QESched: QoE-aware and Energy-efficient Scheduling for Real-time Danmu Video Streaming in Wi-Fi Networks

Yibin Shen, Chi Wang, Yujin Xiang, Xinhe Li, Wenjia Wu*
School of Computer Science and Engineering, Southeast University, Nanjing, China
Email: {syibin, chiwang, yujinxiang, lixinhe, wjwu}@seu.edu.cn

***Abstract*—Danmu, also called barrage, is a newly emerging form of online video comment, which scrolls over the screen and brings better quality of experience (QoE) to users while watching videos. Therefore, it is widely adopted by various video platforms. Meanwhile, it is also found that Danmu also brings additional network energy consumption to clients in Wi-Fi networks. Although some researchers have proposed the corresponding energy-saving solutions, they will lead to serious QoE losses. To this end, we propose a QoE-aware and energy-efficient scheduling mechanism for real-time Danmu video streaming, called QESched, taking both energy saving and QoE guarantee into consideration. Firstly, we design a client-side barrage scheduling strategy to appropriately delay the barrage uploading time, so as to optimize energy consumption. Then, we design a server-side barrage scheduling strategy to decide when the barrage should be sent, where a deep reinforcement learning (DRL) model is adopted to deal with the complicated global status information. Finally, we implement the QESched system, and conduct experiments with real Danmu user traces. Experimental results show that our solution achieves a 12.8% to 26.9% reduction in terms of network energy consumption while reducing QoE loss by up to 92.6% compared to the previous work.**

***Index Terms*—Danmu, QoE, energy efficiency, deep reinforcement learning (DRL)**

I. Introduction

Danmu, also called barrage, refers to users' comments that scroll over the screen in real time while watching a video [1]. Compared with the traditional online video comments which are independent of the video, barrages can give video users a sense of real-time interaction, allowing different users to express their feelings about a certain section of the video, and all other users can synchronously see the comments at the current timestamp. The Danmu service first originates from the Japanese Danmu video-sharing website. Later, other video platforms, such as Bilibili and YouTube, also launch Danmu service in their video streaming. In addition to being applied to these on-demand videos, Danmu is also widely used in live videos. In recent years, the live broadcast business has developed rapidly. According to the 2022 Huya.tv annual big data report [2], the number of monthly active users reached 146 million, and the post-90s accounted for more than 80 percent. Compared with the on-demand video system, live broadcasting services pay more attention to the immersion, immediacy, and interaction of users. Therefore, the Danmu service has also been deployed by most live broadcast platforms, allowing users to interact with others in real time and obtain an immersive sense. Other live broadcast platforms, such as Twitch.tv, provide a live chat window for users to post their comments and subjective feelings in real time, acting as barrages.

At present, the mainstream video services mainly use transmission protocols such as dynamic adaptive streaming over HTTP (DASH), HTTP live streaming (HLS) to transmit videos in the form of chunks and deploy Danmu or real-time chat. The real-time nature of Danmu leads to a large number of small and intensive network transmission tasks, which causes it to greatly increase the network energy consumption of mobile devices under the power saving mode (PSM) proposed by 802.11 Wi-Fi network protocols [3]. In a real-time barrage video service, we find that barrages cause up to a 43.7% increase in network energy consumption. Therefore, it is necessary to optimize the energy consumption caused by the Danmu service.

Prior to our work, there have been several studies that focus on network energy consumption for smartphones and quality of experience (QoE) for barrage video. Kravets el. al [4] proposed an energy-saving algorithm that adjusted the sleep threshold by the system adaptively. Lim et. al [5] designed an energy-aware variant to reduce energy consumption with minimal impact on download latency for multi-path TCP. Yan et. al [6] and Meng et. al [7] proposed application layer based network energy consumption optimization methods to extend the sleep time of network interface card (NIC) by reasonably delaying the processing of data. As for the barrage, some studies have pointed out that the barrage can effectively improve the user's QoE when watching a video [8], [9]. Ding et. al [10] took the barrage QoE as one of the basis to optimize the quality of service (QoS) of live casting. For barrage videos, user QoE has been widely studied, and some researchers have begun to study the network energy consumption optimization method [11].

In terms of previous studies, we find that:

- Few studies on network energy consumption optimization of smartphones focus on energy-saving algorithms of real-time barrage video. Most of them are proposed for

*Corresponding author: Dr. Wenjia Wu

traditional transmission protocols, such as TCP and UDP. These algorithms are not suitable for the scenario of real-time barrage videos because of the high-frequency and bi-directional communication between the server and the clients.
- Jiang et. al [11] first carry out the research on the optimization of network energy consumption for Danmu video services. However, in the proposed system, the scheduling algorithm is relatively simple, and additional edge servers are needed to conduct the scheduling. It is difficult to cope with the complex and dynamic barrage scheduling scenarios which result in severe loss in barrage QoE.
- For barrage QoE, there is a lack of quantitative barrage QoE models and the corresponding applications in real scenarios.

Therefore in this paper, we propose a QoE-aware and energy-efficient scheduling mechanism for real-time Danmu video streaming in Wi-Fi networks, called QESched. Our contributions can be concluded as follows.

- We present an overall framework for QoE-aware and energy-efficient barrage scheduling that considers both energy saving and QoE guarantee, called QESched. Moreover, this framework provides client-side scheduling function and server-side scheduling function to optimize the energy consumption of barrage uploading and barrage receiving, respectively.
- We design a client-side barrage scheduling strategy to determine the barrage upload time. On the basis of the prediction of the next chunk request time, the barrage upload time can be appropriately delayed to be consistent with the chunk request, so as to achieve the purpose of energy saving.
- We design a server-side barrage scheduling strategy to decide whether the barrage should be sent immediately to clients or wait for the next decision, where a deep reinforcement learning (DRL) model is adopted to deal with the complicated global status information including DASH-related status, barrage transmission status, and QoE loss, so as to reduce the energy consumption of barrage receiving at clients. For assessing the QoE loss, we also propose a quantitative model for QoE loss consisting of barrage delay, barrage intensity, and display order.
- We implement the QESched system, and conduct experiments with real Danmu user traces in the Wi-Fi network. Experimental results show that our solution achieves a 12.8% to 26.9% reduction in terms of network energy consumption while reducing QoE loss by up to 92.6% compared to the previous work.

II. Background and Related Work

In this section, we survey the development of DASH video streaming and DRL technologies, the power-saving mode of current Wi-Fi protocols, and existing network energy consumption optimization mechanisms.

A. DASH Video Streaming

DASH is a video streaming transport protocol based on HTTP protocol. The DASH protocol can be used together with barrage. In DASH video streaming, the video is decomposed into a series of chunks with short lengths and different bit rates, and clients request chunks with the appropriate resolution based on the deployed adaptive bit rate algorithm (ABR). Due to the wide use of DASH, more and more related research has emerged, including the research on ABR in pursuit of higher QoE [12], [13] or energy efficiency [14], and low-latency live in DASH broadcast [15], [16].

B. Deep Reinforcement Learning

Traditional decision-making algorithms, such as decision trees, have a static environment in the decision-making process, and cannot cope with complex and unstable environments. To solve this problem, DRL algorithms have emerged. In recent years, DRL has achieved success in many fields, such as Go-Explore and robots [17]. Many researchers have begun to use DRL techniques to solve optimization problems or decision-making problems. In reference [18], a DRL-based shared computing clusters resource scheduling method considering time-varying characteristics was proposed, and improved metrics for cluster operational excellence remarkably. In reference [19], an online computation offloading algorithm based on deep deterministic policy gradient (DDPG) in DRL, called EDDPG, was given to solve the problem of energy consumption minimization. In this paper, we also employ a DRL model to make decisions for server-side barrage scheduling, so as to save energy.

C. PSM in Wi-Fi Network

To reduce the NIC energy consumption of the mobile device, the IEEE 802 standards committee designs the corresponding energy consumption management mechanism in the 802.11 protocol clusters, and introduces the energy-saving mode PSM.

In general, there are four states of a Wi-Fi NIC:

- Sleep: The NIC turns off the sending and receiving modules to sleep, so that the energy consumption is minimized.
- Rx Idle: The NIC listens to the Wi-Fi channel but does not actually receive data.
- Rx: The NIC listens to the data frame and receives it.
- Tx: The NIC sends a data frame.

Among the above four states, only Rx and Tx states are truly active, and the energy consumption of this part is also the highest. On the contrary, the sleep state has the lowest energy consumption. However, in practice, in order to avoid additional overhead caused by trivial switching states and ensure that there is no conflict between receiving and sending, the NIC needs to stay in Rx Idle state for a certain time interval called idle threshold t_i whose energy consumption is much higher than sleep mode. Only if the time interval between two network transmissions is greater than the threshold t_i, can the NIC switch to sleep state. So if we can reduce the total time

NIC stays in idle state, the network energy consumption can be effectively reduced [20].

D. Network Energy Consumption Optimization

At present, there have been multiple researches on network energy consumption optimization of smartphones. Here we present a few of the main relevant researches. Meng et.al [7] proposed a mechanism to apply a dynamic data sending and receiving policy on the client side. The authors mentioned the use of a cached data domain to extend the time when the cell phone network card is in sleep mode by delaying the processing of relevant data. For another, Jiang et. al [11] first studied the impact of Danmu on Wi-Fi network energy consumption, and proposed a heuristic QoS-aware optimization algorithm based on the transport layer. In the proposed system, an edge proxy and client proxies were deployed to monitor the DASH-related transmission. And barrages were delayed within the allowed time to achieve synchronous transmission with other DASH-related packets, so as to prolong the NIC sleep time.

III. Problem Statement and Formulation

In this section, we analyze the network energy consumption by Danmu and the impacts of the barrage scheduling on QoE. For another, we propose relevant formulas to ensure the reliability of the scheduling algorithm proposed in section IV.

A. Network Energy Consumption by Danmu

The Wi-Fi NIC's energy consumption of mobile devices can be expressed by the following formula when PSM is enabled:

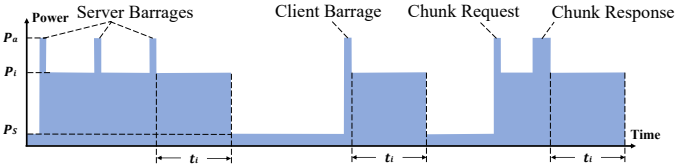
$$W = P_a \times T_a + P_i \times T_i + P_s \times T_s \tag{1}$$

where P_a, P_i, and P_s respectively represent the power of the Wi-Fi NIC in active, idle, and sleeping states; T_a, T_i, and T_s respectively represent the time that NIC stays in the three states.

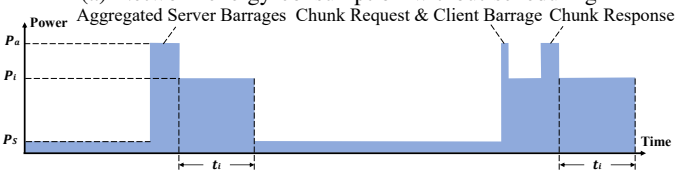
We consider the ideal network environment with no packet loss, constant bandwidth, and constant round trip time (RTT). When the same video is played each time, the DASH-related network transmission time is the same, that is, T_a is constant, and the sum of T_a, T_i, T_s is equal to the video length T. Thus, the formula can be simplified as follows:

$$\begin{aligned} W &= (P_s - P_i) \times T_s + W' \\ W' &= (T - T_a) \times P_i + P_a \times T_a \end{aligned} \tag{2}$$

Since $P_s < P_i$, the NIC energy consumption is monotonically decreasing with respect to the length of sleep time [11]. The use of interactive Danmu service will lead to the non-synchronous transmission of a large number of barrage-related packets, which makes the network module switches between active and idle states frequently, and significantly shortens the sleeping time as well, thus increasing the network energy consumption. As to a real-time barrage video system, each barrage is generated by the client and uploaded to the server, and then sent by the server to other clients watching at the same time. Therefore, for any mobile device, the additional network energy consumption caused by Danmu consists of two parts: uploading the barrages generated by the local clients to the server, and receiving the barrages generated by other clients from the server. In the remainder of this paper, we use client barrage, and server barrage to refer to each of these two types of barrages.



(a) Network energy consumption without scheduling



(b) Network energy consumption with scheduling

Fig. 1: Network energy consumption of client by Danmu without/with barrage scheduling

In Fig. 1, it can be easily found that there are two basic barrage scheduling ideas that can be used to increase the sleep time of the NIC, i.e., synchronously transmitting barrages and DASH-related packets (e.g., chunk request & client barrage in Fig. 1b), and synchronously transmitting multiple barrages (e.g., aggregated server barrages in Fig. 1b).

Therefore, we design a barrage scheduling algorithm with dynamic parameter adjustment on the client and a DRL-based barrage scheduling algorithm that allows synchronous transmission between barrages in the server. We specially add QoE loss as one of the decision factors, so as to reduce energy consumption as much as possible under the premise of minimizing QoE loss.

B. QoE Measurement

Barrage can be regarded as a special form of online video comment, where users can communicate and comment in real time while watching videos. An increasing number of studies have pointed out that barrages can improve the users' QoE while watching videos [8], [9], [21]. Barrage is different from ordinary online video comments in the following aspects:

- Different emphasis on content: The language used in barrages often has the characteristics of conciseness and humor, and it's more about the emotional expression of the audience in a fleeting time [22].
- Special display form: the barrage is usually displayed at the top of the video, entering from the right side of the screen and exiting from the left side. As a result, it will cover the video content, which is just the reason why some people don't like the Danmu video.
- High real-time property: the content of barrages is highly coupled with video content [23], and there is a high intensity of real-time communication between users.
- High interactivity: the Danmu users send barrages to spread their views and emotions, and communicate with

other Danmu users to obtain information, entertainment, and companionship.

In an ideal environment, playing the same video and the same Danmu each time will bring the same amount of QoE improvement. However, applying the energy-saving scheduling of barrages will lead to QoE loss on the basis of this benchmark improvement. We hope to minimize the reduction of this part of QoE. Based on these conclusions, combined with the actual experience, we put forward the quantitative formula of the barrage QoE loss from the following aspects.

Barrage delay: video content almost completely determines the feelings of Danmu users. Therefore, barrages and video content are highly correlated and synchronized in real time [24]. Therefore, the timelines of barrages should match with the timelines of the corresponding videos. Meanwhile, those who watch the video at the same time are more willing to see the barrages display synchronized with the video progress, so as to gain higher QoE. The long-time delay can lead to untimely interactions and get barrages out of sync with the content of the video and reduces QoE to a certain extent.

(a) Without scheduling (b) With scheduling

Fig. 2: Impact on video content display (without/with barrage scheduling)

Barrage intensity: some previous studies show that many users are sensitive to the intensity of barrage [10], because dense barrages will cover the video and decline the viewing experience. Moreover, the increase in barrage intensity will make it more difficult for users to read barrages completely, which will bring a bad interactive experience. And our energy-saving system will make multiple barrages aligned and appear on users' screens simultaneously. The comparison between before and after the barrage scheduling is shown in Fig. 2, and it can be easily told that with the scheduling, the barrages are displayed intensively, which covers the video content and increases the reading difficulty.

In practical use, we conclude that the QoE loss of this part can be calculated separately at each time when there are n barrages displayed simultaneously, and finally summed up. At the same time, the impact brought by the change of barrage frequency has an obvious diminishing marginal effect, so we use the integral of the standard normal distribution function to calculate the reduction of QoE. As a consequence, this part of QoE loss can be calculated as:

$$f(n) = \int_0^{n-1} \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} dz \tag{3}$$

Display order of barrages: It is mentioned that Danmu service has high interactivity. Spreading simple ideas, emotions and providing quick and convenient interaction between users is one of the important functions of the screen. Therefore, the display order of barrages on different clients should be strictly matched according to their generation time, otherwise, QoE will also be reduced.

The QoE loss formula is shown as below:

$$QoE_l = \gamma \times delay + \lambda \times \sum f_i(n) + \mu \times M \tag{4}$$

where $delay$ denotes the barrage delay, M denotes the incorrect number of barrage display order. Finally, these three parts are added by different weight. By default, we set γ, λ, and μ to 1, and it is also the ratio we use in the QoE evaluation in section V.

IV. QESched System

In this section, we first present the overview of our proposed system called QESched, and then propose the client-side barrage scheduling algorithm and server-side barrage scheduling algorithm to optimize the energy consumption of barrage uploading and barrage receiving, respectively.

A. System Overview

As shown in Fig. 3, the QESched system consists of two parts: the client-side barrage scheduling and the server-side barrage scheduling, respectively corresponding to optimizing the additional network energy consumption generated by uploading and receiving barrages.

As for clients, we propose a heuristic barrage scheduling algorithm, which can adaptively adjust parameters. In the scheduling process, we delay some of the barrages and monitor the transmission of DASH-related packets and barrages in order to synchronize the transmission of client barrages with chunk requests, DASH chunks, and server barrages.

The server-side barrage scheduling solution contains the processes of perception, decision and execution.

- Perception layer: once the user starts to use the proposed scheduling system, the server will delay the newly uploaded barrage and wait for the scheduling result. The system starts to update the user's network transmission information and QoE preference in real time to generate the latest user status information.
- Decision layer: The decision layer is composed of a DRL agent. The agent obtains the user's status information from the perception layer and extracts the input of the neural network (NN) model. The NN model outputs the probabilities of taking each action. Finally, the agent selects an action randomly according to the probabilities as the decision result.
- Execution layer: The execution layer has two main tasks. Firstly, implement DASH live video service, which can be simply described as sending the corresponding DASH chunks to clients according to chunk requests. Secondly, send the delayed barrages or continue to delay the transmission of these barrages according to the decision result.

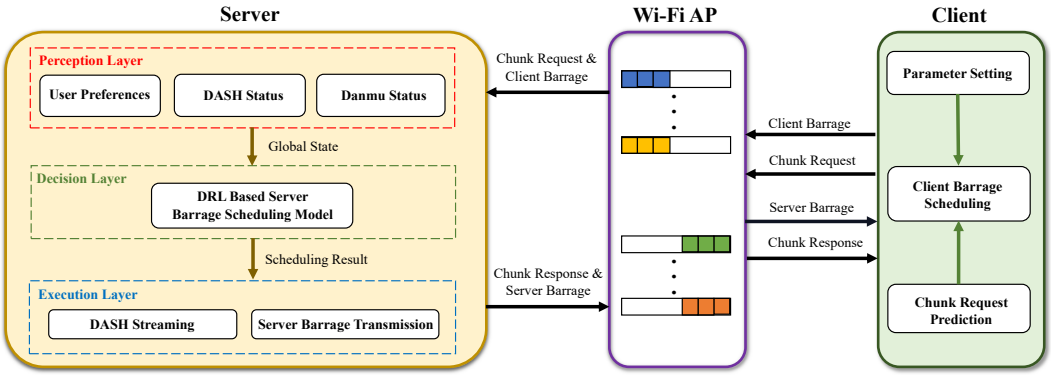


Fig. 3: Framework of QESched

B. Client-side Scheduling

For the client-side barrage scheduling, we hope to reduce network energy consumption by delaying the barrages to synchronize with other video transmission tasks. However, too long delay not only cannot further optimize energy consumption, but also brings severe QoE loss. Therefore, we add some limits to avoid excessive delay. First, at the start of the video streaming, according to the user's preference, the client sets the maximum allowed delay time D for the transmission of a barrage, which means when the barrages must be sent to the server once the delay reaches D. After that, when a new barrage is generated, the buffer of the DASH client is used to predict the request time of the next chunk based on the client's chunk request logic. In the DASH open source fore-end, polling is done every 0.5 seconds, and the next chunk will be requested if the buffer is less than a value set in advance. The following formula represents how to predict request time.

$$t = \begin{cases} \lfloor (buf - buf_l)/0.5 \rfloor \times 0.5, & buf \geq buf_l \\ 0, & buf < buf_l \end{cases} \tag{5}$$

where buf denotes the current buffer of DASH player, and buf_l denotes the lowest buffer needed to keep. Finally, the client-side barrage scheduling is executed. There are four sending types of barrages.

- Type 1: If $t > c \times D$, we send the barrage immediately to avoid unnecessary delay.
- Type 2: If there is another video transmission at the current time, we send the barrage directly.
- Type 3: If $t < c \times D$, we delay the barrage and send it once the delay reaches D.
- Type 4: If $t < c \times D$, we delay the barrage and send it once a new chunk is requested.

Here c is a constant calculated in advance. This algorithm is used to ensure that the barrage could be sent at the same time as the chunk request after being delayed, reducing the possibility of delay reaching D and guaranteeing that there is no obvious loss in the barrage QoE while minimizing the energy consumption. Algorithm 1 shows the pseudocode for client-side scheduling.

We generate the barrages on the client according to the prepared Danmu traces and the numbers of barrages of the four sending types are denoted by n_1, n_2, n_3, and n_4 respectively. First, we fixed c for a preliminary experiment. According to the results of the preliminary experiment, it works best when c ranges from 0.4 to 1, and as c increases, n_3 and n_4 also increase. And our goal is to increase the amount of n_4 while minimizing n_3. The following formula represents how to self-adaptive adjust c.

Algorithm 1: Client-side barrage scheduling

```
c = 0.8 // Initialization of parameter c
index = [] // List of delayed barrages
barrages = [] // List of barrages generated at client
n = 0 // Initialization of Number of barrages
// Called whenever a new barrage b_n is generated
Function Schedule(b_n):
    barrages.add(b_n)
    n++
    predict the time t of next chunk request
    if t < c × D or DASH packets are transmitting
     then
        send b_n
        adjust the value of c
    end
    else
        index.add(b_n)
    end
end
// Called whenever the delay of barrage b_i reaches D
Function OnMaxDelay(i):
    send b_i
    index.delete(b_i)
    adjust the value of c
end
// Called whenever a new chunk is requested
Function OnChunkRequest:
    for j in range(index.size) do
        send index[j]
        adjust the value of c
    end
    index = []
end
```

$$f(c, type) = \begin{cases} c, & type = 1 \\ max(0.4, c - n_2 * D/(N * T_c)), & type = 2 \\ max(0.4, c - n_3 * D/(N * T_c)), & type = 3 \\ min(1, c + n_4 * D/(N * T_c)), & type = 4 \end{cases} \tag{6}$$

where T_c is the length of chunks, and N is the number of barrages sent.

TABLE I: Ratio of four barrage's sending types

c	Type 1	Type 2	Type 3	Type 4
1	23.42%	42.30%	19.61%	14.67%
0.6	23.35%	51.55%	10.86%	14.24%
0.2	23.51%	63.46%	2.40%	10.63%
f(c,type)	23.34%	56.06%	6.37%	14.23%

We conducted 12 experiments for different values of c, and Table I shows the average experimental results. In the table, it could be calculated that the proportion of sending type 4 increases by 3.60% compared to the method that c equals 0.2, and the number of sending type 3 decreases by 13.24% in the condition that the proportion of sending type 4 is nearly identical compared to the method that c equals 1. This algorithm works effectively.

C. Server-side Scheduling

In this section, we propose a server-side barrage scheduling algorithm that involves the design and training method of the DRL model.

1) Global Status Information Acquisition: In the perception layer, the main task is to record the user status information, and provide input for the decision layer. For users who enable energy-saving barrage scheduling, we will firstly initialize users' experience preferences, in order to select DRL models trained with different parameters, which make different decisions on barrage scheduling according to personalized preferences. Secondly, Danmu server will maintain a barrage list, which stores some properties including generation time on clients and the waiting time caused by client-side scheduling of all the barrages that have been received from other clients but not sent to the user. Every time server receives a barrage, it is added to the list. Finally, we will update the latest DASH-related transmission and barrage transmission time of different clients in real-time.

2) Markov Decision Process Formulation: In the following, we present the design of Markov decision process.

- $State$: $State$ outlines the values obtained by the agent from the perception layer and becomes the input of the DRL model. In the case of energy-saving barrage scheduling, we define the $State$ as:

$$\begin{aligned} State &= (L, K, T_n, T_r, T_b, T_d) \\ L &= \{l_1, l_2, ..., l_k\} \end{aligned} \tag{7}$$

  where L represents an array composed of the generation time l_i of each delayed barrage, K represents the number of delayed barrages, T_n represents the present time, T_r represents the time of the client's latest chunk request, T_b represents the time of the client's latest barrage transmission, and T_d represents the sum of the delayed barrages' waiting time caused by client-side barrage scheduling.
  Due to the large change in the time scale of the training process, we need to standardize the time-related data. However, ordinary normalization will lead to the wrong estimation of energy consumption and QoE in the real-world environment, resulting in incorrect scheduling decisions. To address this problem, we standardize by converting time to relative time, that is, subtracting all the time values in the $State$ from the minimum value in L. Therefore, the final form of $State$ can be described as:

$$l_{min} = \min(l_1, l_2, l_3, ..., l_k) \tag{8}$$

$$\begin{aligned} State = (&\{l_1 - l_{min}, l_2 - l_{min}, ..., l_k - l_{min}\}, K, \\ &T_n - l_{min}, T_r - l_{min}, T_b - l_{min}, T_d - l_{min}) \end{aligned} \tag{9}$$

- $Action$: For each step, there are two possible actions: send the barrages in the waiting list together and clear the list or keep these barrages and wait for the next decision. Thus, the $action$ of the DRL algorithm is defined as $(0, 1)$, where 0 means to retain the delayed barrages, 1 means to send all the barrages in the list.
- $Reward$: The reward function is cleverly designed to guide the agent to acquire the optimal policy. We hope to reduce network energy consumption as much as possible while pursuing less QoE loss, so $reward$ should be composed of two parts: network energy consumption function and QoE function QoE_l. It is difficult to calculate energy consumption directly, but as we mentioned in section III, energy consumption is a monotonically decreasing function of NIC sleep time T_s, so we use the sleep time of the NIC to replace the energy consumption function. So the $reward$ for each action is shown as followed:

$$reward = \alpha \times T_s + \beta \times QoE_l \tag{10}$$

  However, in practical training, we find that if $reward$ is calculated in this way for every step, the reward value will be very close no matter whether the $action$ is 0 or 1, resulting in poor algorithm performance. Therefore, we adapt the idea of reward shaping [25] in dealing with sparse rewards. Only when the value of $action$ is 1, the $reward$ is calculated according to this formula. Otherwise, the $reward$ is a default number R_d which is close to 0, such as ± 0.01. So we reshape the reward function as:

$$reward = \begin{cases} \alpha \times T_s + \beta \times QoE_l, & action = 1 \\ R_d, & action = 0 \end{cases} \tag{11}$$

3) DRL Model: In the proposed MDP model, the generation time array of the delayed barrage L in the $State$ has different dimensions from other properties, so we cannot extract the status information by simply using a full connection layer.

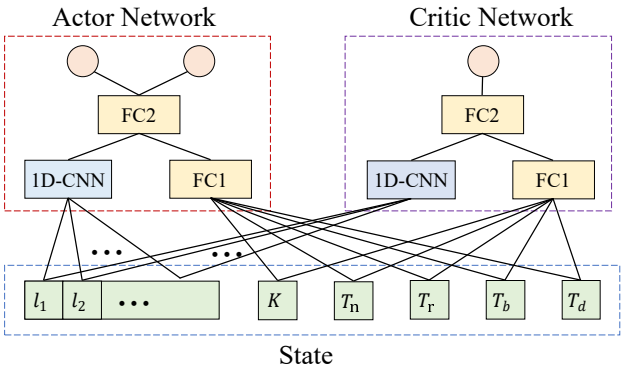


Fig. 4: Actor-Critic algorithm to generate scheduling policies

In order this end, we first fill 0 in the tail of L to a fixed length, and pass it to a $1D$ convolution layer [12] with 64 filters each of size 4 with stride 1. The remaining properties are collectively used as inputs to another fully connected layer. Then the outputs of these two layers are aggregated as inputs to the next fully connected layer. Finally, the probability of actions is the output applying the softmax function. The critic network uses the same NN structure, but its final output is a linear neuron. Fig. 4 pictures the NN model in detail.

4) Model Training: Ideally, we will train the DRL model in the realistic network using the built real-time Danmu video system introduced in section V. However, DRL training needs a large number of datasets, and it results in a significant time cost for training in a realistic environment. Therefore, we build a simulation environment to imitate the process from receiving barrages from one client to uploading the barrages to other clients. We watch the video for some time in a realistic environment, and record the time when the client requests DASH chunks. Then we select 100 Danmu traces from the Danmu API provided by Bilibili [26], and regard the time property of barrages provided as the time that the barrage is generated at the client. Finally, we randomly generate the client-side waiting time for each barrage according to the probability given by section IV-B, that is, 79.40% of the barrages has no delay in clients, 14.23% of the barrages' waiting time is a random number range from 0 to D, and the rest of barrages' waiting time equals to D.

We initialize the simulation environment using the data prepared in advance, and train our NN model with PPO-CLIP algorithm. We also adapt some useful tricks including adding the action probability entropy to the loss function, learning rate decay and using tanh activation function in particular layers [27] to improve the performance of the algorithm. Algorithm 2 describes the training method of the DRL model. In Algorithm 2, the $agent$ extracts status information and makes decisions, the $experience\_memory$ and $reward\_list$ store observation data and reward value for each step respectively.

5) Server-side Barrage Scheduling: Whenever the perception layer detects a new barrage or a new chunk request arriving, the algorithm will make a scheduling decision. If it's a new barrage, add the newly received barrage to the delayed-barrage list and wait for the scheduling decision. Otherwise, update the DASH-related transmission status information. The

Algorithm 2: Server-side Model Training

```
begin
initialize environment, agent, experience_memory,
  and reward_list
initialize n_episodes and target_value
// S_n and r_n denote the state and reward of each step
for i in range(n_episodes) do
    reset environment
    R = 0 // total reward per episode
    n = 0
    while not done do
        action = agent.get_action(S_n)
        execute action in environment
        get r_n, S_{n+1}
        R = R + r_n
        experience_memory.append((S_n, action, r_n,
          S_{n+1}))
        if experience_memory is full then
            agent.learn(experience_memory)
            experience_memory.clear()
        end
        reward_list.append(R)
        if i % 20 == 0 and
        average(reward_list) ≥ target_value then
            break
        end
        n = n + 1
    end
end
close environment
```

agent extracts the input for the DRL model based on the status information provided by the perception layer, outputs the probability value of two actions, and selects actions according to the probabilities. If $action$=1, the execution layer sends the delayed barrages to the client synchronously, and clears the list of delayed barrages. If $action$=0, no barrages will be sent, they will wait for the next scheduling. This operation ensures that the staggered transmission of the server barrages on the original time scale becomes synchronous, which reduces network energy consumption. Algorithm 3 shows the server-side scheduling process in detail.

V. Evaluation

In this section, we implement the QESched system, and conduct real-world experiments to evaluate the performance of our system.

A. Experimental Setup

1) Real-time Barrage Video Service: We use the open-source fore-end provided by dash.js to build video websites. The backend based on Flask is deployed on the remote server, and the WebSocket protocol is used to realize the real-time interactive barrage service. We select the appropriate barrage frequency according to the barrage data published by Huya.tv, and download six real Danmu traces from Bilibili according to the set frequency as test data, that is, three for client

Algorithm 3: Server-side barrage scheduling

```
barrage_list consists of delayed barrages
begin
initialize agent
//Called whenever server just receives client's barrage
Function OnBarrageArriving():
    barrage_list.append(barrage)
    get latest information of DASH request
    extract State
    action = agent.get_action(State)
    if action == 1 then
        send delayed barrages
        barrage_list.clear()
    end
end
//Called whenever server just receives chunk request
Function OnRequestArriving():
    get barrage_list
    update DASH transmission status information
    extract State
    action = agent.get_action(State)
    if action == 1 then
        send delayed barrages
        barrage_list.clear()
    end
end
```

barrage frequency comparison: Low (average of 0.2 client barrages/second and 5.6 server barrages/second), Mid (0.5 and 5.6), and High (1.0 and 5.6); three for server barrage frequency comparison: Low (0.5 and 3.6), Mid (0.5 and 5.6), and High (0.5 and 7.5). We select Mid frequency and 8 second chunk length as default settings.

2) Energy Measurement: We use two smartphones to visit our own video website at the same time and measure the energy consumption of one of the phones. We randomly split the downloaded Danmu traces into two according to the frequency of the client barrages and the server barrages. The test phone generates barrages and uploads them according to the client Danmu traces. The other generates barrages according to the server Danmu traces, which will be relayed to the test phone by the server. The network energy consumption is obtained by the difference between the total energy consumption of watching barrage videos and the energy consumption of watching local videos with the Wi-Fi module off. We use Redmi Note 4X as the test phone and Monsoon power monitor AAA10F [28] to measure energy consumption.

3) Evaluated Method: In the QESched system, we provide three stable DRL decision models, called QESched-A, QESched-B, and QESched-C, which respectively satisfy different user preferences, and the corresponding hyper-parameter settings are shown in Table II. Specifically, QESched-A is the model adopted as default setting, QESched-B is the enhanced energy-saving version, and QESched-C is a version of sacrificing certain energy-saving effects to obtain less QoE loss. We measure the network energy consumption of the solution without barrage scheduling as the benchmark called Original. Meanwhile, we implement the existing barrage scheduling algorithm called QoS-aware [11] as the comparison scheme. We set the QoS-aware algorithm with the maximum allowed delay D = 2, due to the setting can best juggle energy consumption and barrage QoE.

TABLE II: Hyper-parameter setting of different models

Model Name	R_d	α	β	γ	λ	μ
QESched-A	-0.01	1	$\frac{1}{240}$	1	1	1
QESched-B	0.01	1.5	$\frac{1}{240}$	1	1	1
QESched-C	-0.01	1	$\frac{1}{180}$	1.5	1	1

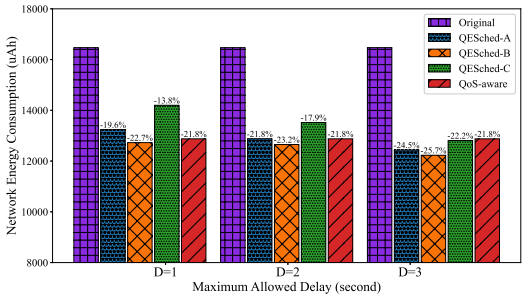
B. Experimental Results

1) Impacts of Maximum Allowed Delay Time D: In this part of the experiments, we set the maximum allowed delay time D to 1 second, 2 seconds, and 3 seconds respectively. Meanwhile, the barrage frequency and the chunk length are set to default values.

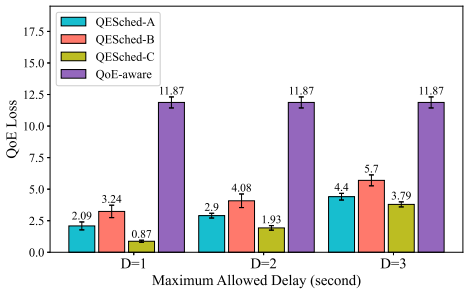
The results are shown in Fig. 5. In this figure, we can see that the parameter D has a significant improvement in the effect of energy consumption optimization. When D reaches 3 seconds, QESched can reduce network energy consumption by up to 25.7%. In addition, in the QESched-A and QESched-B systems, the performance on energy saving is better than the QoS-aware algorithm. Moreover, the energy-saving effect of the low-latency model QESched-C is also greater than that of the QoS-aware algorithm when D is set to 3 seconds. In terms of barrage QoE, our models show excellent performance. All three models we provided are substantially better than the comparison group, with up to 92.6% reduction in barrage QoE loss. Even in the case of using the QESched-B and with D is set to 3 seconds which sacrifices the most QoE to achieve a higher energy reduction, it still shows a 51.9% improvement than the QoS-aware algorithm. In the subsequent experiments, we choose 2 seconds as the default maximum allowed delay time.

2) Impacts of Barrage Frequency: In this part of the experiments, we set client/server barrage frequency to Low, Mid and High, respectively. Meanwhile, other parameters are set to default values.

The results are shown in Fig. 6. Among the three methods we offer, the network energy consumption is reduced by 12.8% to 26.9%. And both of them reduce significantly more energy as barrage frequency increases. It also reduces QoE loss by 58.1% to 84.2% on the premise of achieving no less than or even better energy savings compared to the comparison group. It is noteworthy that when the frequency of client barrages decreases our energy-saving effect is better as shown in the comparison of each density in Fig. 6a. Also, when the frequency of server barrages increases our energy-saving effect is better as shown in Fig. 6c. To sum up, the greater the difference between the frequency of client barrages and server barrages, the better the effect of our energy optimization,

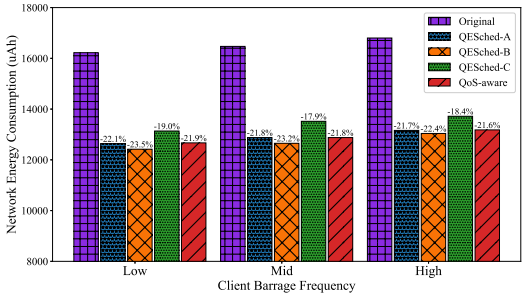


(a)

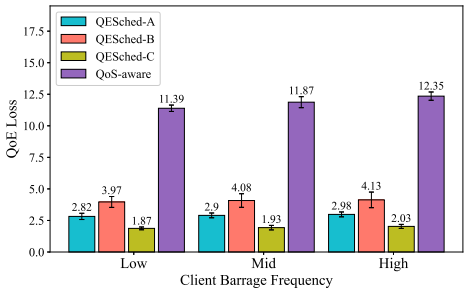


(b)

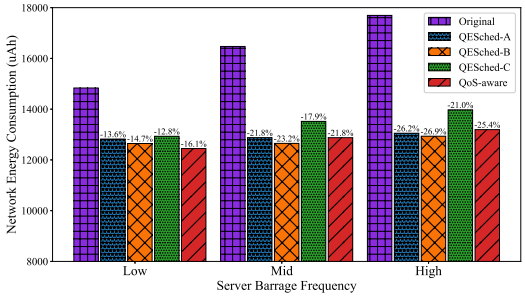
Fig. 5: Impacts of parameter D on network energy consumption and QoE loss



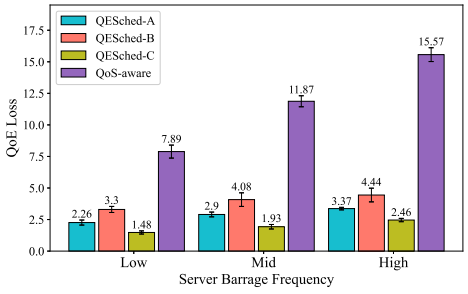
(a)



(b)

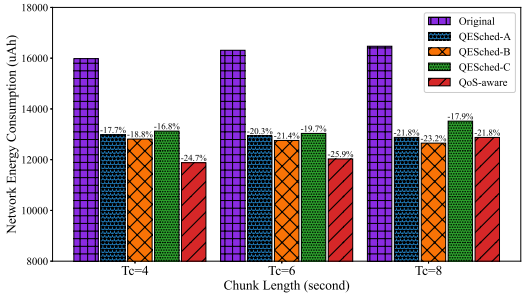


(c)

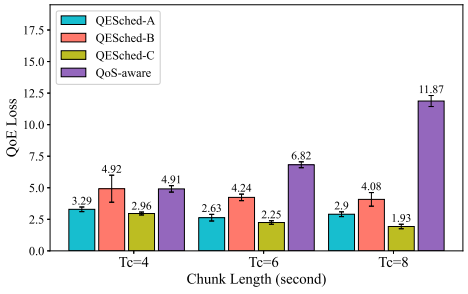


(d)

Fig. 6: Impacts of barrage frequency on network energy consumption and QoE loss



(a)



(b)

Fig. 7: Impacts of chunk length on network energy consumption and QoE loss

which is very much in line with the actual application scenario where the barrage frequency of the server-side is much greater than that of the client-side.

3) Impacts of Chunk Length: In this part of the experiments, we set Chunk Length to 4 seconds, 6 seconds and 8 seconds, respectively. Meanwhile, other parameters are set to default values.

The results are shown in Fig. 7. Our proposed QESched performs better with longer chunk length. With chunk length of 8s, we achieve a 17.9% to 23.2% reduction in network energy consumption, and a 65.6% to 83.7% reduction in QoE loss. With chunk length of 6s, we achieve a 19.7% to 21.4% reduction in network energy consumption, and a 37.8% to 67.0% reduction in QoE loss. With chunk length of 4s, we achieve a 16.8% to 17.7% reduction in network energy consumption, and a 32.3% to 39.7% reduction in QoE loss. Only in the QESched-B system with T_c=4, our method suffers slightly more QoE loss than the QoS-aware algorithm.

VI. Conclusion

In this paper, we investigate the barrage scheduling problem for real-time Danmu video streaming in Wi-Fi networks, which aims to optimize client's network energy consumption while guaranteeing user's QoE. Firstly, we define a quantitative model for QoE loss. On this basis, we propose a QoE-aware and energy-efficient scheduling mechanism called QESched that combines a client-side scheduling algorithm and a server-side DRL-based scheduling algorithm to reduce energy consumption of both barrages uploading and barrages receiving on the client. Finally, we implement the corresponding system and conduct real-world experiments. The results indicate that our system can reduce network energy consumption by 16.8% to 26.9%. Moreover, our work avoids up to 92.6% loss in barrage QoE while achieving similar or even better energy savings.

Acknowledgment

This work was partially supported by the National Natural Science Foundation of China (Nos. 62072102, 62132009); Jiangsu Provincial Key Laboratory of Network and Information Security (No. BM2003201); the Key Laboratory of Computer Network and Information Integration of the Ministry of Education of China (No. 93K-9); Computer Experimental Teaching Center of Southeast University.

References

[1] M. He, Y. Ge, E. Chen, Q. Liu, and X. Wang, "Exploring the emerging type of comment for online videos: Danmu," *ACM Trans. Web*, vol. 12, no. 1, 2017.

[2] Huya, "Huya," *https://www.huya.com/*, 2022.

[3] "Ieee standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications - redline," *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016) - Redline*, pp. 1–7524, 2021.

[4] R. Kravets and P. Krishnan, "Application-driven power management for mobile communication," *Wirel. Netw.*, vol. 6, no. 4, p. 263–277, 2000.

[5] Y.-s. Lim, Y.-C. Chen, E. M. Nahum, D. Towsley, R. J. Gibbens, and E. Cecchet, "Design, implementation, and evaluation of energy-aware multi-path TCP," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '15. New York, NY, USA: Association for Computing Machinery, 2015.

[6] M. Gundlach, S. Doster, H. Yan, D. Lowenthal, S. Watterson, and S. Chandra, "Dynamic, power-aware scheduling for mobile clients using a transparent proxy," in *International Conference on Parallel Processing, 2004. ICPP 2004.*, 2004, pp. 557–565 vol.1.

[7] L.-S. Meng, D.-s. Shiu, P.-C. Yeh, K.-C. Chen, and H.-Y. Lo, "Low power consumption solutions for mobile instant messaging," *IEEE Transactions on Mobile Computing*, vol. 11, no. 6, pp. 896–904, 2012.

[8] Z. Chen, Y. Tang, Z. Zhang, C. Zhang, and L. Wang, "Sentiment-aware short text classification based on convolutional neural network and attention," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019, pp. 1172–1179.

[9] Z. Li, R. Li, and G. Jin, "Sentiment analysis of danmaku videos based on naïve bayes and sentiment dictionary," *IEEE Access*, vol. 8, pp. 75 073–75 084, 2020.

[10] L. Ding, Q. Wang, and Y. Tian, "Exploiting Danmu interactions for optimizing crowdsourced livecast services," in *2022 7th International Conference on Big Data Analytics (ICBDA)*, 2022, pp. 198–203.

[11] N. Jiang, M. C. Vuran, S. Wei, and L. Xu, "Qos-aware network energy optimization for Danmu video streaming in WiFi networks," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, 2021, pp. 1–10.

[12] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with Pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 197–210.

[13] T. Huang, C. Zhou, R.-X. Zhang, C. Wu, X. Yao, and L. Sun, "Stick: A harmonious fusion of buffer-based and learning-based approach for adaptive streaming," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 1967–1976.

[14] M. Mowafi, E. Taqieddin, and H. Al-Dahoud, "Energy efficient fuzzy-based DASH adaptation algorithm," *Digital Communications and Networks*, vol. 7, no. 1, pp. 113–119, 2021.

[15] K. ARUNRUANGSIRILERT, B. WEI, H. SONG, and J. KATTO, "Performance evaluation of low-latency live streaming of MPEG-DASH UHD video over commercial 5G NSA/SA network," in *2022 International Conference on Computer Communications and Networks (ICCCN)*, 2022, pp. 1–6.

[16] A. Bentaleb, Z. Zhan, F. Tashtarian, M. Lim, S. Harous, C. Timmerer, H. Hellwagner, and R. Zimmermann, "Low latency live streaming implementation in DASH and HLS," in *Proceedings of the 30th ACM International Conference on Multimedia*, ser. MM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 7343–7346.

[17] S. Gronauer and K. Dieopold, "Multi-agent deep reinforcement learning: a survey," *Artificial Intelligence Review*, vol. 55, pp. 1–49, 02 2022.

[18] S. S. Mondal, N. Sheoran, and S. Mitra, "Scheduling of time-varying workloads using reinforcement learning," in *AAAI Conference on Artificial Intelligence*, 2021.

[19] C. Wan, S. Guo, and Y. Yang, "Deep reinforcement learning based computation offloading in SWIPT-assisted MEC networks," in *2022 International Conference on Computer Communications and Networks (ICCCN)*, 2022, pp. 1–10.

[20] R. Chen, L. Liu, K. Sayana, and H. Li, "Energy-efficient wireless communications with future networks and diverse devices," *Journal of Computer Networks and Communications*, vol. 2013, 01 2013.

[21] L. T. Zhang and D. Cassany, "Making sense of danmu: Coherence in massive anonymous chats on Bilibili.com," *Discourse Studies*, vol. 22, pp. 483 – 502, 2020.

[22] F. Wang, X. Fu, and Z. Sun, "A comparative analysis of the impact of barrage and comments on video popularity," *IEEE Access*, vol. 9, pp. 164 659–164 667, 2021.

[23] Z. Cui, Q. Qiu, C. Yin, J. Yu, Z. Wu, and A. Deng, "A barrage sentiment analysis scheme based on expression and tone," *IEEE Access*, vol. 7, pp. 180 324–180 335, 2019.

[24] S. Fan, Y. Lu, L. Zhao, and Z. Pan, "You are not alone: the impacts of Danmu technological features and co-experience on consumer video watching behavior," in *Pacific Asia Conference on Information Systems*, 2017.

[25] A. Y. Ng, D. Harada, and S. J. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *Proceedings of the Sixteenth International Conference on Machine Learning*, ser. ICML '99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, p. 278–287.

[26] Bilibili, "Bilibili," *https://www.bilibili.com/*, 2022.

[27] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep policy gradients: A case study on PPO and TRPO," *ArXiv*, vol. abs/2005.12729, 2020.

[28] Monsoon, "Monsoon power monitor," *https://www.msoon.com/high-voltage-power-monitor.*, 2019.