Video Content Placement at the Network Edge: Centralized and Distributed Algorithms

Yanan Gao, Song Yang, *Senior Member, IEEE,* Fan Li, *Member, IEEE,* Stojan Trajanovski, *Member, IEEE,* Pan Zhou, *Senior Member, IEEE,* Pan Hui, *Fellow, IEEE,* and Xiaoming Fu, *Fellow, IEEE*

Abstract—In the traditional video streaming service provisioning paradigm, viewers typically request video content through a central Content Delivery Network (CDN) server. However, because of the uncertain wide area network delays, the (remote) viewers usually suffer from long video streaming delay, which affects the quality of experience. Multi-Access Edge Computing (MEC) offers a way to shorten the video streaming delay by building small-scale cloud infrastructures at the network edge, which are in close proximity to the viewers. In this paper, we present novel centralized and distributed algorithms for the video content placement problem in MEC. In the proposed centralized video content placement algorithm, we leverage the Lyapunov optimization technique to formulate the video content placement problem as a series of one-time-slot optimization problems and apply an Alternating Direction Method of Multipliers (ADMM)-based method to solve each of them. We further devise a distributed Multi-Agent Reinforcement Learning (MARL)-based method with value decomposition mechanism and parallelization policy update method to solve the video content placement problem. The value Decomposition mechanism deals with the credit assignment among multiple agents, which promotes the cooperative optimization of the global target and reduces the frequency of information exchange. The parallelization of policy network can speed up the convergence process. Simulation results verify the effectiveness and superiority of our proposed centralized and distributed algorithms in terms of performance.

Index Terms—Video Content Placement, Mobile Edge Computing, Online Optimization, Multi-Agent Reinforcement Learning

1 Introduction

With the proliferation of mobile device technology and the rapid development of video content providers (e.g., YouTube or Netflix), mobile video streaming has become one of the most popular applications for mobile devices. In the traditional video streaming paradigm, the viewers typically request video content through a central Content Delivery Networks (CDN) server to their mobile devices regardless of their locations. However, due to the bandwidth limitation and uncertain delay of the wide-area networks, with the tremendous traffic growth, it can easily cause a network bottleneck and hence affect the Quality of Experience (QoE) such as delay, especially for long-distance remote viewers. According to Cisco [1], video traffic will account for around 79 percent of the whole mobile data traffic by 2022, up from 59 percent in 2017. Consequently, the traditional video streaming paradigm cannot deal with the ever-increasing mobile traffic without deteriorating the viewers' QoE, which remains a crucial drawback to solve.

- *Y. Gao, S. Yang and F. Li are with the School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China. E-mail: {yanangao, S.Yang, fli}@bit.edu.cn (Corresponding Author: Song Yang)*
- *S. Trajanovski is with Microsoft, W2 6BD London, United Kingdom. E-mail: sttrajan@microsoft.com*
- *P. Zhou is with the Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, 430074, China. Email: panzhou@hust.edu.cn*
- *P. Hui are with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong, also with the Department of Computer Science, University of Helsinki, 00100 Helsinki, Finland. E-mail: panhui@cse.ust.hk*
- *X. Fu is with Institute of Computer Science, University of Göttingen, 37077 Göttingen, Germany. Email: Fu@cs.uni-goettingen.de*

Multi-Access Edge Computing (MEC) [2] has been proposed to bring the computing resources closer to end users by installing a small resource-limited cloud infrastructure called edge cloud at the network edge. In this context, the requested video content can be cached/placed on the edge cloud in order to obtain a shorter path delay. However, the edge cloud usually consists of a restricted number of servers, and hence exhibits limited storage and processing capability. Due to this reason, it is suggested that the edge clouds are connected with each other via local area network or wired peer-to-peer links so as to work collaboratively [3] and expose a greater storage and processing capability. Nevertheless, it happens that all edge clouds are fully utilized and there are still requests remaining to be served when the number of requests increases. In this sense, the remote CDN server which has sufficient storage capability will be in helping to accommodate these requests. However, the edge clouds and CDN server are connected via long core networks. Therefore the requests which are relayed from edge clouds to a CDN server has to experience long service delay. This deals with how to strategically place each requested video file on edge clouds and CDN server such that the video streaming delay is minimized. We call this problem as the video content placement problem at the network edge in this paper.

Existing approaches to solving the video content placement problem can be classified into two major categories: centralized scheme and distributed scheme. In the centralized scheme, it is usually assumed that the decision-maker can have global knowledge and control of the network information. A unique decision-maker calculates the (optimal) solution based on the collected network status and all the received requests. An application of the centralized approach is the Software-Defined Network (SDN) [4]. The key limitation of the centralized approach is that the assumption of knowing global network information needs frequent information updates and exchanges, which induces much more overhead on time and space. Nevertheless, this scheme can find the optimal solution because of having a piece of overall network information. Considering that the video content system operates in a highly stochastic environment with random demand arrivals, the long-term system performance (e.g., streaming delay) is more relevant than the immediate short-term performance. Moreover, the long-term cost constraint couples the video content placement decisions over time, and yet the decisions have to be made without knowing the future traffic information. To cope with this issue, we devise an online time-slot algorithm that combines the Lyapunov optimization and the Alternating Direction Method of Multipliers with lower calculation complexity. Also, we provide a formulated performance analysis that proves our online method can solve the considered problem in finite steps and find the optimal solution. This can reduce the frequency of data exchange and the times of algorithm iteration for cost-saving.

In the distributed scheme, the individual edge cloud makes the decision based on its own collected requests and its local states, without the need for a centralized decision-maker. In this sense, this scheme can cut computation expenditure and obtain more self-adaptive decisions to optimize response performance, especially in large-scale network scenarios. However, dynamic and non-stationary edge network environment [5], temporal pattern and spatial pattern of video request [6], [7], severely affect the performance of distributed decision-makers. In this paper, we solve the considered problem by using a Multi-Agent Reinforcement Learning (MARL) algorithm to capture the dynamic changes of the network environment and reduce calculation complexity. We devise a MARL-based model with Value Decomposition (VD) credit assignment scheme in a distributed manner. All agents have zero communication of observed information, but still, cooperatively update distributed placement policies through optimizing a global reward target. Also, in order to speed up the learning process, we improve the updating method of policy networks by parallelization and loss estimation. Our key contributions are as follows:

- We develop a centralized online video content placement framework by using the Lyapunov technique to decompose the considered problem into a series of one-time-slot optimization problems, and then apply an Alternating Direction Method of Multipliers (ADMM)-based method to solve each of them. We prove that our proposed centralized method can solve the considered problem with finite steps and bound the maximum long-term time average streaming delay as well as individual edge cloud cost expense.
- We devise a distributed MARL-based algorithm with value decomposition credit assignment and policy networks' parallelization for video content placement problem. All edge clouds are regarded as RL agents that can collaboratively optimize a globally customized reward target.
- We conduct extensive simulations to validate the performance of the proposed centralized and distributed algorithms from the algorithm's hyper-parameter adjustment, the changes in problem scale, etc. It is verified that our solutions have superior abilities to reduce delay and save costs compared to several baselines.

TABLE 1: List of abbreviations.

Abbreviation	Explanation
CDN	Content Delivery Network
MEC	Multi-Access Edge Computing
OVCE	Online Video Content placement on the network Edge
INLP	Integer Nonlinear Programming
OVCP	Online Video Content Placement framework
ADMM	Alternating Direction Method of Multipliers
MARL	Multi-Agent Reinforcement Learning
DMVP	Distributed Multi-Agent-based algorithm with Value Decomposition mechanism and Parallelization of policy network updating
VD	Value-Decomposition mechanism
PPO	Proximal Policy Optimization

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 introduces the network service model, formally defines the video content placement problem at the network edge, and formulates this problem as an exact solution. Section 4 presents a centralized Lyapunov-based online video content placement framework and analyzes its complexity. Section 5 proposes a distributed MARL-based algorithm for the video content placement problem at the network edge. Section 6 describes simulation results and we conclude in Section 7. For the description of any abbreviation, please refer to Table 1.

2 Related Work

2.1 Centralized Video Content and Virtual Network Function Placement in Edge Computing

Offline Video Content Placement and Caching: Ma *et al.* [3] study the cooperative service caching/placement and workload scheduling problem in MEC. They develop an offline algorithm based on Gibbs sampling and water filling to solve service caching/placement problem with offline policy iteration. Qu *et al.* [8] generate a caching scheme by maximizing the QoE value over all the viewers under the condition that the QoE function has a general or linear function with received bitrate. They present approximation algorithms to solve the multiple-choice knapsack problem of multiple bitrate video content placement in these two cases, respectively. Tran *et al.* [9] address an adaptive bitrate video problem in polynomial time, where the edge clouds can collaboratively optimize the caching policy with a total objective. Bilal *et al.* [10] manage the addition and removal of videos, and video catalog update by collaborative joint caching, which aims to minimize replication within the edge cloud cluster to save place. Nevertheless, the above literature only works in the offline scenario, and cannot properly solve the online video content placement problem where the future requests are unknown.

Online Video Content Placement and Caching: Wang *et al.* [11] study the dynamic configuration adaptation and bandwidth allocation problem in edge-based video analytics systems for multiple video streams, where the energy consumption of mobile devices and service latency are taken into account. However, only one edge server (cloud) is considered in [11]. Wang *et al.* [7] present a multi-agent actor-critic method to adaptively make caching decisions after they fix the size of the state space through clustering the video requests. Wang *et al.* [12] present a competitive online schedule algorithm but with only two anticipatory caching servers to serve a sequence of requests with minimum costs. Li *et al.* [13] present an online content placement, node association, and power allocation strategy based on Lyapunov optimization and Generalized Benders decomposition. Similarly, Xia *et al.* [14] apply Lyapunov technology to develop an online data caching framework in edge computing. However, the video streaming delay and multiple bitrate levels of video files are not thoroughly considered in the above literature, as we do in this paper.

VNF Placement Problem: Virtual Network Function (VNF) (or service chain) placement problem has been extensively studied recently, and this problem refers to placing a set of functions on network nodes obeying function ordering constraint. The VNF placement problem, therefore, shares some similarities with our addressed problem in some sense. For example, Ma *et al.* [15] propose an exact polynomial-time algorithm to solve the VNF placement problem for the non-ordered VNF dependence case, and prove that the VNF placement problem for the totally and partially-ordered dependence case is NP-hard. A dynamic programming and an efficient heuristic are proposed to solve the VNF placement problem under these two cases, respectively. You and Li [16] model the VNF placement constraint via a bipartite graph and present a load-balanced max-min heuristic to solve the VNF placement problem. Hawilo *et al.* [17] study the VNF placement problem by modeling it as a mixed integer linear programming optimization and devise a heuristic to minimize the intra-communication delay among VNFs in service function chains. Khoshkholghi *et al.* [18] address the the VNF placement problem to jointly optimize the cost and delay. They propose two efficient heuristics using genetic algorithm and bee colony algorithm to solve this problem. Please refer to [19] for a survey about the resource allocation problems in NFV. We can see that the difficulty in the VNF placement problem mainly lies in how to place a set of ordered VNFs on network nodes. However, our addressed video caching problem additionally considers how to use existing cached video files to provision requests (with transcoding if possible) apart from solely placing video content.

2.2 Distributed Video Content Placement in Edge Computing

Poularakis *et al.* [20] reduce the content delivery delay by devising a pseudo polynomial-time optimal solution with the multiple-choice knapsack problem and designing a caching algorithm for multiple network operators that cooperatively calculate caching policy. But [20] ignores the delay parameter to obtain better results. [21] proposes a distributed online approach that uses the stochastic gradient descent method to jointly optimize content placement and delivery without historical knowledge. But [21] cannot adapt to the dynamic changes of the network environment. Some works [22], [23] devise game theory-based algorithms in order to achieve the Nash Equilibrium among "players" (edge clouds for video storage). However, with the increase of individuals, it is hard to guarantee that the optimization process can reach the Nash Equilibrium.

Moreover, based on Markov Decision Process (MDP) with time sequence dependence, reinforcement learning algorithm [24], [25], [26], [27] can dynamically learn placement/caching policy by interacting with the network environment in a distributed manner. Luong *et al.* [24] list a series of DQN-based methods used to provide distributed placement/caching policy through selecting a channel, base station, bitrate, and optimizing throughput, transmitted rate, video quality. But these methods cannot deal with a large-scale network due to DQN method's poor convergence performance. Zhong *et al.* [25] introduce a multi-agent RL method with centralized critic and decentralized actor to seek the optimal cooperative caching policy, which additionally considers the requests from the overlapped areas. But the centralized critic network has to take all observations from each agent which causes a large amount of information transmission. Yeh *et al.* [26] propose an online MDP-based algorithm that finds the optimal policy by aggregating the similar states to reduce the state space. The state compression and approximate MDP enlarges the calculation complexity. Tian *et al.* [27] integrate double DQN and dueling DQN to simultaneously interact with caching environment, aiming to individually learn caching policy for each edge device. Similar to [24], this work still has a long learning process due to DQN's rough estimation of the state value. The features of hard convergence, the large-scale state and action space, as well as low cooperation efficiency severely weaken the above-mentioned algorithms' performance. To overcome these weaknesses, we use centralized credit assignment to implement cooperation among all distributed agents and omit the large-scale transmission of the state information, and perform the policy network updating in parallel to speed up the learning process in this paper.

3 Network Model and Problem Definition

We assume there is a set of edge areas $\mathcal{N}_e$. In each edge area $n \in \mathcal{N}_e$, there is a base station for sending and receiving signals. An edge cloud consisting of a limited number of servers is associated with this base station for processing and caching video contents requested by the viewers. We use $\eta(n)$ to represent the available capacity of the edge cloud in edge area n. $\lambda(n)$ is usually less than the full capacity of n to guarantee that the server can work normally and steadily so that the tasks performed on it will not affect each other. Moreover, there is also a set of intermediate nodes (e.g., router nodes) $\mathcal{N}_i$ and one CDN server $\mathcal{N}_c$ for simplicity[1]. Different edge areas, as well as the CDN server, are interconnected with each other by one or more links, and we use $c(l)$ to represent the capacity of link l. We use $P^{u,v}$ to denote

1. Our work can also be extended to the scenario of multiple geo-distributed CDN servers.

the path set with known K paths[2] between u and v, where $u, v \in \mathcal{N}_e \cup \mathcal{N}_c$. As a result, the network can be represented by $\mathcal{G}(\mathcal{N}, \mathcal{L})$, where $\mathcal{N} = \mathcal{N}_e \cup \mathcal{N}_c \cup \mathcal{N}_i$ denotes the set of N nodes and $\mathcal{L}$ represents the set of L links. In particular, the edge area and the (remote) CDN server are inter-connected via relatively long dedicated backhaul connectivity compared to the connection between edge areas. We assume that the network lifetime consists of 1, 2, ..., T time slots and the span of a time slot can be set manually to be longer than video streaming delay in order to avoid frequent update configurations and reduce overall costs [11]. For instance, in the Software-Defined Wide Area Networks (SD-WAN) of Microsoft [29] or Google [30], the period (time slot span) for each time's update configuration is around 5 minutes while the service delay is in the magnitude of seconds. In each time slot, the viewer sends his/her video content request $r(f, b, \alpha, u)$ to his/her local edge cloud u, where f denotes the requested file content with bitrate level b, and α is the requested transmission rate. In this paper, we use place and cache interchangeably. Moreover, we assume that each viewer can receive the video content from the local edge area via a unique available channel. We, therefore, do not take the wireless delay from viewers to the local edge area into account for brevity in this paper. The reason is that this part of value only depends on the channel bandwidth and file content size [31] in this context and hence cannot be further optimized/minimized.

3.1 Video Streaming Delay Calculation Model

In general, the video streaming delay of viewer at edge area u which requires f with bitrate level b can be calculated as:

$$T(u, v) + \Gamma_v(f, b, \beta) \tag{1}$$

where $T(u, v)$ denotes the path delay from u to v where f is placed. Since a video content may traverse multiple intermediate nodes and links, so We use $T(u, v)$ to approximately represent the sum of transmitting delay and propagation delay in this paper. Clearly, if f is placed on u (in this case $u = v$), then the path delay is 0. $\Gamma_v(f, b, \beta)$ indicates the transcoding delay for f from current placed bitrate level β to the requested bitrate level b. Video transcoding is to encode video content into multiple representations, and we consider that a lower bitrate variant can be obtained from a higher bitrate variant via transcoding [9]. $\Gamma_v(f, b, \beta)$ is a nonlinear function with variables f, b and β according to [32].

For example in Fig. 1, assuming there are 3 edge areas and 1 CDN server, which are interconnected with each other and the path delays are labelled on these links. Suppose there are 4 requested video content files, and we let each edge cloud store one video file with the bitrate level 1080p and the CDN server contains all the video contents with all the bitrate levels. For simplicity, the transcoding delay from 1080p to 720p for all the files is 50 *ms*. As shown in Fig. 1, within the coverage of each edge area, the viewers request video content with different bitrate levels. For example, since viewer 1 requires f_1 with 720p bitrate level and f_1 is placed in E_2 with 1080p bitrate level, the streaming delay is equal to: 8 (path delay) + 50 (transconding delay) = 68 *ms*. Even though f_1 is also stored in CDN server, since it consumes much longer path delay, it is preferable to accommodate a request by using the edge cache. On the contrary, since the required f_4 is not placed on any of the edge clouds, viewer 4 has to fetch the video content from remote CDN server which causes a streaming delay of $15 + 80 = 95$ *ms* by taking the (shortest) path E_3-E_1-C.

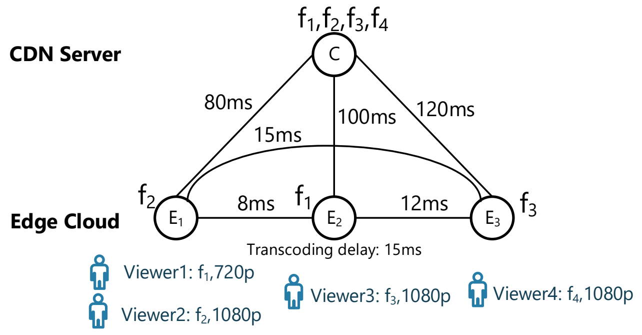


Fig. 1: An example of video streaming delay calculation.

3.2 Problem Definition and Formulation

In this subsection, we formally define the (centralized) Online Video Content placement at the network Edge (OVCE) problem as follows:

Definition 1. *Given is a network $\mathcal{G}(\mathcal{N}, \mathcal{L})$ and for each time slot $t \in T$, there is a set of video requests R_t. For each $r(f, b, \alpha, u) \in R_t$, the OVCE problem is to place f on v ($v \in \mathcal{N}_e \cup \mathcal{N}_c$) and find the associated path[3] from u to v to minimize the total time average video streaming delay such that time average edge cloud cost consumption is no greater than a specified value.*

The OVCE problem is NP-hard in general. To prove it, for simplicity we assume there is only one time slot and we do not consider the streaming delay. In this sense, the OVCE problem can be reduced to the NP-hard bin-packing problem [33], which is to pack a set of items with proper sizes into minimized number of given bins without violating the bin's capacity. Subsequently, we present an exact solution to formulate the OVCE problem. For ease of reading, the notations used in this paper are summarized in Table 2.

2. According to [28], at most 6 paths in GÉANT network are enough for serving 11460 traffic matrices during the entire 4-month duration without violating Quality of Service (QoS). We, therefore, assume that a (small) set of paths is sufficient for calculating the optimal solution. This set of paths is precalculated and given in the problem.

3. It is worthwhile to mention that the distributed video content placement scheme usually does not consider the routing issue, since there is no global view of overall the network information. We, therefore, do not consider the routing issue in the distributed video content placement problem in this paper. Consequently in the distributed video content placement problem, one link (u, v) will be drawn with delay value $T(u, v)$ for two edge cloud nodes $u, v \in \mathcal{N}_e$, if they are the neighbor (nearby) edge areas.

TABLE 2: Notations.

Notation	Description
$\mathcal{G}(\mathcal{N},\mathcal{L})$	A network with set of nodes and links $\mathcal{N}$ and $\mathcal{L}$. $\mathcal{N}=\mathcal{N}_e\cup\mathcal{N}_c\cup\mathcal{N}_i$, where $\mathcal{N}_e$ denotes the set of edge cloud, $\mathcal{N}_c$ represents the CDN server and $\mathcal{N}_i$ indicates the intermediate nodes (e.g., router nodes)
$\eta(n)$, ω_n, π_n	The capacity of edge node n, the per-unit cost of transcoding on n, the per-unit cost of placing on n
$c(l)$, $T(p_k)$, B	The available capacity of link l, the delay of path p_k, the set of bitrate levels
$D(t)$, $E_n(t)$, C_n	Total delay of all the requests at time t, the cost consumption of edge cloud n at time t, time average cost constraint for node n
$\Gamma_v(f,b,\beta)$	Transcoding delay for f from current cached bitrate level β to the requested bitrate level b
S_β^f	The size of file f of the bitrate level β
R_t	The set of requests in time t. For each request $r(f,b,\alpha,u)\in R_t$, f denotes the requested file content with bitrate level b, α is the requested transmission rate, and u means its serving local base station
$H_{l,k}^{u,v}$	A given boolean array indicating whether link l is traversed by path p_k between u and v
$X_{n,t}^{r,\beta}$	A boolean variable. It is 1 (true) if r is accommodated by the requested cached file with bitrate level β on node n at time t, and 0 (false) otherwise
$Y_{k,t}^{r,u,v}$	A boolean variable. It is 1 (true) if r's requested file is cached on v and $p_k\in P^{u,v}$ is selected at time t, and 0 (false) otherwise, where u is the local edge cloud for r

Objective:

$$\min_{\forall \mathbf{X},\mathbf{Y}} \lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\sum_{r\in R_t}\Bigg(\sum_{v\in\mathcal{N}_e,k\in K} Y_{k,t}^{r,u,v}\cdot T(p_k)+ + \sum_{n\in\mathcal{N}_e,\beta\in B} X_{n,t}^{r,\beta}\cdot\Gamma_n(f,b,\beta)\Bigg) \quad (2)$$

Placement Constraints:

$$\sum_{n\in\mathcal{N}}\sum_{\beta\in B} X_{n,t}^{r,\beta}=1 \ \ \forall t\in T, r\in R_t \quad (3)$$

$$\sum_{v\in\mathcal{N}}\sum_{p_k\in P^{u,v}} Y_{k,t}^{r,u,v}=1 \ \ \forall t\in T, r\in R_t \quad (4)$$

Path Selection Constraint:

$$\sum_{p_k\in P^{u,v}} Y_{k,t}^{r,u,v}=\sum_{\beta\in B} X_{v,t}^{r,\beta} \ \ \forall t\in T, r\in R_t, v\in\mathcal{N}_e \quad (5)$$

Edge Cloud Capacity Constraint:

$$\sum_{\beta\in B,f'\in F:f'==f}\max_{r\in R_t} X_{n,t}^{r,\beta}\cdot S_\beta^{f'}\le\eta(n) \ \ \forall t\in T, n\in\mathcal{N}_e \quad (6)$$

Link Capacity Constraint:

$$\sum_{r(f,b,\alpha,u)\in R_t}\sum_{v\in\mathcal{N}}\sum_{p_k\in P^{u,v}} Y_{k,t}^{r,u,v}\cdot H_{l,k}^{u,v}\cdot\alpha\le c(l) \ \ \forall t\in T, l\in\mathcal{L} \quad (7)$$

Edge Cloud Cost Consumption Constraint:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=1}^{T}\Bigg(\sum_{r\in R_t}\sum_{\beta\in B} X_{n,t}^{r,\beta}\cdot(S_\beta^f-S_b^f)\omega_n+ \sum_{f\in F,\beta\in B}\max_{r(f,b,\alpha,u)\in R_t}\left(X_{n,t}^{r,\beta}\cdot S_\beta^f\cdot\pi_n\right)\Bigg)\le C_n \ \ \forall n\in\mathcal{N}_e \quad (8)$$

Eq. (2) minimizes the long-term total video streaming delay over all the requests. More specifically, for each time slot t and each request $r\in R_t$, $\sum_{v\in\mathcal{N}_e,k\in K} Y_{k,t}^{r,u,v}\cdot T(p_k)$ calculates the path delay, and $\sum_{n\in\mathcal{N}_e,\beta\in B} X_{n,t}^{r,\beta}\cdot\Gamma_n(f,b,\beta)$ calculates the transcoding delay. As a result, Eq. (2) returns the time average total delay over all the time slots. Eqs. (3)-(4) ensure that each request should be accommodated by placing a video file with one bitrate level on one node. The above two placement constraints are set for variables $X_{n,t}^{r,\beta}$ and $Y_{k,t}^{r,u,v}$, respectively, and Eq. (5) establishes the equality relation between $X_{n,t}^{r,\beta}$ and $Y_{k,t}^{r,u,v}$. More specifically, Eq. (5) indicates that when r places f on v, only one path between u and v can be selected to use. In case $u=v$ which means that the video content is placed on its local edge cloud, the dummy path $u-u$ with traversing delay of 0 and consumed bandwidth of 0 will be used. Eq. (6) ensures that the capacity of each edge cloud cannot be exceeded. In particular, there may exist the case when two or more requests access the same video file with the same bitrate level on n, but n only needs to store one copy of the required video file. That is why a max function is taken for all $X_{n,t}^{r,\beta}$ in Eq. (6). Eq. (7) ensures that the capacity of each link cannot be exceeded. In order to avoid congestion, $c(l)$ can be set to be less the link's full capacity. Eq. (8) is the long-term cost constraint for each edge cloud node, which ensures that the time average cost consumption for edge cloud n should not be greater than a given value C_n. More specifically, for each time slot t and each request $r(f,b,\alpha,u)\in R_t$, $X_{n,t}^{r,\beta}\cdot(S_\beta^f-S_b^f)\omega_n$ calculates the transcoding cost[4], and $\max_{r\in R_t}\left(X_{n,t}^{r,\beta}\cdot S_\beta^f\cdot\pi_n\right)$ calculates the cost for placing f with bitrate level β on n. It is worthwhile to mention that when a decision is made to place a video file on a node, the video file will be migrated from the local edge cloud or the CND server by following a path. In this paper, we assume to use a greedy approach by copying a video file from the nearest node and using the shortest path. Accordingly, the placing cost here can implicitly represent the sum of maintaining cost for placing the file, the migration cost, and the re-routing cost.

Eqs. (2)-(8) is an Integer Nonlinear Programming (INLP) formulation, because $\Gamma_n(f,b,\beta)$ is a nonlinear function and $X_{n,t}^{r,\beta}$ and $Y_{k,t}^{r,u,v}$ are boolean variables. INLP has exponential computational complexity, which cannot be used in practice. Moreover, Eqs. (2)-(8) require the complete offline knowledge of R_t over all time slots, which is difficult to obtain in practice. In the following, we will present a centralized online video content placement framework without requiring traffic requests over all time slots as well as a distributed

4. In this paper, we assume that the size of a video file is proportional with the bitrate level and adopt the transcoding cost model similar to [34].

video content placement algorithm.

4 A Centralized Online Video Content Placement Framework

In this section, we present a centralized Online Video Content Placement framework (OVCP) and then prove it has a performance guarantee compared to an offline optimal solution. The proposed framework leverages on the Lyapunov optimization technique [35], and it can transform the original long-term optimization problem in Eqs. (2)-(8) to short-term (per time-slot) optimization problem, which only requires the current time-slot information. We then apply an ADMM method to solve each short-term optimization problem.

4.1 Lyapunov-based Online Video Content Placement Framework

Eq. (8) is a long-term edge cloud cost constraint, which is also a major challenge to directly solve the optimization problem in Eqs. (2)-(8). To tackle it, we resort to the Lyapunov technique and transform Eq. (8) into a queue stability problem. More specifically, we introduce a virtual queue $Q_n(t)$ for each edge cloud n and initially assume that $Q_n(0) = 0$, $\forall n \in \mathcal{N}_e$. Denote $E_n(t) = \left(\sum_{r\in R_t}\sum_{\beta\in B} X_{n,t}^{r,\beta}\cdot(S_\beta^f - S_b^f)\omega_n + \sum_{f\in F,\beta\in B}\max_{r(f,b,\alpha)\in R_t}\left(X_{n,t}^{r,\beta}\cdot S_\beta^f\cdot\pi_n\right)\right)$ as the total cost consumption of edge cloud at time t. In time slot 1,2, $\dots$, T, we have following queueing dynamic equation for each edge cloud n's virtual queue:

$$Q_n(t+1) = \max[Q_n(t) + E_n(t) - C_n, 0] \qquad (9)$$

where $Q_n(t)$ denotes the queue backlog of edge cloud n in time slot t representing the deviation of the current edge cloud n's cost consumption. In particular, we regard $E_n(t)$ as the "arrival" of virtual queue $Q_n(t)$ and C_n as the "departure" of the virtual queue.

Next, let $\mathbf{Q}(t) = \{Q_n(t)\}$ denote the vector of all the virtual queues $\forall n \in \mathcal{N}_e$, we define the following Lyapunov function:

$$L(\mathbf{Q}(t)) = \frac{1}{2}\sum_{n\in\mathcal{N}_e} Q_n^2(t) \qquad (10)$$

According to [35], the queue backlog can be understood as an amount of work that needs to be done. In this sense, $L(\mathbf{Q}(t))$ can reflect the "congestion level" of all the virtual queues. A small value of $L(\mathbf{Q}(t))$ implies that the backlogs of all the queues are small, which means that all the virtual queues have strong stability.

In order to keep the virtual queues stable, that is, to enforce edge cloud cost constraint by persistently pushing the Lyapunov function towards a lower congestion level, we introduce $\Delta(\mathbf{Q}(t))$, which is a one-slot Lyapunov drift:

$$\Delta(\mathbf{Q}(t)) = \mathbb{E}[L(\mathbf{Q}(t+1)) - L(\mathbf{Q}(t))|\mathbf{Q}(t)] \qquad (11)$$

It follows that:

$$\begin{aligned}
&\Delta(\mathbf{Q}(t)) = \mathbb{E}[L(\mathbf{Q}(t+1)) - L(\mathbf{Q}(t))|\mathbf{Q}(t)] \qquad (12)\\
&= \sum_{n\in\mathcal{N}_e}\mathbb{E}[L(\max[Q_n(t)+E_n(t)-C_n,0]) - L(Q_n(t))|\mathbf{Q}(t)]\\
&= \frac{1}{2}\sum_{n\in\mathcal{N}_e}\mathbb{E}[\max(Q_n(t)+E_n(t)-C_n,0)^2 - Q_n^2(t)|\mathbf{Q}(t)]\\
&\overset{\diamond}{\leq} \frac{1}{2}\sum_{n\in\mathcal{N}_e}\mathbb{E}[E_n^2(t) + C_n^2 + 2Q_n(t)(E_n(t)-C_n)|\mathbf{Q}(t)]\\
&= \frac{1}{2}\sum_{n\in\mathcal{N}_e}\mathbb{E}[C_n^2 + E_n^2(t)|\mathbf{Q}(t)] + \sum_{n\in\mathcal{N}_e}\mathbb{E}[Q_n(t)(E_n(t)-C_n)|\mathbf{Q}(t)]\\
&\leq W + \sum_{n\in\mathcal{N}_e}\mathbb{E}[Q_n(t)(E_n(t)-C_n)|\mathbf{Q}(t)]
\end{aligned}$$

where $W = \frac{1}{2}\sum_{n\in\mathcal{N}_e}\mathbb{E}[C_n^2 + \max(E_n^2(t))]$ and inequality $\diamond$ comes from the fact that $[\max(a+b-c,0)]^2 \leq a^2+b^2+c^2+2a(b-c)$. According to [35], if a policy greedily minimizes the Lyapunov drift $\Delta(t)$ in each time slot, then all backlogs are consistently pushed towards a low level, which potentially maintains the stability of all queues. Therefore, we jointly consider the queue stability and objective function by minimizing both drift function and the objective function as in Eq. (13), which is called *drift-plus-penalty*.

$$\begin{aligned}
&\Delta(\mathbf{Q}(t)) + V\cdot\mathbb{E}[\sum_{r\in R_t}(\sum_{v,k}Y_{k,t}^{r,u,v}\cdot T(p_k)\\
&\quad + \sum_{n,\beta}X_{n,t}^{r,\beta}\cdot\Gamma_n(f,b,\beta))] \qquad (13)
\end{aligned}$$

For brevity, we denote $D(t) = \sum_{r\in R_t}(\sum_{v,k}Y_{k,t}^{r,u,v}\cdot T(p_k) + \sum_{n,\beta}X_{n,t}^{r,\beta}\cdot\Gamma_n(f,b,\beta)$ as the sum of delay for all the requests at time t. Together with Eq. (12), Eq. (13) yields:

$$\begin{aligned}
&\Delta(\mathbf{Q}(t)) + V\cdot\mathbb{E}[D(t)] \qquad (14)\\
\leq& W + \sum_{n\in\mathcal{N}_e}\mathbb{E}[Q_n(t)(E_n(t)-C_n)|\mathbf{Q}(t)] + V\cdot\mathbb{E}[D(t)]
\end{aligned}$$

where $V \geq 0$ is a control parameter to choose a tradeoff between the optimality and queue backlog. Consequently, we propose an Online Video Content Placement framework (OVCP) in Algorithm 1 to solve the OVCE problem. More specifically, in each time slot t, OVCP obtains values of $Q_n(t)$ and $E_n(t)$ and video requests R_t. After that, OVCP solves (**P1**) in Eq. (15) by obtaining video content placement and routing strategies. Accordingly, the queue backlog will be updated in the next time slot.

$$(\mathbf{P1}) \qquad \min \sum_{n\in\mathcal{N}_e} Q_n(t)E_n(t) + V\cdot D(t) \qquad (15)$$
$$\text{subject to } (3)-(7)$$

However, we notice that directly solving (**P1**) consumes exponential computational time since it is an INLP, which is intractable in practice. Next, we will present an ADMM-based approach that can converge quickly to solve (**P1**).

4.2 An ADMM Approach to solve P1

In this subsection, we present an alternating direction method of multipliers (ADMM)-based approach to solve **P1**.

Algorithm 1: OVCP

```
Input: $\mathcal{G}(\mathcal{N},\mathcal{L}), R$
Output: Video content placement and routing paths
        decisions over different time slots
1 for $t \leftarrow 0$ to $T$ do
2     Observe $Q_n(t)$ and $E_n(t)$
3     Receive the traffic requests $R_t$
4     Solve (P1)
5     Update $Q_n(t+1) = \max[Q_n(t) + E_n(t) - C_n, 0]$
```

ADMM [36] works in the procedure of a decomposition-coordination, where the solutions to small local subproblems are coordinated to find a solution to a large global problem. ADMM works in the following general form:

$$\begin{aligned} \min \quad & f(x) + g(y) \\ \text{subject to:} \quad & ax + by = c \end{aligned} \tag{16}$$

where x and y are variables with the equality constraint, and $f(x)$ and $g(y)$ are convex functions. The augmented Lagrangian function in ADMM is expressed as:

$$\mathcal{I}(x,y,\mu) = f(x) + g(y) + \mu^\top(ax+by-c) + \frac{\rho}{2}\|ax+by-c\|^2 \tag{17}$$

where μ is the Lagrangian multiplier and ρ is the penalty parameter. As a result, ADMM works iteratively as follows in a sequential manner until convergence or a stopping criterion is satisfied:

$$\begin{aligned} x^{k+1} &:= \arg\min_x \mathcal{I}(x, y^{k+1}, \mu^k) \\ y^{k+1} &:= \arg\min_x \mathcal{I}(x^{k+1}, y, \mu^k) \\ \mu^{k+1} &:= \mu^k + \rho \cdot (ax^{k+1} + by^{k+1} - c) \end{aligned} \tag{18}$$

In our situation, we aim to apply ADMM to solve (**P1**). To that end, we first expand the objective in Eq. (15) as the following:

$$\begin{aligned} &\sum_{n\in\mathcal{N}_e} Q_n(t)\left(\sum_{r\in R_t}\sum_{\beta\in B} X_{n,t}^{r,\beta}\cdot(S_\beta^f - S_b^f)\omega_n + \right. \\ &\left.\sum_{f\in F,\beta\in B}\max_{r(f,b,\alpha)\in R_t}\left(X_{n,t}^{r,\beta}\cdot S_\beta^f\cdot\pi_n\right)\right) + \\ &V\left(\sum_{r\in R_t}(Y_{k,t}^{r,v}\cdot T(p_k) + X_{n,t}^{r,\beta}\cdot\Gamma_n(f,b,\beta))\right) \end{aligned} \tag{19}$$

For ease of notation, in the following we use $f(X)$ and $g(Y)$ to denote the terms containing X and Y in Eq. (19), respectively.

$$\begin{aligned} f(X) = &\sum_{n\in\mathcal{N}_e} Q_n(t)\left(\sum_{r\in R_t}\sum_{\beta\in B} X_{n,t}^{r,\beta}\cdot(S_\beta^f - S_b^f)\omega_n + \right. \\ &\left.\sum_{f\in F,\beta\in B}\max_{r(f,b,\alpha)\in R_t}(X_{n,t}^{r,\beta}\cdot S_\beta^f\cdot\pi_n)\right) + V\cdot X_{n,t}^{r,\beta}\cdot\Gamma_n(f,b,\beta) \end{aligned} \tag{20}$$

$$g(Y) = V\cdot(\sum_{r\in R_t} Y_{k,t}^{r,v}\cdot T(p_k)) \tag{21}$$

Algorithm 2: ADMM-based Algorithm to solve (**P1**)

```
Input: $\mathcal{G}(\mathcal{N},\mathcal{L}), R$
Output: Video content placement and routing paths
        decisions at time $t$
1 Repeat until convergence:
2   $X^{k+1} := \arg\min_X \mathcal{I}_\rho(X, Y^{k+1}, \boldsymbol{\mu}^k)$
3   $Y^{k+1} := \arg\min_X \mathcal{I}_\rho(X^{k+1}, Y, \boldsymbol{\mu}^k)$
4   $\mu_i^{k+1} := \mu_i^k + \rho \cdot H_i^{k+1}$, where $1 \le i \le 5$
```

Moreover, we notice that Eqs. (6) and (7) are inequality constraints, and we need to transform them into equality constraints in order to fit the required form of ADMM. Without loss of generality, for a constraint $h(x) < 0$, we can equivalently transform it into context constraint $\max\{0, h(x)\}^2 = 0$ according to [37]. In this way, Eqs. (6) and (7) can be transformed into equality constraints. Subsequently, Eq. (15) can be transformed into the following optimization formulation:

$$\begin{aligned} &\min \quad f(X) + g(Y) \\ &s.t. \begin{cases} H_1(X) := \sum\limits_{n\in\mathcal{N}}\sum\limits_{\beta\in B} X_{n,t}^{r,\beta} - 1 = 0 \\ H_2(Y) := \sum\limits_{v\in\mathcal{N}}\sum\limits_{p_k\in P^{u,v}} Y_{k,t}^{r,u,v} - 1 = 0 \\ H_3(X,Y) := \sum\limits_{p_k\in P^{u,v}} Y_{k,t}^{r,u,v} - \sum\limits_{\beta\in B} X_{v,t}^{r,\beta} = 0 \\ H_4(X) := \max\{0, \sum\limits_{\beta,f'}\max\limits_{r\in R_t} X_{n,t}^{r,\beta}\cdot S_\beta^{f'} - \eta(n)\}^2 = 0 \\ H_5(Y) := \max\{0, \sum\limits_{r(f,b,\alpha)\in R_t}\sum\limits_{v\in\mathcal{N}}\sum\limits_{p_k\in P^{u,v}} Y_{k,t}^{r,v} H_{l,k}^{u,v}\alpha \\ -c(l)\}^2 = 0 \end{cases} \end{aligned} \tag{22}$$

Accordingly, in Eq. (23) we define the augmented Lagrangian function and we devise Algorithm 2 to solve (**P2**).

$$\mathcal{I}_\rho(x,y,\boldsymbol{\mu}) = f(x) + g(y) + \sum_{i=1}^{5}\mu_i^\top H_i(\bullet) + \sum_{i=1}^{5}\frac{\rho}{2}\|H_i(\bullet)\|^2 \tag{23}$$

Theorem 1. *Algorithm 2 can converge in finite steps and return the optimal solution.*

Proof. The proof follows similarly to Theorem 1 in [37] and we have omitted it due to space limitation. ☐

4.3 Performance Analysis

Theorem 2. *The proposed OVCP in Algorithm 1 employing Algorithm 2 can achieve the following performance guarantee:*

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T}\mathbb{E}\{D(t)\} \le P^* + \frac{W}{V} \tag{24}$$

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T}\sum_{n\in\mathcal{N}_e}\mathbb{E}\{Q_n(t)\} \le \frac{W + VP^*}{\epsilon} \tag{25}$$

*where P^* represents the optimal solution for problem (**P1**), $\epsilon > 0$ is a constant denoting the distance between the time average cost consumption and cost budget, and $W = \frac{1}{2}\sum_{n\in\mathcal{N}_e}\mathbb{E}[C_n^2 + \max(E_n^2(t))]$ is a constant parameter defined in Eq. (12).*

Proof. The proof follows similarly from [35], and we omit its details due to the space limitation. □

Theorem 2 reveals that OVCP can achieve a $[O(1/V), O(V)]$ tradeoff between delay and queue backlog. By setting an arbitrarily large V for OVCP, it can achieve the optimal performance in terms of time average streaming delay. Such performance is achieved at the expense of increasing queue backlogs as shown in Eq. (25), which has a linear relationship with V. Another observation is that [38], if we set the constraint C_n too tight, it is not possible to place all the requested video files. In this sense, OVCP will have to place a subset of R on the remote CDN server.

5 A Distributed Video Content Placement Framework

In this section, we devise a **D**istributed **M**ulti-Agent-based algorithm with **V**alue Decomposition mechanism and **P**arallelization of policy network updating (DMVP) for video content placement problem. We start with the preliminary introduction of Multi-Agent Reinforcement Learning (MARL). Then, we demonstrate the DMVP's design for the distributed video content placement framework. After that, we detail the DMVP algorithm with the VD credit assignment scheme and improved parallelization on policy network updating. Finally, we analyze the convergence of DMVP.

5.1 Preliminary of MARL

Markov Decision Process (MDP), a premise of RL, refers to a process where the current state only depends on the last-time-slot state. An RL agent takes its action to interact with the environment. The environment uses transition probability function $p(s'|s, u)$ to complete the state transition and return the next-time-slot state s'. Also, it uses the global reward function to calculate the return reward r. It aims to learn the policy that maximizes the accumulative discount reward

$$Rc_t = \sum_{l=1}^{\infty} \gamma^l r_{t+l} \tag{26}$$

where γ is the discount factor.

There exist multiple RL agents that can simultaneously interact with the network environment in MARL, which breaks up the MDP premise for each agent. The most important issue of MARL is the non-stationary state transition caused by the state's disordered changes due to the simultaneous execution of multiple actions. Hence, the MARL model takes a joint action U composing of each agent's action u to avoid the non-stationary state changes of each agent, which can make the whole MARL model satisfy the MDP premise. At the same time, the environment returns a global reward r based on joint action U and the current state s. We adopt a cooperative MARL pattern among all agents to maximize the globally accumulative reward Rc. Also, we assume that each agent only has an ability of local observation rather than global knowledge in order to reduce the frequently large-size information transmission.

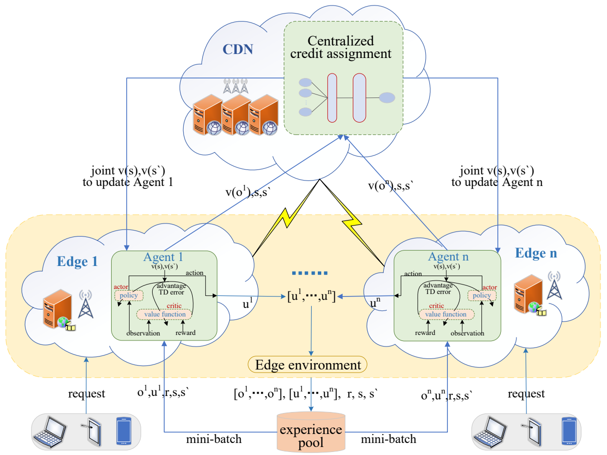


Fig. 2: The distributed video content placement framework.

In single agent Actor-Critic method, each agent's policy updating with a mini-batch experience by using advantage Temporal Difference (TD) error is shown as follows:

$$\theta = \theta + \alpha \forall_\theta log\pi(a|\tau)(Q(s, a) - V(s)) \tag{27}$$

where α is the learning rate, τ is the mini-batch sampled from the experience buffer pool. $Q(s, a) - V(s)$ is the advantage function. a is the current action. $Q(s, a)$ is the state-action reward value which is estimated by $Q(s, a) = r + \gamma V(s')$ by taking the action a with the state s following agent's policy π. s' is the next-time-slot state. γ is the discount factor of accumulation reward. Especially, V value is the critic network's output that denotes the state value (the estimated reward after the state s occurred), which can evaluate the policy performance of actor network. Here, we adopt the actor-critic framework for each distributed agent to learn its own video content placement policy.

5.2 Design of Proposed Distributed Framework

Fig. 2 depicts the distributed video content placement framework: there are totally N_e (for simplicity, we use n to denote N_e in Fig. 2) edge clouds, and each of them is equipped with an actor-critic RL agent. Moreover, a centralized credit assignment model is used to control each agent's policy evaluation from a global perspective by taking the global state s as input. In time slot t, all agents take their actions to form the joint action $U_t = [u_t^1, ..., u_t^n]$. Then, edge environment uses U_t with a set of observations $O_t = [o_t^1, ..., o_t^n]$ to complete the state transition and return the global reward r_t as well as the next-time-slot global state s_t'. A historical record tuple including $< O_t, U_t, r_t, s_t, s_t' >$ is inserted into the experience buffer pool. Then, each edge cloud takes *batch_size* mini-batch records to train its own RL agent (including policy network and critic network). The credit contribution of each agent is uploaded to the credit assignment tool on CDN that is trained by mini-batch $< v(o)_t, r_t, s_t, s_t' >$. Next, the joint credit $V(s)$ is used to update the distributed RL agents by calculating the advantage TD error. Even though a centralized server is needed in Fig. 2 (only) for a small-scale state information transmission and no weight exchanges, it makes Fig. 2 not a fully distributed framework. Nevertheless, the local state is still private for each edge cloud and does not need to be shared among different edge clouds. To measure the contributions to the global optimized objective from each edge cloud, part of the parameters are shared in the centralized server. To guarantee privacy and decrease expenditure, we set the global state information, which is transmitted between the centralized server and edge clouds during the learning process of our distributed method, to be in a small size. Next, we design each RL agent as follows:

5.2.1 *Episode, Step and Constraints*

In each time slot t, a set of requests R_t is collected from different edge areas. We regard a completed response process of R_t as an episode. Each RL agent decomposes an episode into $|R_t|$ steps, where a request $r(f,b,n)$ with video file f of bitrate b from its local edge cloud n is accommodated by DMVP. Hence, the limitation of an episode's life span in time slot t is set as a constant $|R_t|$.

There are two considered constraints in the learning process of DMVP: the capacity constraint and the cost consumption constraint. However, the constraint judgement of DMVP is executed at the end of the algorithm, which wastes more cost for RL agents. In order to amend the learning direction in time, we convert Eq. (6) to Eq. (28) and convert Eq. (8) to Eq. (29).

$$\begin{gathered}\sum_{\beta\in B, f'\in F: f'==f} \max_{r\in R'_t} X_{n,t}^{r,\beta} * S_{\beta}^{f'} \le \eta(n) \\ \forall t\in T, R'_t\in R_t, n\in\mathcal{N}_e\end{gathered} \tag{28}$$

$$\begin{gathered}\sum_{r\in R'_t}\sum_{\beta\in B} X_{n,t}^{r,\beta}\cdot(S_\beta^f - S_b^f)\omega_n + \\ \sum_{f\in F,\beta\in B}\max_{r(f,b,\alpha,u)\in R'_t}\left(X_{n,t}^{r,\beta}\cdot S_\beta^f\cdot\pi_n\right)\le C_n \\ \forall t\in T, R'_t\in R_t, n\in\mathcal{N}_e\end{gathered} \tag{29}$$

where R'_t is a subset of R_t with size of ts. It denotes the requests that occur before ts steps. In each step of an episode, Eq. (28) and Eq. (29) determine whether an agent can take its action u_{ts} on ts-th step.

5.2.2 *Observation and Action*

For an RL agent $n\in N_e$, its observation o^n on ts-th step consists of the storage information of each video file and the residual capacity information, which is denoted by

$$o^n = [Z_{n,ts}^{f,\beta}, \overline{C}_{n,ts}]\ \forall f\in F, \beta\in B \tag{30}$$

where $Z_{n,ts}^{f,\beta}$ is 1 if video file f with bitrate β has already been placed in edge cloud n on ts-th step, and 0 otherwise. $\overline{C}_{n,ts}$ denotes the residual capacity of each edge cloud, which is calculated by

$$\begin{gathered}\overline{C}_{n,ts} = \eta(n) - \sum_{f\in F,\beta\in B}[Z_{n,ts}^{f,\beta} * S_\beta^f]) \\ \forall ts\in|R_t|, n\in N_e\end{gathered} \tag{31}$$

So, each agent's observation size is $|F| * |B| + 1$.

For a requested file f with bitrate b, agent n's action $u_{n,ts}^{f,b}$ is set as 1, which means agent n can place this file in its edge cloud on ts-th step of an episode, and 0 otherwise. So, the joint action U for the request $r(f,b,n)$ is denoted by $U = [u_{1,ts}^{f,b}, ..., u_{N_e,ts}^{f,b}]$.

5.2.3 *Joint Observation, Action and State*

The joint observation O is denoted by $[o^1, o^2, ..., o^n]$, where o^i in Eq. (30) is generated by the interaction between the edge cloud agent i and edge environment as shown in Fig. 2. The joint action U is denoted by $[u^1, u^2, ..., u^n]$, where u^i represents the action taken by agent i in the current iteration. The joint observation O and the joint action U are just regarded as storage forms in the experience pool. When via mini-batch sampling, they are split into a series of individual o and u that are input into each agent, as shown in Fig. 2. The joint state s, denoted by $[\overline{C}_{n,ts}]$ with the size of $|N_e|$, is the input of the centralized credit assignment model, which is globally collected by CDN.

5.2.4 *Reward*

Recall the definition in Eq. (1), we denote a utility function in Eq. (32) for DMVP by calculating the global reward on each step of an episode.

$$\mathbb{U}_{ts} = \frac{\mathrm{c}}{T(m,n|r_{ts}(f,b,n)) + \Gamma_n(f,b,\beta|r_{ts}(f,b,n))} \tag{32}$$

where $T(m,n|r_{ts}(f,b,n))$ is the path delay of request $r_{ts}(f,b,n)$ which is sent from n and responded on m. It is set as 0 when $m == n$. If there is no file f with bitrate b, we search f with high-bitrate to accommodate the request r_{ts} by transcoding from bitrate B to bitrate b that causes the cost $\Gamma_n(f,b,\beta|r_{ts}(f,b,n))$. Especially, when edge cloud n accommodates $r_{ts}(f,b,n)$ by video file f with bitrate b, we use an extra reward $\frac{\mathrm{c}}{r_{ext}}$ (r_{ext} is a much smaller value) to encourage the learning process.

5.3 Credit Assignment and An Improved Policy Network Update

5.3.1 *Credit Assignment Based on Value Decomposition*

The credit assignment mechanism is the basis of the cooperation among distributed agents. Each agent's decision can affect the joint video content placement policy by the global reward. The contributions to the joint policy optimization from all distributed agents are completely different because each agent has respective updating progress of distributed policy. The agent with a faster learning process can contribute more to the global reward, which brings a sharp increase in the return reward. The advantage function $Q(s,U) - V(s)$ of MARL suddenly becomes bigger, which can cause the agent with a slower learning process to miss its local optimum policy. We introduce the credit assignment mechanism which can deal with the inconsistent updating of distributed agents by a credit assignment tool. It can avoid "lazy agent" (seldomly updating its policy) when the global reward becomes larger due to other agents' positive policies. The credit assignment tool is implemented by Value Decomposition (VD) scheme [39] which can control the distributed critic network updating by inputting each distributed v and defining their loss $L(\omega)$ with the output $V^{tot}(s)$ of the Mixing network, as is shown in Fig. 2.

The centralized critic network can also provide a global evaluation of all agents' policies [40]. But it depends on the frequent large-size observation information transmission between edge clouds and CDN. As we all know that better

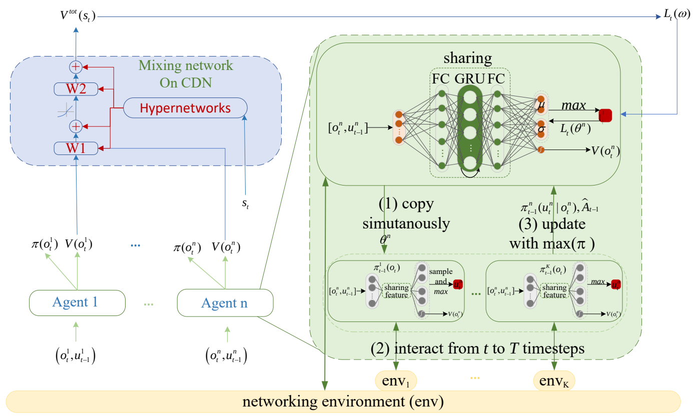


Fig. 3: Parallelization of policy network update.

policies for all distributed agents can bring a better performance of the whole model. Hence, we adopt the VD scheme by introducing the Mixing network whose weights are all non-negative to verify the positive correlation between distributed $V(o)$ value and global $V(s)$ value. Hypernetwork, consisting of a single linear layer with $ReLU$ activation function, takes the global state s as input to generate the Mixing network's weights and bias as is shown in Fig. 3. In order to avoid a large amount of information transmission, we set the global state input of the Mixing network as the features including the number of idle storage units in each edge cloud.

Based on the global $V(s)$, the distributed critic network with parameter ω is updated to minimize the following loss on time slot t:

$$\begin{aligned} L_t(\omega) &= (y_t - V(s_t))^2 \\ &= (y_t - f_{mix}(V_{\omega^1}(o_t^1), ..., V_{\omega^n}(o_t^n)))^2 \end{aligned} \tag{33}$$

where $y_t = \sum_{i=t}^{k-t-1} \gamma^i r_i + \gamma^{k-t} V^{tot}(s^k)$ is the global accumulative reward from the last state s_k. k is upper-bounded by the episode length. f_{mix} denotes the Mixing network, a non-negative non-linear map between $V(o)$ and $V(s)$ in Fig. 3. It is noted that Mixing network's parameter is also updated by Eq. (33).

5.3.2 An Improved Policy Network Update

To implement an approximately distributed decision-making for video content placement, we equip each edge agent with both actor policy network and critic network. Then, the VD scheme is introduced to implement cooperation among all agents via the centralized credit assignment model (Mixing network). It has a slower learning process with taking the small-scale state information with a lower frequency of information transmission. Hence, we must provide more accurate $v(o)$ values of each agent for the credit assignment model in each joint updating process to speed up the learning process. We improve the updating of the policy network resort to parallelization and sharing the feature abstraction between the policy network and the critic network in each edge agent, as is shown in Fig. 3. The details of updating the policy network are specified as follows:

Firstly, we extend the policy gradient of distributed actor network by Proximal Policy Optimization (PPO) [41] to relieve the weakness caused by stable learning step size of the advantage function $A(s, U)$. Kullback-Leible (KL) divergence [42] is proposed based on relative entropy theory, which is used to measure the difference between two kinds of distributions. For the policy distribution ($\pi(\cdot|o)$ and $\pi_{old}(\cdot|o)$) of discrete action variables, their KL divergence is extended as

$$KL[\pi_{old}(\cdot|o), \pi(\cdot|o)] = \sum_u \pi_{old}(u|o) log \frac{\pi_{old}(u|o)}{\pi(u|o)} \tag{34}$$

which can measure the difference between new and old policy network model. Because of its asymmetry property ($KL[\pi_{old}(\cdot|o), \pi(\cdot|o)] \neq KL[\pi(\cdot|o), \pi_{old}(\cdot|o)]$) [42] which can bring more noisy step size for the advantage function value, we therefore replace KL divergence as $\frac{\pi(u_t^a|o_t^a)}{\pi_{old}(u_t^a|o_t^a)}$ in loss function

$$L_t(\theta^a) = min(r_t(\theta^a) A_t^{old}, clip(r_t(\theta^a), 1 - \varepsilon, 1 + \varepsilon) A_t^{old}) \tag{35}$$

where $r_t(\theta^a) = \frac{\pi_t(u_t^a|o_t^a)}{\pi_{old}(u_t^a|o_t^a)}$. $clip()$ function with threshold ε (usually set as 0.02 [41]) is used to filter the over-large or over-small advantage value A_t^{old} for each agent. Further, $mini$ value is used to avoid missing better policy. For elaborated calculation, we define the policy probability outputted from the Softmax layer shown in Fig. 3 as

$$\pi(u_t^a|o_t^a) = \pi(\mu_t^a|o_t^a) + \pi(\sigma_t^a|o_t^a) \tag{36}$$

where μ and σ are used to randomly select action u_t^a through Gaussian distribution. To calculate policy loss, we need to store the policy network parameter θ_{t-1} of each agent in advance. Then, we can obtain $\pi_{old}(u_t^a|o_t^a)$ and $\pi(u_t^a|o_t^a)$ through inputting o_t^a respectively into the policy network θ_{old} and θ in time slot t to calculate $r_t(\theta^a)$.

Secondly, in order to speed up the convergence of training return, we parallelize K policy networks (called "workers") that asynchronously interact with parallelized networking environments to minimize original advantage TD loss from time slot t. Hence, the loss function in Eq. (35) is improved as the following:

$$L_t(\theta^a) = \mathbb{E}_t[min(r_t(\theta^a) \widehat{A}_t^{old}, clip(r_t(\theta^a), 1 - \varepsilon, 1 + \varepsilon) \widehat{A}_t^{old})] \tag{37}$$

where $\widehat{A}_t^{old}$ is an estimated value (mean) of advantage function with old policy (parallelized workers' policy), calculated by

$$\widehat{A}_t^{old} = -V_{old}(o_t) + r_t + ... + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V_{old}(o_T) \tag{38}$$

where γ is the discount factor. T is the time step, until when each parallelized model runs the policy from time slot t. The target policy network is updated by the parallelized worker that has the highest action probability $\pi(u_t|o_t)$.

Lastly, we add the entropy item of target action distribution on each time slot to avoid the over-convergence as Eq. (39).

$$L_t(\theta^a) = L_t(\theta^a) + \kappa H(\pi^a(*|o_t^a, \pi_{t-1}^a)) \tag{39}$$

where κ is a non-negative hyperparameter of entropy item $H(*)$. High entropy enlarges the range of action probability which can improve agent's exploration performance.

5.4 Convergence Analysis of DMVP

We use the following lemma to substantiate the local convergence of the proposed DMVP.

Lemma 1. *For the improved method, its policy gradient*

$$g_k = \mathbb{E}_k[min(\bigtriangledown_{\theta_k} logr(\theta)\widehat{A}^{old}, \\ clip(\bigtriangledown_{\theta_k} logr(\theta), 1-\varepsilon, 1+\varepsilon)\widehat{A}^{old})] \\ + \kappa \bigtriangledown_{\theta_k} H(\pi(*|o_t, \pi')) \tag{40}$$

at each iteration k, lim inf_k $|| \bigtriangledown L|| = 0$.

Proof. For parallelized workers, its policy updating follows the gradient

$$g' = \mathbb{E}_{\pi}[\sum_a \nabla_\theta log\pi(u|\tau^a)A(s, \mathrm{u})] \tag{41}$$

where $A(s, \mathrm{u}) = \mathrm{Q(s,u)} - \mathrm{V^{tot}(s)}$ is the Temporal Difference (TD) error of a distributed critic. We decompose g' through $g' = g'_1 + g'_2$ to prove each part's convergence, where

$$g'_1 = \mathbb{E}_{\pi}[\sum_a \nabla_\theta log\pi(u^a|\tau^a)Q(s, \mathrm{u})] \\ = \mathbb{E}_{\pi}[\nabla_\theta log\prod_a \pi(u^a|\tau^a)Q(s, \mathrm{u})] \tag{42}$$

which yields a standard single-agent actor-critic policy gradient that converges to a local maximum of the expected return [39]. And,

$$g'_2 = -\mathbb{E}_{\pi}[\sum_a \nabla_\theta log\pi(u^a|\tau^a)V(s)] \\ = -\mathbb{E}_{\pi}[\nabla_\theta log\prod_a \pi(u^a|\tau^a)V(s)] \tag{43}$$

where the joint policy is written as a product of independent actors. Hence, $V(s)$ can be calculated by $f(V(o^1), ..., V(o^n))$ where f is a non-negative function (Mixing network). [39] proved that g'_2 converges to 0 resorting to the conversion from the policy expectation to a discounted ergodic state distribution.

For target policy network, omitting the entropy item, we have:

$$g = \mathbb{E}_{\pi}[\mathcal{F}_{mix}(\nabla_\theta logr(\theta)\widehat{A})] \tag{44}$$

where $\mathcal{F}_{mix}$ represents the positive coefficient of credit assignment. So, we arrive at:

$$g \approx \mathbb{E}_{\pi}[\lambda_1 \nabla_\theta logr(\theta^1)\widehat{A}^1 + ... + \lambda_n \nabla_\theta logr(\theta^n)\widehat{A}^n] \\ = \mathbb{E}_{\pi}[\nabla_\theta log\prod_a [r(\theta^a)\widehat{A}^a]^{\lambda_a}] \tag{45}$$

where $\lambda_i > 0, i \in \{1, ..., n\}$. Eq. (45) yields a standard single-agent PPO policy gradient $\mathbb{E}_{\pi}[\nabla_\theta logr(\theta^a)\widehat{A}^a]$, which means that a TD error-based PPO (KL divergence: $r(\theta) = \frac{\pi(u|o)}{\pi_{old}(u|o)}$) can converge to a local optimal policy of a loss function with an entropy item [43]. The proof is therefore complete.

After that we can guarantee the local convergence of DMVP's policy gradient including the parallelized workers' an the target policy network's. Via the training process (guided by the policy gradient g_k that can locally convergence to 0), DMVP can learn a local optimal policy for the selection of joint action U to obtain a locally largest accumulated reward which is related to the joint policy

$$\mathbb{U} = \sum_{ts=0}^{|R_t|} \mathbb{U}_{ts} \tag{46}$$

where $\mathbb{U}_{ts}$ is calculated by Eq. (32). Correspondingly, we can obtain a locally smallest Objective in Eq. (2), as we define in problem definition. □

6 Simulations

6.1 Simulation Setup

6.1.1 Centralized Scheme

In the centralized video content placement scheme, we use the tool GT-ITM [44] to generate one MEC network topology with 40 nodes in a 100 units * 100 units square. We let 10 out of 40 nodes be the edge cloud nodes, and all the other 30 nodes be the intermediate nodes. The capacity of each edge cloud node is set to 5 *Gb*. For each link, its capacity is randomly picked in $[1, 2]$ *Gb/s* and its delay takes value in $[10, 40]$ *ms*. For simplicity, we add one more node as the CDN server node and connect it with each edge cloud node with a set of paths whose delay falls in the range of $[200, 400]$ ms. There are in total 40 video files with 5 bitrate levels, which are 360p, 480p, 720p, 1080p and 1440p. The required bandwidth and transcoding delay for a video file with different bitrate levels are shown in Table 3 according to [6] and [45]. The video access pattern is assumed to follow

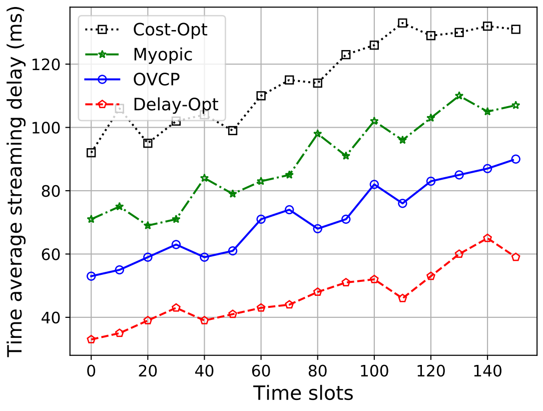
TABLE 3: Bandwidth and transcoding delay.

Versions (p)	1440	1080	720	480	360
bandwidth (Mbps) [45]	6	3	1.5	0.5	0.4
transcoding delay(s) [6]	NA*	0.27	0.19	0.16	0.13

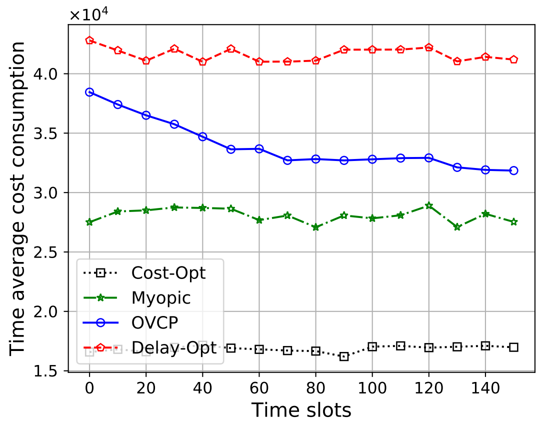
*This value will not be used since transcoding can only happen from a high bitrate level to a low bitrate level [8].

the Zipf distribution according to [5], and each video file f_i $(1 \leq i \leq 40)$ with bitrate level j $(1 \leq j \leq 5)$ has an accessing probability: $p_{ij} = \frac{(i*j)^{-z}}{\sum_{1 \leq a \leq A} a^{-z}}$ $, \forall 1 \leq a \leq A$, where $A = 40 * 5 = 200$ is the number of all the distinct video files with different bitrate levels and z is set to 0.8 which denotes the aggregation degree of video requests similar to [8]. Without loss of generality, we regard that the video file consists of a series of same-duration chunks (the duration is set 10 seconds in this case), and each 10-seconds chunk takes up 0.5 *Mb* according to [46]. We first set $S^f_\beta \in [50, 500]$ *Mb*, where $\beta = 1440$p. Afterwards, for a same video file f with bitrate level β' in {360p, 480p, 720p, 1080p}, we set $S^f_{\beta'} = \frac{\beta'}{1440} \cdot S^f_{1440}$. Accordingly, we set $\omega_n \in [0.01, 0.1]$ and $\pi_n \in [0.01, 0.12]$ [47], where $n \in \mathcal{N}_e$. The transmission rate for all the requests is set to 20 *Mb/s* for simplicity. Due to the lack of workload trace of public accessible video requests in MEC, we randomly generate 100 requests among all the edge areas in each time slot, and we generate such 100 requests for 150 time units in total. We set $K = 10$ and $C_n = 3000$ for all $n \in \mathcal{N}_e$. We first set $V = 1$ in OVCP and compare OVCP with following three algorithms:

- Delay-Opt: It only minimizes the video streaming delay without considering the cost consumption in



(a) Time average streaming delay.



(b) Time average cost consumption.

Fig. 4: Performance of centralized video content placement algorithms.

TABLE 4: Parameter setting in distributed algorithm

Parameter	Setting	Parameter	Setting
α	0.005	γ	0.9
buffer size	1000	$\|R\|$	40
batch size	32	κ	0.01
workers' quantity	4	ϵ	0.02
workers' step size	$\|R\|$	c	100
MixingNet-embed-dim	32	HyperNet-edbed-dim	32
FC/GRU units	64	edge node capacity	$5Gb$
T in parallelization	$t+20$	r_{ext}	5

each time slot. Delay-Opt can therefore provide a (loose) lower bound of video streaming delay.

- Cost-Opt: It only focuses on minimizing the total cost consumption without caring about video streaming delay in each time slot. Cost-Opt can therefore provide a (loose) lower bound of cost consumption.
- Myopic Caching: It imposes a hard cost constraint C_n in each time slot and runs the optimization formulation in Eqs. (2)-(8) to minimize the total video streaming delay. In case there is no feasible solution because of a large number of requests in some time slot(s), Myopic will iteratively remove one request from R_t and run the optimization again. This process continues until a solution is returned or R is empty.

The simulation for the centralized method was run on a high-performance desktop PC with 8 core 3.40GHz Intel(R) Core(TM) i7-6700 processors and 16 GB memory. We use IBM ILOG CPLEX (CPLEX Callable Library interface) 12.6 with Java 1.8.0_221 to implement the centralized algorithm.

6.1.2 *Distributed Scheme*

The simulation setup of the distributed video content placement scheme is the same as the centralized simulation setting, such as the edge node density, the distribution of requests, the number of video files, and so on. The distributed algorithm adopts the delay-greedy method to deal with path selection. Moreover, we set the number of video files as 200 in an RL episode. Other parameter settings of our MARL-based algorithm are listed in Table 4. We compare our distributed algorithm DMVP with the following 3 methods:

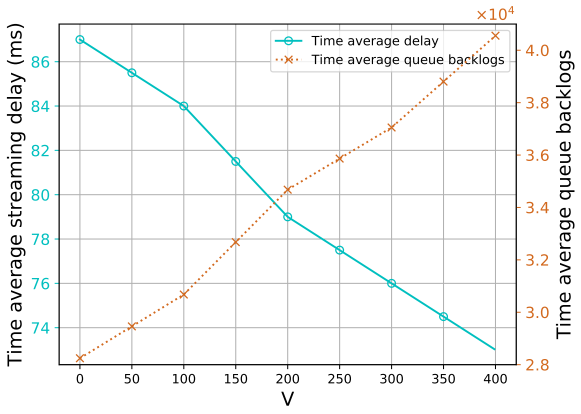
- Independent Actor-Critic [48] (IAC): It is a Multi-Agent method with Actor-Critic model without applying the credit assignment and each agent in IAC is independent and not interfering.
- Multi-Agent Deep Deterministic Policy Gradient [49] (MADDPG): This method performs a centralized learning process and distributed deployment at the expense of frequent large-scale information exchanges.
- Multi-Agent Value Decomposition Actor-Critic [39] (VDAC): It is similar to DMVP but without improved policy network updating.

For distributed video content placement simulation, the proposed framework is executed by Pytorch 1.9.0 with Cuda 11.1.

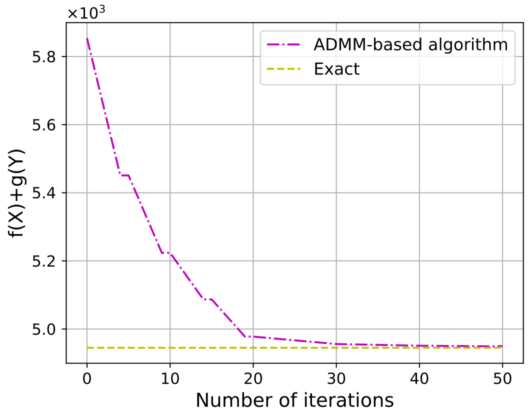
6.2 Simulation Results

6.2.1 *Centralized Video Content Placement Scheme*

Fig. 4(a) and Fig. 4(b) respectively show the time average delay and costs over different time slots. We see from Fig. 4(a) that Delay-Opt can always achieve the lowest average delay value. The reason is that Delay-Opt always tries to place the requested video file on the local or nearest edge clouds in order to minimize the streaming delay without considering whether it is expensive to store files on the node. In this sense, Delay-Opt consumes the largest cost consumption, as shown in Fig. 4(b). Our proposed OVCP can obtain the second lowest delay value by satisfying the long-term cost constraint, which verifies its correctness and shows its performance efficiency. Myopic has a higher time average delay than OVCP, because Myopic does not efficiently make use of cost constraint in a long-run perspective. More specifically, due to the reason that in each time slot a cost constraint C_n is imposed for each edge cloud node, it happens that the per time slot constraint C_n can be far above the needed cost in some slots. It also happens that C_n is far below the needed cost in some slots for high-demanding traffic, and in order to meet the constraint, some of the requests have to be rejected,

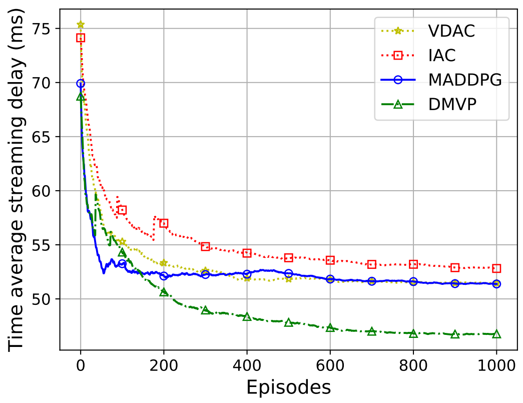


(a) Time average streaming delay and queue backlogs with V.

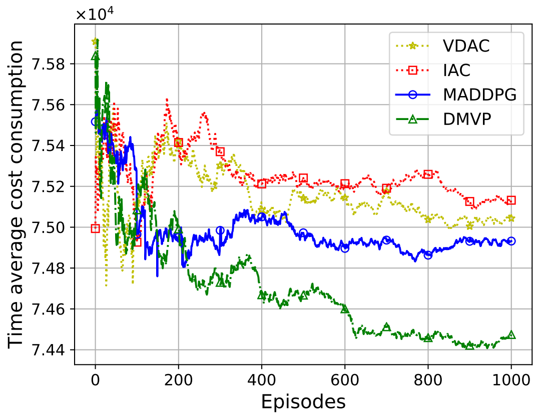


(b) Convergence of ADMM-based algorithm.

Fig. 5: Performance of proposed centralized OVCP algorithm.



(a) Time average streaming delay.



(b) Time average cost consumption.

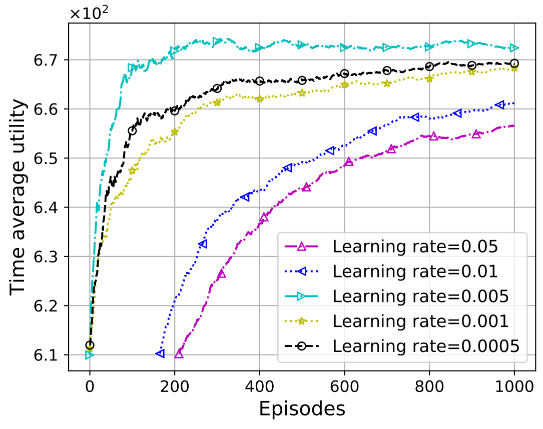
Fig. 6: Performance of distributed video content placement algorithms.

and this will "save" more cost. This also reflects the effect of long-term cost constraint C_n on the performance of video streaming delay and total cost consumption. Meanwhile, Cost-Opt has the highest average streaming delay since it is a delay-oblivious algorithm and always places the file in order to minimize the total cost consumption. In this sense, Cost-Opt consumes the lowest cost as seen in Fig. 4(b). Myopic has a total time average cost consumption less than $C_n \cdot N_e$ as shown in Fig. 4(b) and achieves the second lowest cost consumption. OVCP can achieve the third lowest cost consumption and obtains a close performance with Myopic when $T = 150$, which indicates that OVCP always tries to reduce the total cost consumption in a best-effort manner.

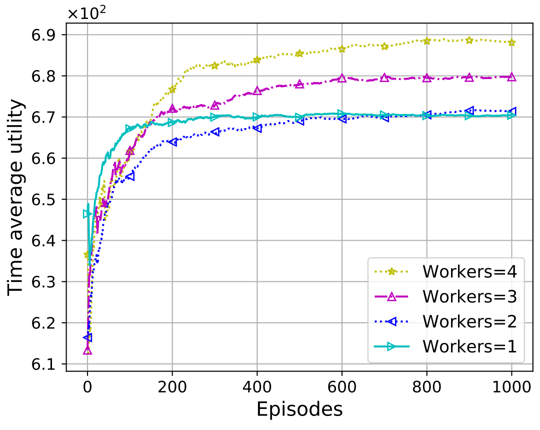
Fig. 5(a) illustrates the time average streaming delay and time average queue backlog values with different V. As V increases from 1 to 400, the time average delay values decreases, which is proved in Eq. (24) in Theorem 2, and the time average backlog values increases, which is reflected in Eq. (25) in Theorem 2. As a result, Fig. 5(a) shows that OVCP can achieve a tradeoff between delay minimization and queue stability, corresponding to Theorem 2. We then evaluate the convergence of Algorithm 2. To do that, we set $Q(n) = 1$ for $n \in \mathcal{N}_e$. We randomly generate one time slot traffic R and run Algorithm 2 to solve (**P1**), where $f(X) + g(Y)$ represents the total objective function in Eq. (22). Fig. 5(b) reveals that the ADMM-based algorithm can converge in 20 (finite) steps, which verifies the correctness of Theorem 1. Moreover, we also directly run the exact solution in Eq. (15) to solve the same problem instance, where we can see that Algorithm 2 can achieve the same performance after convergence with the exact solution. This has also verified the correctness of Algorithm 2.

6.2.2 *Distributed Video Content Placement Scheme*

Fig. 6 shows the performance analysis on streaming delay and cost consumption of DMVP with IAC, MADDPG, VDAC. All algorithms have a similar setting of common hyperparameters. We execute 1000 episodes to verify the reliability of each distributed algorithm and respectively discuss the learning results from time average streaming delay and cost consumption. As is shown in Fig. 6(a), DMVP achieves the lowest streaming delay performance and the smallest cost consumption performance after convergence, although its learning speed is slower than MADDPG in the initial training stage. MADDPG adopts a centralized training method in which an integrated RL agent with a large-scale state input learns the optimal placement policy on CDN. Because of the lower frequency of exploration on

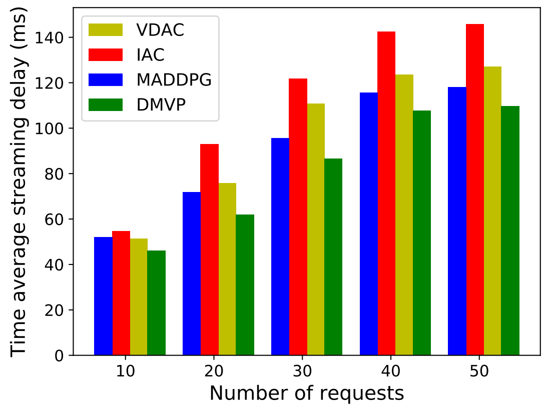


(a) Agent's learning rate.

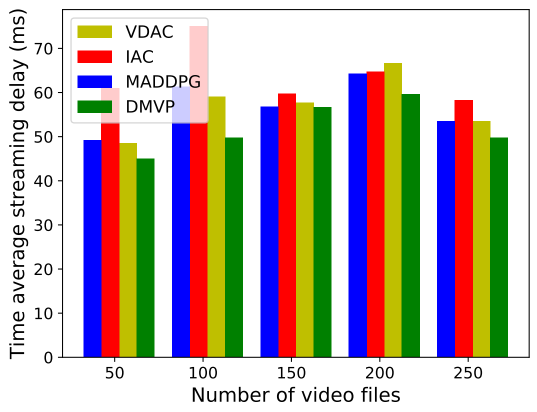


(b) Number of parallelized workers.

Fig. 7: Utility value of DMVP under different parameters.



(a) Number of requests in each edge cloud.



(b) Number of video files.

Fig. 8: Performance of DMVP under different problem settings.

the policy network, VDAC and MADDPG finish convergence early at about 400 episodes. IAC does not obtain a good result when facing such a complicated optimization task due to the completely non-stationary and the lack of cooperation among all distributed agents. The limited number of bitrate leads to a violent oscillation in cost consumption even though it is calculated by time average. By comparing Fig. 6(a) and Fig. 4(a), we can see that our distributed algorithm DMVP has comparable performance to our centralized algorithm OVCP on the time average streaming delay when they deal with the problems with the similar setup.

Fig. 7 shows the performance of DMVP with different hyperparameters. Fig. 7(a) shows the learning rate adjustment of distributed policy network and distributed critic network. We can see that larger learning rates with 0.05 and 0.01 lead to a slower convergence for DMVP and smaller learning rates with 0.0005 and 0.001 make DMVP falls into the local optimal in advance. In particular, DMVP under learning rate 0.005 can obtain a better performance no matter on convergence speed or final utility value after convergence of about 200 episodes. Fig. 7(b) shows the influence of the number of parallelized workers on time average utility. We can see that few parallelized workers (e.g., 1 worker) can cause a non-smooth learning process in the initial training stage. On the contrary, DMVP with 4 parallelized workers for each agent can achieve better performance. Moreover, We also change the problem scale by increasing the number of requests generated in each edge cloud area and the number of video files that can be requested. Fig. 8(a) demonstrates that more requests result in a larger time average streaming delay. This is because, when the number of requests increases, most agents can hardly satisfy all local requests because of storage limitation and have to relay requests to remote CDN, which incurs larger path delay. Fig. 8(b) shows the streaming delay performance of all the algorithms with different number of video files. We see that the number of video files does not impact the streaming delay performance of all the algorithms with a clear trend, but DMVP can always achieve the smallest streaming delay performance. In all, Fig. 8 verifies that DMVP behaves scalable and effectively under different problem settings.

6.2.3 Comparison between centralized and distributed methods

Based on the same simulation setup in Section 6.2.1 and 6.2.2 and in order to compare them fairly, on the one hand, we choose the time-average performance of OVCP during 150 time slots. On the other hand, for DMVP, we adopt

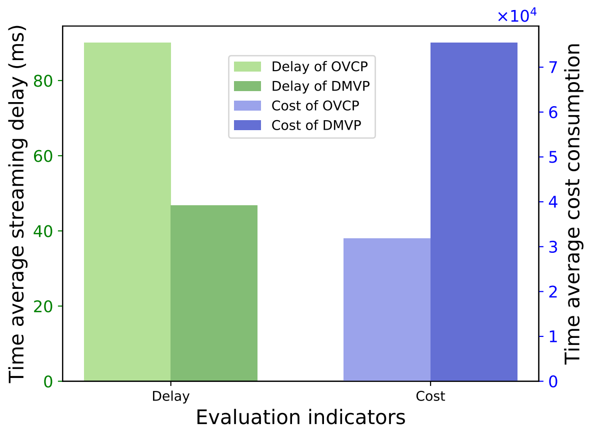


Fig. 9: Comparison between centralized and distributed methods.

the performance of DMVP during the training process after convergence based on the pre-adjusted hyper parameters. Fig. 9 demonstrates the comparison respectively of time average streaming delay and time average cost consumption. We can see that the distributed method DMVP performs better in terms of time average streaming delay than the centralized method OVCP. It is because the greedy-delay path forwarding selection plays a more important role than cost optimization during the learning process of DMVP. Accordingly, as shown by the blue histogram in Fig. 9, the time average cost consumption of DMVP is larger than the centralized method OVCP's. This comparison result also reflects and corresponds to the working paradigm of these two algorithms. Centralized OVCP focuses on the long-term system performance regarding the time-average delay by obeying a long-term time average cost constraint. DMVP works in a distributed manner and always tries to accommodate the request in local or nearby edge cloud greedily by a reward function, which neglects the cost consumption and hence whole system performance.

To conclude, the advantages of the centralized method OVCP lie in globally guaranteed system performance and less/no hyper-parameter adjustment. The disadvantages of OVCP are its overhead for obtaining the global network view and the failure of the central decision-maker node. The advantages of the distributed method DMVP are less information exchanging and updating overhead and efficient response on a well-trained model. The disadvantages of DMVP are its cost for hyper-parameter configuration and model training, as well as no theoretical performance guarantee.

7 Conclusion

In this paper, we study the video content placement problem at the network edge from both centralized and distributed perspectives. We first devise a centralized online video content placement framework by leveraging the Lyapunov optimization technique to decompose the considered problem into a series of one-time-slot problems and then apply an ADMM-based method to solve each one time-slot problem. We prove that the proposed framework has an $O(V, 1/V)$ tradeoff between video streaming delay and queue backlog. We further propose a distributed multi-Agent-based algorithm by introducing value decomposition to implement credit assignment among all edge clouds and improving the policy updating method by parallelization. Simulation results reveal that our proposed centralized Online Video Content Placement (OVCP) framework can achieve arbitrarily close-to-optimal total time average video streaming delay while still maintaining the long-term individual edge cloud cost budget. Moreover, our distributed video content placement algorithm can be verified to achieve lower time average delay compared to the contrast methods and has strong robustness on a large-scale edge cloud environment.

It is worthwhile to mention that our proposed solutions are general to solve the video content problem in edge computing, which means that they can also deal with real-world experimental scenario. On the one hand, we try to simulate the real-world scenario as much as possible in our work. The video requests are generated according to [5], which follows similarly with the real-world video request scheme in edge network from Youtube in recent years. The edge network topology is randomly generated with various distances and node capacities. The value of bandwidth, types of video bitrate, and transcoding delay in the simulation are chosen according to the real-world applications. Based on these simulation setup, we evaluate the performance for centralized and distributed solutions. On the other hand, our solutions are also feasible for the larger-feature space resorting to their advantages of lower complexity and convergence improvement. Therefore, the proposed solutions are expected to perform efficiently and effectively in the real-world experimental scenario.

Acknowledgments

The work of Song Yang is partially supported by the National Natural Science Foundation of China (NSFC) under Grant No. 62172038. The work of Fan Li is partially supported by the NSFC under Grant No. 62072040. The work of Pan Zhou is partially supported by NSFC under grant no. 61972448. The work of Xiaoming Fu is partially supported by the EU H2020 RISE COSAFE project under Grant No. 824019. Song Yang is the corresponding author.

References

[1] "Cisco visual networking index (vni) global and americas/emear mobile data traffic forecast, 2017–2022," 2019. [Online]. Available: https://www.cisco.com/c/dam/m/en_us/network-intelligence/service-provider/digital-transformation/knowledge-network-webinars/pdfs/190320-mobility-ckn.pdf
[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing—a key technology towards 5G," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.
[3] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *IEEE INFOCOM*, 2020.
[4] B. A. A. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications surveys & tutorials*, vol. 16, no. 3, pp. 1617–1634, 2014.
[5] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, 2007, pp. 15–28.
[6] F. Wang, C. Zhang, J. Liu, Y. Zhu, H. Pang, L. Sun *et al.*, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized QoE," in *IEEE INFOCOM*, 2019, pp. 910–918.

[7] F. Wang, F. Wang, J. Liu, R. Shea, and L. Sun, "Intelligent video caching at network edge: A multi-agent deep reinforcement learning approach," in *IEEE INFOCOM*, 2020.

[8] Z. Qu, B. Ye, B. Tang, S. Guo, S. Lu, and W. Zhuang, "Cooperative caching for multiple bitrate videos in small cell edges," *IEEE Transactions on Mobile Computing*, vol. 19, no. 2, pp. 288–299, 2020.

[9] T. X. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 18, no. 9, pp. 1965–1978, 2018.

[10] K. Bilal, E. Baccour, A. Erbad, A. Mohamed, and M. Guizani, "Collaborative joint caching and transcoding in mobile edge networks," *Journal of Network and Computer Applications*, vol. 136, pp. 86–99, 2019.

[11] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *IEEE INFOCOM*, 2020, pp. 1–10.

[12] Y. Wang, Y. Zhang, X. Han, P. Wang, C. Xu, J. Horton, and J. Culberson, "Cost-driven data caching in the cloud: an algorithmic approach," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[13] R. Li, L. Wang, Y. Gong, M. Song, M. Pan, and Z. Han, "Dynamic cache placement, node association, and power allocation in fog aided networks," in *IEEE Global Communications Conference*, 2019, pp. 1–6.

[14] X. Xia, F. Chen, Q. He, J. Grundy, M. Abdelrazek, and H. Jin, "Online collaborative data caching in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 281–294, 2021.

[15] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent NFV middleboxes," in *IEEE INFOCOM*, 2017.

[16] C. You and L. M. Li, "Efficient load balancing for the VNF deployment with placement constraints," in *IEEE International Conference on Communications (ICC)*, May 2019, pp. 1–6.

[17] H. Hawilo, M. Jammal, and A. Shami, "Network function virtualization-aware orchestrator for service function chaining placement in the cloud," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 643–655, 2019.

[18] M. A. Khoshkholghi, M. Gokan Khan, K. Alizadeh Noghani, J. Taheri, D. Bhamare, A. Kassler, Z. Xiang, S. Deng, and X. Yang, "Service function chain placement for joint cost and latency optimization," *Mobile Networks and Applications*, vol. 25, no. 6, pp. 2191–2205, 2020.

[19] S. Yang, F. Li, S. Trajanovski, R. Yahyapour, and X. Fu, "Recent advances of resource allocation in network function virtualization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 295–314, 2021.

[20] K. Poularakis, G. Iosifidis, A. Argyriou, I. Koutsopoulos, and L. Tassiulas, "Distributed caching algorithms in the realm of layered video streaming," *IEEE Transactions on Mobile Computing*, vol. 18, no. 4, pp. 757–770, 2018.

[21] X. Lyu, C. Ren, W. Ni, H. Tian, R. P. Liu, and X. Tao, "Distributed online learning of cooperative caching in edge cloud," *IEEE Transactions on Mobile Computing*, 2020.

[22] J. Moura and D. Hutchison, "Game theory for multi-access edge computing: Survey, use cases, and future trends," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 260–288, 2018.

[23] Q. He, G. Cui, X. Zhang, F. Chen, S. Deng, H. Jin, Y. Li, and Y. Yang, "A game-theoretical approach for user allocation in edge computing environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 3, pp. 515–529, 2019.

[24] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[25] C. Zhong, M. C. Gursoy, and S. Velipasalar, "Deep multi-agent reinforcement learning based cooperative edge caching in wireless networks," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.

[26] M. Yeh, C.-H. Wang, J. Lee, D.-N. Yang, and W. Liao, "Mobile proxy caching for multi-view 3d videos with adaptive view selection," *IEEE Transactions on Mobile Computing*, 2020.

[27] H. Tian, X. Xu, T. Lin, Y. Cheng, C. Qian, L. Ren, and M. Bilal, "Dima: distributed cooperative microservice caching for internet of things in edge computing by deep reinforcement learning," *World Wide Web*, pp. 1–24, 2021.

[28] M. Leconte, A. Destounis, and G. Paschos, "Traffic engineering with precomputed pathbooks," in *IEEE INFOCOM*, 2018.

[29] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, 2013, pp. 15–26.

[30] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 3–14, 2013.

[31] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[32] G. Gao, Y. Wen, and C. Westphal, "Dynamic priority-based resource provisioning for video transcoding with heterogeneous QoS," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 5, pp. 1515–1529, 2019.

[33] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman & Co., 1979.

[34] Y. Jin, Y. Wen, and C. Westphal, "Optimal transcoding and caching for adaptive streaming in media cloud: An analytical approach," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 12, pp. 1914–1925, 2015.

[35] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[36] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[37] J. Giesen and S. Laue, "Combining admm and the augmented lagrangian method for efficiently handling many constraints." in *IJCAI*, 2019, pp. 4525–4531.

[38] Z. Zhou, F. Liu, R. Zou, J. Liu, H. Xu, and H. Jin, "Carbon-aware online control of geo-distributed cloud services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 9, pp. 2506–2519, 2015.

[39] J. Su, S. Adams, and P. A. Beling, "Value-decomposition multi-agent actor-critics," *arXiv preprint arXiv:2007.12306*, 2020.

[40] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[42] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[43] M. Holzleitner, L. Gruber, J. Arjona-Medina, J. Brandstetter, and S. Hochreiter, "Convergence proof for actor-critic methods applied to ppo and rudder," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XLVIII*. Springer, 2021, pp. 105–130.

[44] K. Calvert and E. Zegura, "Gt-itm: Georgia tech internetwork topology models," 1996.

[45] "Live stream on youtube: Choose live encoder settings, bitrates, and resolutions," 2020. [Online]. Available: https://support.google.com/youtube/answer/2853702?hl=en

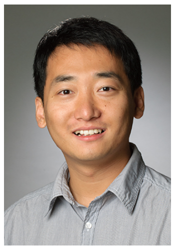
[46] J. Summers, T. Brecht, D. Eager, and B. Wong, "To chunk or not to chunk: Implications for http streaming video server performance," in *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, 2012, pp. 15–20.

[47] Z. Xu, L. Zhou, S. C.-K. Chau, W. Liang, Q. Xia, and P. Zhou, "Collaborate or separate? distributed service caching in mobile edge clouds," in *IEEE INFOCOM*, 2020, pp. 1–10.

[48] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proceedings of the tenth international conference on machine learning*, 1993, pp. 330–337.

[49] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *arXiv preprint arXiv:1706.02275*, 2017.

Yanan Gao received the B.S. degree from the School of Computer Science and Technology, Shandong University of Science and Technology, Qingdao, China, in 2015, and the M.S. degree in software engineering from Beijing Forestry University, Beijing, China, in 2020. She is currently a Ph.D. student at the School of Computer Science and Technology, Beijing Institute of Technology. Her research interests include deep learning, reinforcement learning, federated learning, and their applications to cloud/edge computing.

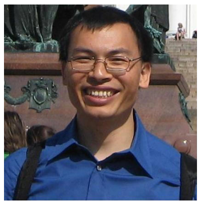

Song Yang is currently an associate professor at the School of Computer Science in Beijing Institute of Technology, China. Song Yang received the B.S. degree in software engineering and the M.S. degree in computer science from Dalian University of Technology, Dalian, Liaoning, China, in 2008 and 2010, respectively, and the Ph.D. degree from Delft University of Technology, The Netherlands, in 2015. From August 2015 to August 2017, he worked as postdoc researcher for the EU FP7 Marie Curie Actions CleanSky Project in Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG), Göttingen, Germany. His research interests focus on data communication networks, cloud/edge computing, and network function virtualization.

Fan Li received the PhD degree in computer science from the University of North Carolina at Charlotte in 2008, MEng degree in electrical engineering from the University of Delaware in 2004, MEng and BEng degrees in communications and information system from Huazhong University of Science and Technology, China in 2001 and 1998, respectively. She is currently a professor at the School of Computer Science in Beijing Institute of Technology, China. Her current research focuses on wireless networks, ad hoc and sensor networks, and mobile computing. Her papers won Best Paper Awards from IEEE MASS (2013), IEEE IPCCC (2013), ACM MobiHoc (2014), and Tsinghua Science and Technology (2015). She is a member of IEEE and ACM.

Stojan Trajanovski received his PhD degree (cum laude, 2014) from Delft University of Technology, The Netherlands and his master degree in Advanced Computer Science (with distinction, 2011) from the University of Cambridge, United Kingdom. He is currently an applied scientist in Microsoft, working in London, UK and Bellevue, WA, USA. He was in a similar role with Philips Research in Eindhoven, The Netherlands from 2016 to 2019. Before that, he spent some time as a postdoctoral researcher at the University of Amsterdam and at Delft University of Technology. He successfully participated at international science olympiads, winning a bronze medal at the International Mathematical Olympiad (IMO) in 2003. His main research interests include network science & complex networks, machine learning, game theory, and optimization algorithms.

Pan Zhou is currently a full professor and PhD advisor with Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology (HUST), Wuhan, P.R. China. He received his Ph.D. in the School of Electrical and Computer Engineering at the Georgia Institute of Technology (Georgia Tech) in 2011, Atlanta, USA. He received his B.S. degree in the Advanced Class of HUST, and a M.S. degree in the Department of Electronics and Information Engineering from HUST, Wuhan, China, in 2006 and 2008, respectively. He held honorary degree in his bachelor and merit research award of HUST in his master study. He was a senior technical member at Oracle Inc., America, during 2011 to 2013, and worked on Hadoop and distributed storage system for big data analytics at Oracle Cloud Platform. He received the “Rising Star in Science and Technology of HUST” in 2017, and the “Best Scientific Paper Award” in the 25th International Conference on Pattern Recognition (ICPR 2020). He is currently an associate editor of IEEE Transactions on Network Science and Engineering. His current research interest includes: security and privacy, big data analytics, machine learning, and information networks.

Pan Hui received his PhD from the Computer Laboratory at University of Cambridge, and both his Bachelor and MPhil degrees from the University of Hong Kong. He is a Professor of Computational Media and Arts and Director of the HKUST-DT Systems and Media Lab at the Hong Kong University of Science and Technology (HKUST). He is also the Nokia Chair in Data Science at the University of Helsinki. He has founded and chaired several IEEE/ACM conferences/workshops, and has served as track chair, senior program committee member, organizing committee member, and program committee member of numerous top conferences including ACM WWW, ACM SIGCOMM, ACM Mobisys, ACM MobiCom, ACM CoNext, IEEE Infocom, IEEE PerCom, IEEE ICNP, IEEE ICDCS, IJCAI, AAAI, and ICWSM. He served as an Associate Editor for IEEE Transactions on Mobile Computing (2014-2019) and IEEE Transactions on Cloud Computing (2014-2018), and as a guest editor for various journals including IEEE Journal on Selected Areas in Communications (JSAC), IEEE Transactions on Secure and Dependable Computing, IEEE Communications Magazine, and ACM Transactions on Multimedia Computing, Communications, and Applications. He is an International Fellow of the Royal Academy of Engineering, an IEEE Fellow, an ACM Distinguished Scientist, and a member of the Academia Europaea.


Xiaoming Fu received his Ph.D. in computer science from Tsinghua University, Beijing, China in 2000. He was then a research staff at the Technical University Berlin until joining the University of Göttingen, Germany in 2002, where he has been a professor in computer science and heading the Computer Networks Group since 2007. He has spent research visits at universities of Cambridge, Uppsala, UPMC, Columbia, UCLA, Tsinghua, Nanjing, Fudan, and PolyU of Hong Kong. Prof. Fu’s research interests include network architectures, protocols, and applications. He is currently an editorial board member of IEEE Network, IEEE Transactions on Network and Service Management, IEEE Transactions on Network Science and Engineering, IEEE Networking Letters, and Elsevier Computer Communications, and has served on the organization or program committees of leading conferences such as INFOCOM, ICNP, ICDCS, MOBICOM, MOBIHOC, CoNEXT, ICN and COSN. He is an IEEE Fellow, an IEEE Communications Society Distinguished Lecturer, an ACM Distinguished Member, a fellow of IET and member of the Academia Europaea.