placement at the mobile edge network. This framework incorporated a DQN-based algorithm for slicing the cache resources, and a customized ADMM for placing the content of the different MVNOs. We formulated the reward of the DQN algorithm to maximize a balance between QoS satisfaction and resource utilization and introduced weights to prioritize one over the other if needed. In this work, we chose to give more priority to satisfaction. The objective function of the ADMM solution offered customized content placement for the MVNOs based on their individual QoS requirements, but the general objective function was formulated to always maximize the QoS satisfaction. We conducted extensive simulations to test our framework and the results showed that 1) the cache resource allocation of our system properly corresponded to the increasing demands of the network and consistently allocated sufficient resources, 2) in comparison to other benchmarks, our framework was the only solution to maintain satisfying service levels for ALL the slices, and it did so with the lowest resource utilization rates, and 3) the virtual cache resources of different slices are isolated and a sudden spike in demand for one slice does not disturb the performance of the other slices.

REFERENCES

[1] CISCO, "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021," White Paper c11-481360. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.pdf

[2] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[3] X. Wang, M. Chen, T. Taleb, A. Ksentini, and V. C. M. Leung, "Cache in the air: Exploiting content caching and delivery techniques for 5G systems," *IEEE Commun. Mag.*, vol. 52, no. 2, pp. 131–139, Feb. 2014.

[4] T. K. Forde, I. Macaluso, and L. E. Doyle, "Exclusive sharing & virtualization of the cellular network," in *Proc. IEEE Int. Symp. Dyn. Spectr. Access Netw. (DySPAN)*, Aachen, Germany, May 2011, pp. 337–348.

[5] C. Liang and F. R. Yu, "Wireless network virtualization: A survey, some research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 1, pp. 358–380, 3rd Quart., 2015.

[6] C. Liang, F. R. Yu, H. Yao, and Z. Han, "Virtual resource allocation in information-centric wireless networks with virtualization," *IEEE Trans. Veh. Technol.*, vol. 65, no. 12, pp. 9902–9914, Dec. 2016.

[7] A. Sadeghi, F. Sheikholeslami, and G. B. Giannakis, "Optimal and scalable caching for 5g using reinforcement learning of space-time popularities," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 180–190, Feb. 2018.

[8] A. Aijaz, "Hap—SliceR: A radio resource slicing framework for 5G networks with haptic communications," *IEEE Syst. J.*, vol. 12, no. 3, pp. 2285–2296, Sep. 2018.

[9] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[10] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Atlanta, GA, USA, Nov. 2016, pp. 50–56.

[11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[12] X. Zhang and Q. Zhu, "Information-centric network virtualization for QoS provisioning over software defined wireless networks," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, Baltimore, MD, USA, Nov. 2016, pp. 1028–1033.

[13] Y. Zhou, F. R. Yu, J. Chen, and Y. Kuo, "Resource allocation for information-centric virtualized heterogeneous networks with in-network caching and mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 66, no. 12, pp. 11339–11351, Dec. 2017.

[14] T. D. Tran and L. B. Le, "Joint resource allocation and content caching in virtualized content-centric wireless networks," *IEEE Access*, vol. 6, pp. 11329–11341, Feb. 2018.

[15] K. Thar, N. H. Tran, J. Son, and C. S. Hong, "Resources management in virtualized information centric wireless network," in *Proc. 18th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Kanazawa, Japan, Oct. 2016, pp. 1–4.

[16] Q. Jia, R. Xie, T. Huang, J. Liu, and Y. Liu, "Efficient caching resource allocation for network slicing in 5G core network," *IET Commun.*, vol. 11, no. 18, pp. 2792–2799, Dec. 2017.

[17] Z. Chang, C. Jing, X. Guo, Z. Han, and T. Ristaniemi, "Resource allocation for wireless virtualized hetnet with caching and hybrid energy supply," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Barcelona, Spain, Apr. 2018, pp. 1–6.

[18] Y. He, F. R. Yu, N. Zhao, V. C. M. Leung, and H. Yin, "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach," *IEEE Commun. Mag.*, vol. 55, no. 12, pp. 31–37, Dec. 2017.

[19] Y. He, F. R. Yu, N. Zhao, and H. Yin, "Secure social networks in 5G systems with mobile edge computing, caching, and device-to-device communications," *IEEE Wireless Commun.*, vol. 25, no. 3, pp. 103–109, Jun. 2018.

[20] T. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

[21] M. Mamman, Z. M. Hanapi, A. Abdullah, and A. Muhammed, "Quality of service class identifier (QCI) radio resource allocation algorithm for LTE downlink," *PLoS ONE*, vol. 14, no. 1, Jan. 2019, Art. no. e0210310.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[23] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.

[24] M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús, *Neural Network Design*, vol. 20. Boston, MA, USA: PWS Publishing, 1996.

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, ''Mastering the game of go with deep neural networks and tree search,'' *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[26] J. Eckstein and W. Yao, ''Augmented Lagrangian and alternating direction methods for convex optimization: A tutorial and some illustrative computational results,'' RUTCOR, Rutgers Univ., Camden, NJ, USA, Res. Rep. RRR-32-2012, Dec. 2012.

[27] M. Leinonen, M. Codreanu, and M. Juntti, ''Distributed joint resource and routing optimization in wireless sensor networks via alternating direction method of multipliers,'' *IEEE Trans. Wireless Commun.*, vol. 12, no. 11, pp. 5454–5467, Nov. 2013.

[28] R. Kokku, R. Mahindra, H. Zhang, and S. Rangarajan, ''NVS: A substrate for virtualizing wireless resources in cellular networks,'' *IEEE/ACM Trans. Netw.*, vol. 20, no. 5, pp. 1333–1346, Oct. 2012.

GUOLIN SUN received the B.S., M.S., and Ph.D. degrees in communication and information systems from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2000, 2003, and 2005, respectively.

After his Ph.D. graduation, in 2005, he has got eight years industrial work experience on wireless research and development for LTE, Wi-Fi, the Internet of Things (ZIGBEE and RFID, etc.), cognitive radio, localization, and navigation. Before he joined the School of Computer Science and Engineering, University of Electronic Science and Technology of China, as an Associate Professor, in 2012, he was with Huawei Technologies Sweden. He has filed over 40 patents, and published over 40 scientific conference and journal papers, and acts as a TPC member of conferences. His general research interests include software-defined networks, network function virtualization, and radio resource management.

Dr. Sun currently serves as the Vice-Chair for the 5G oriented cognitive radio SIG of the IEEE (Technical Committee on Cognitive Networks (TCCN)) of the IEEE Communication Society.

HISHAM AL-WARD received the bachelor's degree in electrical engineering (communications and electronics) from Sana'a University, Yemen, in 2011. He is currently pursuing the master's degree with the School of Computer Science and Engineering, University of Electronic Science and Technology of China (UESTC).

He is currently a Graduate Research Assistant with the Mobile Cloud-Net Research Laboratory, UESTC. His current research interests include networking technologies, mobile telecommunications, machine learning, and the IoT.

GORDON OWUSU BOATENG received the bachelor's degree in telecommunications engineering from the Kwame Nkrumah University of Science and Technology, Kumasi-Ghana, West Africa, in 2014. He is currently pursuing the M.Sc. degree in computer science and engineering from the University of Electronic Science and Technology of China (UESTC).

From 2014 to 2016, he worked under sub-contracts for Ericsson, Ghana, and TIGO, Ghana. He is also a member of the Mobile Cloud-Net Research Team, UESTC. His interests include mobile/cloud computing, 5G wireless networks, data mining, D2D communications, and SDN.

GUISONG LIU received the B.S. degree in mechanics from Xi'an Jiao Tong University, Xi'an, China, in 1995, and the M.S. degree in automatics and the Ph.D. degree in computer science from the University of Electronic Science and Technology of China, Chengdu, China, in 2000 and 2007, respectively.

He was a Visiting Scholar with the Humbolt University of Berlin, in 2015. He is currently a Full Professor with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu. He is also the Dean of the School of Computer Science, Zhongshan Institute, UESTC, Zhongshan, China. His research interests include pattern recognition, neural networks, and machine learning.