MA360: MULTI-AGENT DEEP REINFORCEMENT LEARNING BASED LIVE 360-DEGREE VIDEO STREAMING ON EDGE

*Yixuan Ban, Yuanxing Zhang, Haodan Zhang, Xinggong Zhang, Zongming Guo**

Wangxuan Institute of Computer Technology, Peking University, Beijing, China
Email:{banyixuan, longo, pkuzhd, zhangxg, guozongming}@pku.edu.cn

ABSTRACT

The mobile edge caching has made video service providers deliver live 360-degree videos worldwide. However, these services still suffer from the huge network traffic on the core network due to the spherical nature and the diverse requests generated from large user populations. It is challenging to optimize the Quality of Experience (QoE) and the bandwidth consumption simultaneously under the significant number of users as well as dynamic network and playback status. In this paper, we propose a Multi-Agent deep reinforcement learning based 360-degree video streaming system, named *MA360*, to tackle this multi-user live 360-degree video streaming problem in the context of the edge cache network. Specifically, MA360 employs the Mean Field Actor-Critic (MFAC) algorithm to make clients collaboratively and distributively request tiles aiming at maximizing the overall QoE while minimizing the total bandwidth consumption. Experiments over real-world datasets show that MA360 can improve the QoE while significantly reducing the bandwidth consumption compared with several state-of-the-art edge-assisted 360-degree video streaming strategies.

***Index Terms*—** 360-degree video, live video streaming, adaptive streaming, multi-agent deep reinforcement learning

1. INTRODUCTION

360-degree videos, also known as Virtual Reality (VR) videos, have become increasingly popular in recent years. Many commercial platforms, such as YouTube and Facebook, have launched live 360-degree video streaming services.

To provide the same Quality of Experience (QoE) for users, clients require much more bitrates to deliver 360-degree videos than common 2D videos due to the spherical nature. However, only a small portion of the scene, named *viewport*, can be seen by users due to the limited screen size [1], which makes the multi-user high-quality 360-degree video delivery pretty challenging. Inspired by Dynamic Adaptive Streaming over HTTP (DASH) standard [2] applied on ordinary videos, both industry and academia have proposed *tile-based* viewport-adaptive video streaming strategies to deliver 360-degree videos efficiently [1, 3]. These strategies allocate high tile rates inside the viewport and low tile rates outside the viewport to maximize the QoE for users and alleviate the bandwidth consumption accordingly.

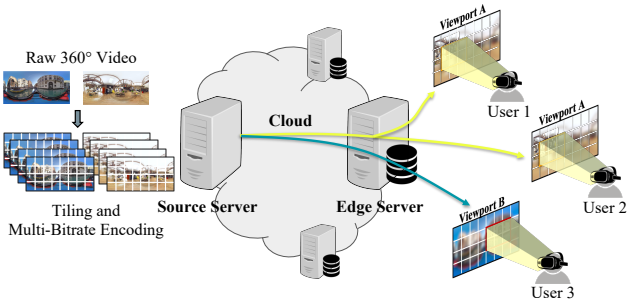


Fig. 1. Edge-assisted live 360-degree video streaming system

In addition to tile-based streaming, the edge cache network is also deployed to serve multiple users simultaneously. However, the traffic on the core network is still huge since the user requests are highly diverse in practice. In 360-degree video streaming, users may watch the same video with different viewports and bitrates, which means the edge server has to retrieve various tile representations from the source server. It will lead to considerable delivery cost for service providers and tremendous pressure for the source server. Besides, the slight quality changes in different tile representations are difficult to perceive by customers due to the non-linear relationship between bitrates and distortions. In current adaptive streaming systems, clients usually request tiles without cooperation [1, 3], i.e., the tile requests are made only by their own network conditions and playback status. It is essential and necessary to coordinate all clients' requests to alleviate traffic to the source server. Several works about multi-user adaptive streaming have proposed to deliver videos to clients cooperatively [4, 5], but all in a centralized scheme. They have to solve a sophisticated model to decide the overall bitrate allocation schemes after collecting all clients' statistics. It will lead to considerable computational complexity for edge servers as well as inevitable latency for users, and degrade the live streaming system's performance eventually.

To reduce the traffic on the core network in low computational complexity, we utilize an edge-assisted framework to

*Corresponding author. E-mail:guozongming@pku.edu.cn.

This work was supported by the National Key R&D program of China (2019YFB1802701, 2018YFB0803702), Toutiao Research Funding.

978-1-7281-1331-9/20/$31.00 ©2020 IEEE

deliver tile-based live 360-degree videos *collaboratively* and *distributively*, as shown in Fig. 1. In this framework, clients will actively adjust their tile rates to optimize both the QoE and the overall bandwidth consumption by communicating with the edge server. Ideally, if all clients with the same viewports request the same tile rates, the traffic could get significant alleviation. To get aware of the system's overall states, clients watching the same videos in our framework will report their local states to the edge server and receive the processed global states generated from the server. Then the local states and global states are combined to generate cooperative requests distributively, which can effectively reduce the streaming system's computational complexity.

Based on the proposed framework, we design a Multi-Agent deep reinforcement learning based live 360-degree video streaming system, called *MA360*, to learn the optimal collaboration streaming scheme in multi-user live 360-degree video streaming. Specifically, we utilize the Long Short-Term Memory (LSTM) network [6] to predict each user's future bandwidth and viewport to fit the dynamic network and playback conditions. Then, we leverage the Mean Field Actor-Critic (MFAC) model [7] to learn the optimal rate allocation strategy. Extensive experiments on real-world datasets demonstrate that MA360 can achieve significant improvement on overall QoE and bandwidth reduction compared with state-of-the-art streaming schemes.

To summarize, our main contributions include:

- We provide a live 360-degree video streaming framework to make clients allocate tile rates collaboratively and distributively, which can maximize the overall QoE while minimizing the total bandwidth consumption.
- We propose an MFAC-based model to solve the large-scale multi-user rate allocation problem in real-time under dynamic environments.
- We demonstrate the MA360 can outperform the existing schemes under various network and playback conditions. It can improve QoE by 14% and reduce bandwidth consumption by 52% on average.

2. MADRL BASED LIVE 360-DEGREE VIDEO STREAMING

2.1. Multi-User Live 360-Degree Video Streaming

Fig. 2 illustrates the core idea of our edge-assisted live 360-degree video streaming system. Suppose K users are watching the same 360-degree video through the same edge server with similar viewports. If they request the same tile rates, the bandwidth consumption C between the edge server and the source server can be much saved. Let U^k denote client k's QoE, where $k \in \{1...K\}$, then our optimization objective can be depicted as below with coefficient η:

$$\max \quad \sum_{k=1}^{K} U^k - \eta C. \tag{1}$$

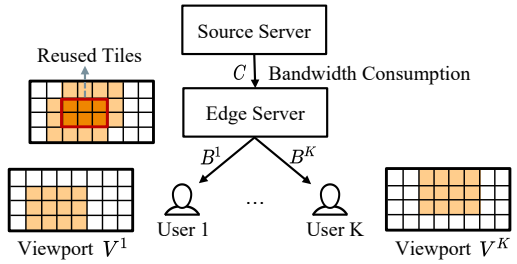


Fig. 2. Core idea on bandwidth reduction in MA360

2.2. MADRL Based Live 360-Degree Video Streaming

The edge-assisted network for tile-based adaptive 360-degree video streaming is a complex system influenced by a large number of factors, which can be seen as a high-dimensional environment for multi-user bitrate adaptation strategies. The total bandwidth consumption is calculated based on all clients' joint decisions. Traditional centralized optimization can hardly solve this problem in low complexity. Even if the system is deployed distributively, the communication overhead among large-scale clients is still high. To deal with the high-dimensional distributive collaboration problem, we introduce the Multi-Agent Deep Reinforcement Learning (MADRL) model to learn the optimal rate allocation scheme.

Formally, a typical MADRL model can be described by a Markov process denoted by a tuple $G =< K, S, A, R, P, \gamma >$ representing agent numbers, state space, joint action space, reward function, transition probability function and discount factor respectively. In each iteration of the MADRL model, *agents* make *actions* based on their current *states* and impact the *environment* simultaneously. The actions prompt the environment to yield *rewards* for agents, and then the states of all agents are changed by the transition probability function. During that, the agents can gradually learn the optimal cooperative *policy*, which can map the states to actions to maximize the cumulative discounted reward.

As shown in Fig. 3, in our framework, clients are seen as agents interacting with the playback environment jointly to maximize the utility function in Eqn. (1). To make each client allocate tile rates with the knowledge of the global circumstances, the clients are designed to report some of their *local states* s_l^k including estimated viewport V'^k, estimated bandwidth B'^k and other local information to the edge server. Then the edge server will embed the previous report of each client into a *global state* s_g, and distribute it back in real-time, instead of implementing heavy and intense computing tasks like the centralized schemes. Based on their own states s^k, where $s^k = \{s_l^k, s_g\}$, the clients can independently make requests a^k on tiles, and receive rewards r^k generated from the edge server. After thousands of interactions among s^k, a^k, and r^k, the clients can learn the optimal policy π to allocate tile rates cooperatively.

Following the adaptive streaming standard like DASH [2], we segment the 360-degree videos periodically into L chunks

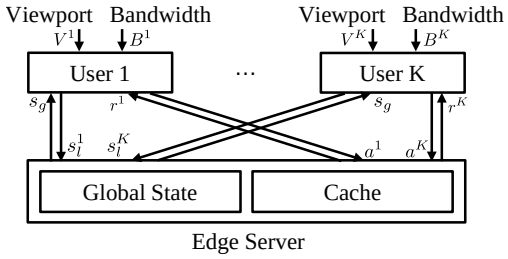


Fig. 3. Streaming framework of MA360 based on MADRL

with the same duration T, denoted as l_t, where $t \in \{1, ..., L\}$. The chunks are further cropped into N tiles with M bitrate levels, denoted as b_j, $j \in \{1, \ldots, M\}$. Let V_t^k be the k-th client's viewport on chunk l_t in euler angle, which can be mapped into the viewing tiles $\{v_{i,t}^k\}$, where $v_{i,t}^k$ indicates whether the i-th tile ($i \in \{1, \ldots, N\}$) locates in the client's viewport ($v_{i,t}^k = 1$) or not ($v_{i,t}^k = 0$). Meanwhile, let B_t^k represent the bandwidth of the k-th client while downloading chunk l_t.

To maximize the QoE, clients should decide each tile's bitrate for the chunk. We denote $x_{i,j,t}^k$ as whether the i-th tile on the t-th chunk l_t is allocated with the j-th bitrate level for client k, where $x_{i,j,t}^k = 1$ means yes vice versa.

2.3. MADRL Semantics

The multi-user live 360-degree video streaming system can be transformed into the following MADRL semantics.

State: We formulate the live streaming system into a time-slotted representation, where the source server releases new chunk every T seconds. Therefore, clients are expected to start downloading chunk l_t at time $t \times T$, as shown in "User-1" in Fig. 4. However, due to the dynamic network conditions in practice, clients may fail to download the chunk l_{t-1} entirely by the end of the slot $t-1$. Then the *remain data* should be downloaded in slot t, leading to the *asynchronous* downloading of a specific chunk and possible one-way delay, as shown in "User-2" in Fig. 4. Here, d_t^k represents the remain data in slot t of the k-th client.

To make clients aware of the dynamic changes and reduce the delay, we come up with a *timing communication* mechanism, where clients are designed to report their local states $s_{l,t}^k$ including estimated viewport V'^k_t, estimated bandwidth B'^k_t, and remain data d_t^k to the edge server by the end of the current slot. The edge server then takes the mean of the reports as the global state, i.e., $s_{g,t} = \{\overline{V'_t}, \overline{B'_t}, \overline{d_t}\}$, to depict the system's network and playback conditions and provide clues for clients to make cooperative decisions asynchronously. The simple calculation can significantly reduce the computational pressure on the edge server and can efficiently deal with the multi-user scenario in practice. Meanwhile, the client's last action a_{t-1}^k is also involved in the local state considering the temporal viewing variance's influence on QoE [3].

Within the timing communication mechanism, the clients

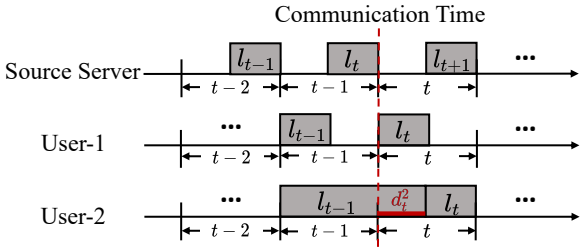


Fig. 4. Time slot-based decision process

can make decisions based on their states $s_t^k = \{s_{l,t}^k, s_{g,t}\}$. To make our framework adapt to the system's uncertainties well, we leverage the Long Short-Term Memory (LSTM) [6] model to predict each user's future viewport and bandwidth. The prediction process can be formulated as below where φ_V and φ_B are the LSTM network's parameters, $h_t(V^k)$ and $h_t(B^k)$ are the historical viewports and bandwidth traces.

$$\begin{cases} V'^k_t = \text{LSTM}(h_t(V^k); \varphi_V) \\ B'^k_t = \text{LSTM}(h_t(B^k); \varphi_B), \end{cases} \tag{2}$$

Action: The target of our streaming system is to decide each tile's bitrates, i.e., $a_t^k = \{x_{i,j,t}^k\}$. We constrain each client to select one rate level for any tile in case of the bandwidth waste:

$$\sum_{j=1}^{M} x_{i,j,t}^k \leq 1, \quad x_{i,j,t}^k \in \{0, 1\}, \quad \forall k, i, t. \tag{3}$$

Besides, to avoid from seeing a "blank" screen and ensure the smoothness playback in viewports, we constrain that tiles outside the viewport are allocated in lowest rates and the tiles inside the viewports are allocated with the same rates as below:

$$\begin{cases} \sum_{j=1}^{M} x_{i,j,t}^k b_j = b_1, & \forall k, t, v_{i,t}^k = 0 \\ \sum_{j=1}^{M} x_{i,j,t}^k b_j = \sum_{j=1}^{M} x_{i',j,t}^k b_j, & \forall k, t, v_{i,t}^k = v_{i',t}^k. \end{cases} \tag{4}$$

Given the constraints above, each client's action can be denoted as a one-hot vector a_t^k indicating which bitrate inside user k's viewport should be requested at time slot t.

Reward: As shown in Eqn. (1), MA360's main idea is to maximize the QoE and minimize the bandwidth consumption. Thus the reward of each client r_t^k is assigned as its own perceived QoE minus the average bandwidth consumption shared by each client, as shown below:

$$r_t^k = U_t^k - \eta \frac{1}{K} C_t. \tag{5}$$

In practice, the k-th client's QoE U_t^k in slot t should consider not only the average quality inside the viewport, denoted by $U_{q,t}^k$, but also the temporal viewing variance $U_{v,t}^k$ between consecutive chunks [3]. Besides, the QoE metrics should also involve the newly generated remain data to reduce delay $U_{d,t}^k$ in live streaming, as shown below:

$$U_t^k = U_{q,t}^k - \lambda U_{v,t}^k - \mu U_{d,t}^k \tag{6}$$

Thus the objective of the live 360-degree video streaming system focuses on four aspects:

Average Quality $U^k_{q,t}$: We denote the quality of the i-th tile with the j-th bitrate level on chunk t as $q_{i,j,t}$, representing the Peak Signal to Noise Ratio (PSNR) calculated by Mean Squared Error (MSE) on points. Thus the estimated perceived quality $U^k_{q,t}$ can be calculated as:

$$U^k_{q,t} = \frac{\sum_{i=1}^{N} v^k_{i,t} \sum_{j=1}^{M} x^k_{i,j,t} q_{i,j,t}}{\sum_{i=1}^{N} v^k_{i,t}}. \tag{7}$$

Temporal Viewing Variance $U^k_{v,t}$: The temporal viewing variance can be formulated as the average quality difference between the consequent chunks:

$$U^k_{v,t} = |U^k_{q,t} - U^k_{q,t-1}|. \tag{8}$$

Playback Delay $U^k_{d,t}$: The playback delay can be represented by the newly generated remain data on the reward:

$$U^k_{d,t} = [d^k_t]^+ + [\sum_{i=1}^{N} v^k_{i,t} \sum_{j=1}^{M} x^k_{i,j,t} b_j + (N - \sum_{i=1}^{N} v^k_{i,t}) b_1 - B^k_t]^+ T, \tag{9}$$

where $[x]^+ = \max\{0, x\}$ ensures the delay to be positive.

Bandwidth Consumption C_t: Due to the edge server's caching capability, multiple same requests for one tile will only forward once. Let $f_{i,j,t}$ denote whether tile-i in j-th bitrate level are requested in slot t, where $f_{i,j,t} = 1$ means it's requested and $f_{i,j,t} = 0$ otherwise. The total bandwidth consumption on slot t can be described as:

$$C_t = \sum_{i=1}^{N} \sum_{j=1}^{M} f_{i,j,t} b_j \tag{10}$$

Policy: The clients should follow the streaming *policy* $a^k_t \sim \pi_\theta(s^k_t)$ to select tile rates, where θ is the policy's parameter. Then our task can be written in MADRL semantics as:

$$\pi^*_\theta(s^k_t) = \arg\max_{\pi_\theta} Q_{\boldsymbol{\pi}}(s^k_t, \boldsymbol{a_t}) \tag{11}$$

Value: MA360 requires offline training before online fine-tuning to be efficient in real-world streaming services. During the training process, we introduce the *value* to measure whether a policy can optimize the objective of MA360. We denote the *joint state-action value function* as $Q_{\boldsymbol{\pi}}(s^k_t, \boldsymbol{a_t})$ to indicate the estimated performance of each client, which is the expectation of the cumulative discounted reward of client k on state s^k_t with all clients following joint action $\boldsymbol{\pi}$:

$$Q_{\boldsymbol{\pi}}(s^k_t, \boldsymbol{a_t}) = E_{\boldsymbol{\pi},p}[\sum_{t'=t}^{L} \gamma^{(t'-t)} r^k_t | s^k_t, \boldsymbol{a_t}] \tag{12}$$

2.4. Learning Strategy of MA360

In MA360, each client's reward is determined by the joint action $\boldsymbol{a}_t$, thus the dimension of joint state-action pairs grows proportionally $w.r.t$ the number of users K. To deal with this problem, we use the Mean-Field Actor-Critic learning infrastructure (MFAC) [7] to learn the optimal policy, which can

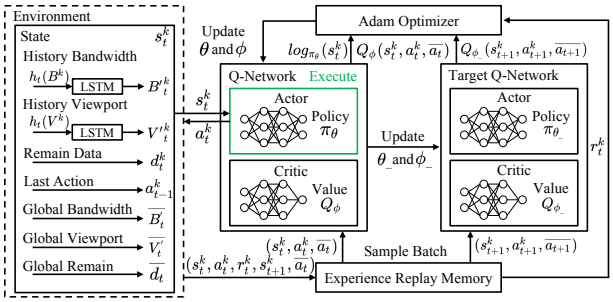


Fig. 5. MFAC-DQN based learning network

decompose the complex interaction among all clients into a simple learning process. Specifically, the k-th client only selects bitrates by cooperation with other clients' average action $\overline{a}_t = \frac{1}{K}\sum_{k=1}^{K} a^k_t$. Then the joint state-action value function for each client can be estimated by a deep learning network [8] with parameter ϕ:

$$Q_{\boldsymbol{\pi}}(s^k_t, \boldsymbol{a_t}) \approx Q_\phi(s^k_t, \boldsymbol{a_t}) = Q_\phi(s^k_t, a^k_t, \overline{a_t}) \tag{13}$$

To stabilize the learning process, we also utilize the Deep Q-Network (DQN) model [9] to update our model's parameters by leveraging the *experience replay memory* and *target Q-network*.

The structure of the MFAC-DQN based learning network is shown in Fig. 5. The model is composed of two networks, called the *Q-network* and the *target Q-network* separately. Each network consists of two sub-networks called *actor-network* and *critic-network* respectively. The actor-network is to learn the optimal cooperation policy π_θ and generate tile rate selections according to the learned policy, which is the core component. The critic-network is to evaluate the joint state-action value function Q_ϕ and guide the update of the actor-network's parameters. To break the training samples' correlation, the experience replay memory is also adopted. During each video playback process namely *episode* in MADRL semantics, the edge server will calculate the mean action $\overline{\boldsymbol{a_t}}$ and store the interaction tuple $< \boldsymbol{s_t}, \boldsymbol{a_t}, \boldsymbol{r_t}, \boldsymbol{s_{t+1}}, \overline{\boldsymbol{a_t}} >$ for all clients as *experience* in the experience replay memory. When the current episode of video playback is complete, we randomly select ε mini-batches from experience replay memory to update Q-network's parameters, where each mini-batch contains δ experiences.

To stabilize the training process and to keep the memory of the previously learned strategy, target Q-network with parameter θ_- and ϕ_- is also adopted to evaluate the joint state-action value function with Q-network together. Specifically, after each episode, target Q-network uses the moving average of Q-network's parameters computed with the learning rate τ_ϕ and τ_θ to update its own parameters :

$$\begin{cases} \phi_- = \tau_\phi \phi + (1 - \tau_\phi)\phi_- \\ \theta_- = \tau_\theta \theta + (1 - \tau_\theta)\theta_- \end{cases} \tag{14}$$

Additionally, the standard Adam Optimizer [8] is used to train the network.

Mathematically, we use the actor-critic model [3] to compute the system's cumulative reward gradient $\nabla_\theta J(\theta)$ to learn the optimal cooperation policy π_θ. It utilizes the next slot's joint state-action value function and reward to estimate the value on this slot as below:

$$\nabla_\theta J(\theta) \approx \sum_{k=1}^{K} \sum_{t=1}^{L} \nabla_\theta \log_{\pi_\theta}(s_t^k) A_\phi(s_t^k, a_t^k, \overline{a_t}) \tag{15}$$

Here, $A_\phi(s_t^k, a_t^k, \overline{a_t})$ represents the *advantage* function in critic-network to accelerate the convergence of the model, which can be calculated as:

$$A_\phi(s_t^k, a_t^k, \overline{a_t}) = y_t^k - Q_\phi(s_t^k, a_t^k, \overline{a_t}) \tag{16}$$

The y_t^k is the target value on the next slot generated by target Q-network, which can be expressed as:

$$y_t^k = r_t^k + \gamma Q_{\phi\_}(s_{t+1}^k, a_{t+1}^k, \overline{a_{t+1}}) \tag{17}$$

Then the actor-network's parameters can be updated in the direction of generating a more determinate action on that state with the learning rate α. As for the critic-network, to get a more specific evaluation on joint state-action value, we minimize the advantage function with learning rate β. The update of parameters can be formulated as below:

$$\begin{cases} \theta = \theta + \alpha \sum_{k=1}^{K} \sum_{t=1}^{L} \nabla_\theta \log_{\pi_\theta}(s_t^k) A_\phi(s_t^k, a_t^k, \overline{a_t}) \\ \phi = \phi + \beta \sum_{k=1}^{K} \sum_{t=1}^{L} \nabla_\phi (y_t^k - Q_\phi(s_t^k, a_t^k, \overline{a_t}))^2 \end{cases} \tag{18}$$

3. PERFORMANCE EVALUATION

3.1. Settings and Evaluation Methodologies

Datasets and Experimental configurations. The 360-degree video head movement dataset and bandwidth dataset are collected from [10] and [11], where the former contains 864 head movement traces with 48 users watching 18 videos each, the latter includes 86 traces scaled to ensure playback. We randomly choose 3×48 head movement traces from three videos and 20 bandwidth traces as the test set, while the others as the training set. To enlarge our training data, we always randomly choose head movement traces and bandwidth traces for clients at each training episode. Also, we normalize the viewing quality, video size and the bandwidth before training. In the experiments, we crop and encode these chunks into 4×8 tiles ($N = 32$) and 5 kinds of bitrates ($M = 5$) including {100kbps, 300kbps, 500kbps, 1000kbps, 1500kbps} with segment duration $T = 1$ second.

Hyper-parameters. In MA360's training process, the learning rate α, β is initialized by 10^{-4} and updated according to Adam Optimization [8]. Other hyper-parameters used in our framework are listed as follows:

Table 1. Hyper-parameters of MA360

η	μ	λ	γ	τ_ϕ	τ_θ	ε	δ
1	0.01	0.1	0.95	0.99	0.99	10	100

Methodology. We compare MA360 with four state-of-the-art streaming systems to evaluate the performance:

SDASH [2] acts like the Standard DASH algorithm, which delivers the entire video contents to users.

LRTile [1] leverages LR to predict the viewport and the bandwidth. It delivers tiles inside the viewports with the highest affordable rates while the others in the lowest rates.

ECache [5] utilizes a centralized framework to maximize user's QoE and minimize the bandwidth consumption by edge caching placement, which can be extended to our streaming scenario by conducting the placement process in real-time and adopting the same prediction algorithm like MA360.

Pytheas [12] adopts a single-agent reinforcement learning model to maximize the system's utility function. The prediction algorithm also sets the same as MA360.

3.2. Performance on Viewport and Bandwidth Prediction

To evaluate the performance of LSTM-based prediction, we compare it with the LR algorithm [1] and the Baseline approach, which utilizes the past three segments' information to conduct Linear Regression and average calculation separately. The prediction precision of viewport is calculated by the ratio of predicted tiles in real viewport tiles, and the precision of bandwidth is calculated by one minus the percentage of absolute error. As shown in Tab. 2, LSTM outperforms the LR and Baseline approach on both viewport prediction and bandwidth prediction, which proves its adaptations.

Table 2. Performance on Viewport and Bandwidth Prediction

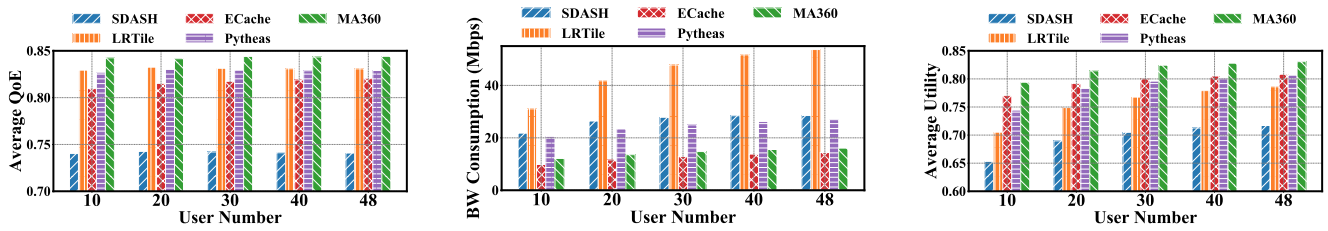
Metrics	Baseline	LR	LSTM
Viewport Prediction Precision	85.17%	86.10%	**88.14%**
Bandwidth Prediction Precision	88.87%	91.86%	**92.50%**

3.3. Performance on Different User Numbers

To check our framework's performance under different user numbers in real-world, we constrain the video index and conduct five experiments with user number equal to {10, 20, 30, 40, 48} separately. As shown in Fig. 6, no matter how many users, MA360 can effectively enhance the average normalized QoE and reduce the total bandwidth consumption, thus improve the average system utility (Eqn. (1)) on all users. As the user number grows, MA360 can further promote these metrics by extensive cooperation among clients.

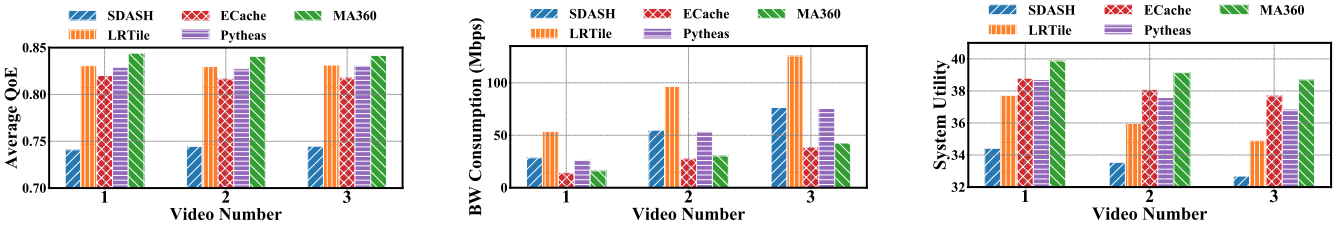
As depicted in Fig. 6(b), our model can reduce the total bandwidth consumption by 68% compared with LRTile and 41%-45% compared with Pytheas and SDASH, which greatly demonstrates our cooperation model's effectiveness. With such bandwidth consumption reduction, our model can still enhance the QoE by 14% on average compared with SDASH, while others only improve 10%-12% as shown in Fig. 6(a). Eventually, the system's average utility can get a promotion by 18% compared with SDASH in MA360, and the other schemes' improvement are 9%-14% as shown in Fig. 6(c).

The reason behind this is that SDASH, LRTile, and Pytheas act without cooperation. They only select the best tile rates for themselves, thus lead to higher bandwidth consumption and lower QoE. ECache can reduce the bandwidth consumption about the same extent as MA360 due to the cen-



(a) Average QoE (b) Bandwidth Consumption (c) Average Utility

Fig. 6. Performance on different user numbers



(a) Average QoE (b) Bandwidth Consumption (c) System Utility

Fig. 7. Performance on different live video numbers

tralized bandwidth-efficient optimization. Nevertheless, as illustrated in Fig. 6(a), the harm on QoE can not be neglected. Furthermore, the complexity of selecting reasonable tile rates for huge user populations is enormous, which makes ECache scheme impractical in real world.

3.4. Performance on Different Live Video Numbers

To evaluate MA360's performance on different live video numbers, we constrain the user number to 48 and conduct three experiments with video number equal to $\{1, 2, 3\}$ separately. As shown in Fig. 7(a), as the video number grows, the normalized QoE almost remains steady for all methods, but the download traffic grows proportionately in Fig. 7(b), which leads to the decrease of the system utility as shown in Fig. 7(c). In this case, we can still observe MA360's effectiveness on the three metrics. The cooperation strategy in MA360 can achieve the most absolute bandwidth reduction while enhancing the user's QoE, which further confirms the validity and necessity of our scheme in real-world since there could be substantial live videos broadcasting over the Internet.

4. CONCLUSION

In this paper, we present a Multi-Agent deep reinforcement learning based system, named MA360, to maximize users' QoE while minimizing the bandwidth consumption on the core network through cooperation in multi-user live 360-degree video streaming. MA360 utilizes the LSTM model to predict users' bandwidth and viewports. Besides, an MFAC-based model is also employed to enhance the system utility by generating a robust and effective rate allocation scheme for clients distributively. The framework of MA360 can easily migrate to the existing streaming systems with variable user numbers and video numbers. Extensive experiments indicate that MA360 can outperform the existing state-of-the-art methods on a variety of network conditions. Specifically, the MA360 can improve the QoE by about 14% compared with the baseline and reduce the bandwidth consumption by 41% to 68% over different methods.

5. REFERENCES

[1] F. Qian, L. Ji, B. Han, et al., "Optimizing 360 video delivery over cellular networks," *ACM SIGCOMM All Things Cellular*, pp. 1–6, 2016.

[2] T. Stockhammer, "Dynamic adaptive streaming over http: standards and design principles," *ACM MMSys*, pp. 133–144, 2011.

[3] Y. Zhang, P. Zhao, K. Bian, et al., "Drl360: 360-degree video streaming with deep reinforcement learning," *IEEE INFOCOM*, pp. 1252–1260, 2019.

[4] C. Liu, N. Kan, J. Zou, et al., "Server-side rate adaptation for multi-user 360-degree video streaming," 2018.

[5] C. Li, L. Toni, J. Zou, et al., "Qoe-driven mobile edge caching placement for adaptive video streaming," *IEEE TMM*, vol. 20, no. 4, pp. 965–984, 2018.

[6] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, pp. 1735–1780, 1997.

[7] Y. Yang, R. Luo, M. Li, et al., "Mean field multi-agent reinforcement learning," *ICML*, pp. 5567–5576, 2018.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, MIT press, 2016.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529, 2015.

[10] C. Wu, Z. Tan, Z. Wang, et al., "A dataset for exploring user behaviors in vr spherical video streaming," *ACM MMSys*, pp. 193–198, 2017.

[11] H. Riiser, P. Vigmostad, C. Griwodz, et al., "Commute path bandwidth traces from 3g networks: analysis and applications," *ACM MMSys*, pp. 114–118, 2013.

[12] J. Jiang, S. Sun, V. Sekar, et al., "Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation," *NSDI*, pp. 393–406, 2017.