Maestro: QoE-Aware Dynamic Resource Allocation in Wi-Fi Networks

UMAKANT KULKARNI[*], Purdue University, USA
KHALED DIAB, LIANJIE CAO, FARAZ AHMED, SHIVANG AGGARWAL, and PUNEET SHARMA, Hewlett Packard Labs, USA
SONIA FAHMY, Purdue University, USA

Wi-Fi is an integral part of today's Internet infrastructure, enabling a diverse range of applications and services. Prior approaches to Wi-Fi resource allocation optimized Quality of Service (QoS) metrics, which often do not accurately reflect the user's Quality of Experience (QoE). To address the gap between QoS and QoE, we introduce Maestro, an adaptive method that formulates the Wi-Fi resource allocation problem as a partially observable Markov decision process (PO-MDP) to maximize the overall system QoE and QoE fairness. Maestro estimates QoE without using any application or client data; instead, it treats them as black boxes and leverages temporal dependencies in network telemetry data. Maestro dynamically adjusts policies to handle different classes of applications and variable network conditions. Additionally, Maestro uses a simulation environment for practical training. We evaluate Maestro in an enterprise-level Wi-Fi testbed with a variety of applications, and find that Maestro achieves up to 25× and 78% improvement in QoE and fairness, respectively, compared to the widely-deployed Wi-Fi Multimedia (WMM) policy. Compared to the state-of-the-art learning approach QFlow, Maestro increases QoE by up to 69%. Unlike QFlow which requires modifications to clients, we demonstrate that Maestro improves QoE of popular over-the-top services with unseen traffic without control over clients or servers.

CCS Concepts: • **Networks** → **Wireless access networks**; **Network management**; *Wireless access points, base stations and infrastructure*; *Intermediate nodes*; • **Computing methodologies** → *Machine learning*.

Additional Key Words and Phrases: Wi-Fi, QoE, Access Category, Priority Queue, PO-MDP, LSTM, DDQN

ACM Reference Format:
Umakant Kulkarni, Khaled Diab, Lianjie Cao, Faraz Ahmed, Shivang Aggarwal, Puneet Sharma, and Sonia Fahmy. 2025. Maestro: QoE-Aware Dynamic Resource Allocation in Wi-Fi Networks. *Proc. ACM Netw.* 3, CoNEXT1, Article 4 (March 2025), 24 pages. https://doi.org/10.1145/3709371

1 Introduction

Wi-Fi networks have rapidly become a key element of today's ubiquitous connectivity. They efficiently connect mobile devices to the outside world, support a variety of services, and enable interactive experiences and seamless data exchange. The key to Wi-Fi popularity is its ability to support a wide range of deployments. In addition to small residential deployments, Wi-Fi networks are extensively deployed in large spaces such as airports, shopping malls, campuses, and stadiums, which serve hundreds or even thousands of mobile devices, concurrently running diverse

[*]Umakant Kulkarni worked on this project during an internship at Hewlett Packard Labs.

Authors' Contact Information: Umakant Kulkarni, ukulkarn@purdue.edu, Purdue University, Indiana, USA; Khaled Diab, khaled.diab@hpe.com; Lianjie Cao, lianjie.cao@hpe.com; Faraz Ahmed, faraz.ahmed@hpe.com; Shivang Aggarwal, shivang.aggarwal@hpe.com; Puneet Sharma, puneet.sharma@hpe.com, Hewlett Packard Labs, California, USA; Sonia Fahmy, fahmy@purdue.edu, Purdue University, Indiana, USA.

applications [54]. For example, a single access point in an airport supports several users engaged in streaming, browsing, and video calls, requiring both high throughput and low latency [31]. Similarly, stadiums connect thousands of users to a centralized controller managing millions of traffic flows for live streaming, social media, and real-time updates [50].

Large-scale Wi-Fi networks often require deploying a wireless LAN (WLAN) controller that manages several Wi-Fi access points (APs) and provides different services to users and system administrators, such as traffic classification and access control lists. Modern controllers can perform deep packet inspection (DPI) to classify traffic flows into *application classes* such as video streaming, video conferencing, web traffic, file transfer, or enterprise-specific business applications [20].

To efficiently utilize the limited network resources in large-scale deployments, a WLAN controller needs to carefully allocate resources to each traffic flow. This is accomplished by controlling several configuration parameters. In this work, we restrict our scope to two key parameters: *access category* and *traffic priority* [1], which jointly control the transmission rate of each queue. Specifically, for each traffic flow, the WLAN controller can assign one of the following four *access categories*: Voice, Video, Best-effort, or Background, where each access category corresponds to the length of the transmit opportunity. In addition, the controller can specify the traffic priority of each flow to control the dequeuing rate from the transmission queue.

Resource allocation in Wi-Fi networks has proven to be a complex task for the following reasons. First, most existing methods are *oblivious* to the actual quality of experience (QoE) of users, and instead optimize quality of service (QoS) metrics such as throughput, delay, and packet loss [11, 17]. Although critical to improve, QoS metrics often fail to accurately capture user QoE [55]. For instance, Zhang et al. [76] demonstrated that the highest throughput or lowest round trip time (RTT) does not always correspond to the best QoE, due to the complex relationships between optimized QoS and achieved QoE. Second, proposals such as Wi-Fi multimedia (WMM) [1, 4] that attempt to address this problem employ a *static policy* that assigns traffic flows to a specific access category based on the application class. These static policies, however, do not consider the variable network conditions and heterogeneity of concurrent applications. Thus, static policies may increase network congestion during periods of high demand, which reduces the overall QoE [22]. Third, although more recent work, e.g., QFlow [7], has proposed dynamic resource allocation policies, such work encounters practical deployment challenges because it requires modifying client devices to collect QoE metrics.

In this paper, we propose a new QoE-aware, adaptive, and practical resource manager for Wi-Fi networks, called Maestro, that maximizes overall QoE and QoE fairness, addressing the shortcomings of current methods. The key idea of Maestro is to divide the complex resource allocation problem into two sub-problems and efficiently address each of them. First, Maestro addresses the QoE obliviousness and practical deployment challenges by accurately estimating the achieved QoE for each user without collecting their internal application state. Specifically, Maestro leverages telemetry data that is already available at WLAN controllers, and builds a probabilistic prediction model for each application class to estimate its QoE.

Next, Maestro employs a data-driven approach in its resource allocation policy. Specifically, Maestro applies a reinforcement learning (RL) policy to jointly decide the access category and traffic priority given the estimated QoE metrics and telemetry data. This data-driven approach enables Maestro to dynamically adjust its policy decisions to handle application heterogeneity and variable network conditions. Unlike WMM, Maestro implements a fine-grained allocation policy that does not limit an access category to a specific application class. For example, Maestro can assign video traffic to the voice access category if this assignment increases QoE and/or QoE fairness.

Despite potential benefits, training RL agents for Wi-Fi resource management in real environments is computationally expensive as it requires complex configuration, sending real traffic, and collection of QoE and telemetry. To address this challenge, we design and implement a practical simulation environment that leverages an existing knowledge base of QoE and telemetry data. The environment simulates the training process by starting the RL agent at a random state and using the knowledge base to evaluate actions and determine subsequent states. This optimizes training efficiency and eliminates the need for expensive real setups.

We implement and deploy Maestro in an enterprise-level Wi-Fi testbed consisting of a Wi-Fi 6 AP, an Aruba mobility controller, an Aruba mobility conductor, 4–7 clients, and 4 servers. We use client/server video streaming, video conferencing, file transfer, and audio conferencing applications in our Wi-Fi testbed to evaluate Maestro and compare its performance to existing systems in both single and multiple concurrent application class scenarios. Our experimental results show that Maestro improves QoE and QoE fairness by up to 25× and 78%, respectively, compared to the widely-used WMM policy. Additionally, we execute several popular Internet applications such as Netflix and Zoom concurrently in our Wi-Fi testbed. Our experimental results show that Maestro improves the overall QoE of these applications by an average of 12% and up to 29% compared to WMM. Maestro outperforms WMM in terms of QoE fairness. We also compare Maestro to QFlow which requires access to client devices. Our experimental results show that Maestro can match or exceed the QoE of QFlow despite lack of access to client devices.

In summary, this work makes the following key contributions:

(1) We develop an accurate QoE estimation model for a set of application classes that executes at the WLAN controller, without requiring modifications to client devices. We leverage the temporal dependencies in telemetry data to capture application performance over extended time sequences.
(2) We develop a dynamic and fine-grained resource allocation policy that adapts its decisions using the estimated QoE metrics, without limiting any particular access category to a specific application class.
(3) We design a simulation environment to efficiently train reinforcement learning agents, in order to reduce training time and resource requirements.
(4) We rigorously evaluate Maestro and compare it to state-of-the-art systems using a variety of applications running in an enterprise-level Wi-Fi testbed. Our experimental results show that Maestro can improve the overall QoE by up to 25×.

This work does not raise any ethical issues. We execute open-source and popular applications in our testbed using scripts.

2 Maestro Design and Implementation

In this section, we first provide an overview of the proposed Maestro resource manager and its main components. Then, we describe the details of the QoE estimator and the proposed RL resource allocation policy. Finally, we discuss how we expedite RL training using our simulation environment.

2.1 Overview of Maestro

Maestro is a QoE-aware, adaptive, and practical resource manager for Wi-Fi networks. Unlike several prior systems, we design Maestro to be easy to integrate with existing WLAN controller-based deployments. Figure 1 depicts the main components of Maestro and their interaction with the WLAN controller. As the figure shows, APs are directly connected to the WLAN controller to transmit and receive data, and to periodically send radio, control, and user plane statistics. The WLAN controller has a management interface to configure the Wi-Fi network, push configurations to APs, and manage logs, event triggers, and alarms based on network conditions.

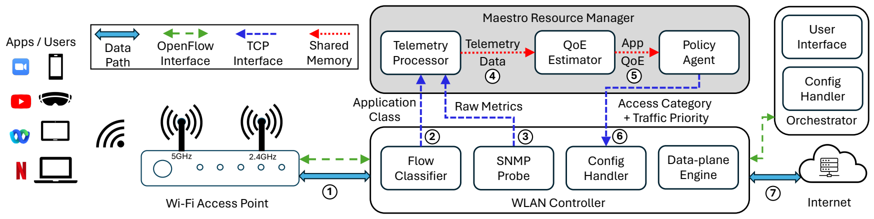


Fig. 1. System operation: ①,② WLAN flow classifier determines application class; ③,④ Maestro telemetry processor passes relevant data to Maestro QoE estimator; ⑤,⑥ Maestro policy agent decides access category and traffic priority based on estimated QoE; ⑦ configured policy is applied on data traffic.

We model our resource allocation problem as a partially observable Markov decision process (PO-MDP). The QoE metrics constitute the system state, and the Maestro resource manager decides the access category and traffic priority for each traffic flow. However, unlike fully observable systems, e.g., QFlow [7], that have access to exact QoE metrics, Maestro only receives telemetry data from the WLAN controller. Maestro also cannot use existing PO-MDP algorithms as they require conditional transition and observation probabilities, which are difficult to obtain in real Wi-Fi deployments. In addition, large PO-MDPs are computationally expensive to solve due to the extremely large search space, which can consist of all historical observations.

We address the challenges of solving our PO-MDP by dividing the resource allocation problem into two sub-problems: QoE estimation and resource allocation policy. Maestro addresses these two sub-problems by employing data-driven approaches to (1) estimate the QoE metrics from previous observations of telemetry, and (2) learn an adaptive resource allocation policy. Maestro allocates resources for elephant flows running for an extended duration with continuous streams of data packets. We leave the resource allocation problem of short-lived flows to future work.

Maestro consists of three components: a telemetry processor, a QoE estimator, and a QoE-aware policy agent, as depicted in the shaded region in Figure 1.

The *telemetry processor* takes raw telemetry data consisting of user, system, and radio metrics from the controller and extracts the relevant data, converting it to the required formats. Note that this telemetry data is the only data to which Maestro has access; no data from the client devices or applications is used by Maestro. To address the heterogeneity in QoE requirements across different applications and media types, Maestro employs dedicated QoE estimators and policy agents for each application class.

The *QoE estimator* (Section 2.2) infers the non-observable QoE metric of each running application from the processed telemetry. Our current design includes four QoE estimator instances for video streaming, video conferencing, file transfer, and audio conferencing applications.

Finally, the *policy agent* (Section 2.3) computes the access category and traffic priority for application flows based on estimated QoE metrics and telemetry data to maximize the overall QoE and QoE fairness. Maestro uses multi-agent RL [10], where each agent serves a specific application class: video streaming, video conferencing, file transfer, or audio conferencing. These agents are loaded into the memory of the device executing the Maestro resource manager, which can either be an external server or the WLAN controller itself. The policy agent modifies application flow configurations via the AP configuration handler, which then pushes these policies to the AP.

2.2 QoE Estimator

We design the QoE estimator to infer the QoE metrics for each application class based on telemetry data available at the WLAN controller. This telemetry includes radio link parameters between the client and the AP, metrics related to the application traffic, wireless capabilities, underlying Wi-Fi configuration, and overall system statistics.

Table 1. Wi-Fi control parameters.

Frequency Band	**PHY/MAC Mode**	**Channel Width**	**Access Category**	**Traffic Priority**
2.4 GHz	High Throughput (HT)	20 MHz	Background (BK)	High Priority (HQ)
5 GHz	Very High Throughput (VHT)	40 MHz	Best Effort (BE)	Low Priority (LQ)
	High Efficiency (HE)	80 MHz	Video (VI)	
		160 MHz	Voice (VO)	

Maestro trains a prediction model for each application class, and pre-loads the model into its memory. Then, when packets arrive to the controller, the flow classifier identifies the application class, and feeds it as an input to the Maestro QoE estimator, which applies the corresponding maintained prediction model to estimate the QoE. The flow classification function is part of the WLAN controller, and hence outside the scope of this paper.

Estimating QoE poses two challenges. First, telemetry data is heterogeneous and significantly varies in scale and format. In addition, this data can include instantaneous values, cumulative statistics, and derived features, each with potential outliers and different domains, ranges, and distributions. Second, the relationship between telemetry and QoE is complex as it is non-linear and is based on hidden or latent features [15].

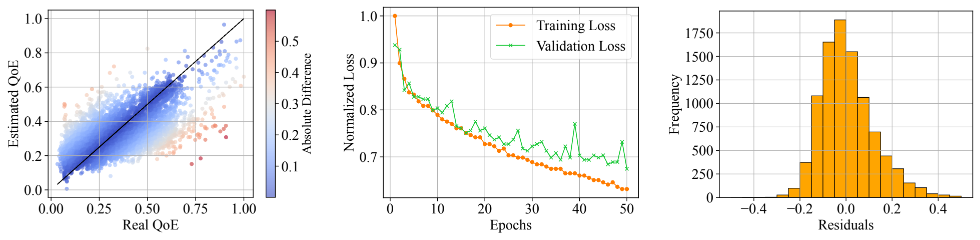
Our key insight in designing the QoE estimator is that QoE metrics are often impacted by temporal context and dependencies of the telemetry data. In other words, the QoE of an application potentially depends on the history of telemetry. Therefore, we need to accurately model the temporal and hidden relationship between telemetry and QoE metrics.

We propose to process sequential telemetry data to capture long-term dependencies while maintaining a reasonably long context over extended periods. Specifically, we model the QoE estimation using a long short-term memory (LSTM) network that can capture trends that develop over time and maintain representative gradient values as we increase the context length. After training the LSTM network, we validated our design choice by comparing it to several prediction models that have been used in the literature in Appendix C.1. Our experimental results show that the proposed QoE estimation model outperforms other prediction models.

To train the QoE estimation model, we execute all applications in our Wi-Fi testbed and collect both telemetry data and the exact QoE for video streaming, video conferencing, file transfer, and audio conferencing. Each LSTM model takes telemetry data as input, and outputs the estimated QoE. We describe the data acquisition, alignment, and model training in the following paragraphs.

Collecting telemetry data and QoE. We gather telemetry and QoE under diverse network conditions ranging from underloaded to overloaded scenarios by systematically varying Wi-Fi configurations as outlined in Table 1. These network configurations encompass 15 different parameters, including frequency band, PHY/MAC mode, channel width, access category, and traffic priority, with each parameter varied independently. Additionally, for each configuration setting, we vary the number of concurrent applications from 1 to n, where n is chosen such that all concurrently running application instances will congest the network, leading to poor QoE. We distribute application sessions evenly among 4 Wi-Fi client devices. For each session, we execute the applications and measure the actual QoE at the client devices. We run each session for 200 seconds and repeat every scenario twice to increase the size and diversity of the dataset. Details of the applications and performance metrics are given in Sections 3.2 and 3.3, respectively. Using this approach, our telemetry data encompasses applications running for a total of 402 hours across all clients for all four application classes, resulting in over 362K data points. We refer to this dataset as the *knowledge base* as it encompasses telemetry data, corresponding QoE metrics, and system configurations.

Aligning telemetry data with QoE. We collect the telemetry data every four seconds. We calculate the QoE metrics of applications within every interval as described in Section 3.3. We also normalize



(a) Real vs. estimated QoE (b) Training and validation loss (c) Residual distribution

Fig. 2. Performance of the Maestro QoE estimator.

the QoE metrics against the maximum achievable QoE value for each application, which allows comparing QoE values across application classes.

The QoE and telemetry data are synchronized based on their timestamps. Since each application may report its QoE at a different rate than that of telemetry, we need to align these measurements by averaging the QoE metrics over each four-second interval. We associate telemetry data collected between t and $t + 4$ seconds with the corresponding QoE metrics observed during that period to ensure a correct alignment. This mapping accurately reflects the system's state and captures the network conditions that influence the QoE outcome. We maintain consistency between observed telemetry and the resulting QoE by structuring the data collection and QoE measurement in synchronized intervals.

Training the QoE model. We construct an LSTM network using an input layer of 257 telemetry data items [42] and three stacked LSTM layers, where the first, second, and third layers have 256, 128, and 64 hidden units, respectively. The LSTM layers are followed by a dense output layer to compute the estimated QoE metric. We train the proposed model using the Adam optimizer and mean squared error (MSE) loss function. We divide the dataset into training, validation, and test sets using split ratios of 60%, 20%, and 20%, respectively. The training process comprises 50 epochs with a batch size of 32. Appendix A gives further details.

Figure 2(a) plots the actual versus model-estimated QoE on the test dataset for a video conferencing application. The majority of data points align closely with the 45-degree line. Figure 2(b) depicts the training and validation loss over the 50 epochs; the validation loss stabilizes after approximately 30 epochs. Figure 2(c) shows the distribution of residuals, which indicates that the majority of residuals are centered around zero, demonstrating a high QoE estimation accuracy.

2.3 QoE-aware Policy Agent

The *policy agent* assigns the access category and traffic priority for each flow given the estimated QoE, with the objective of maximizing the overall QoE and QoE fairness. Although the state of our PO-MDP is estimated using the QoE estimator, the transition probability is not easy to model due to the complex relationship between actions taken and observed state. We thus design a data-driven policy using a multi-agent reinforcement learning (RL) approach, where each agent corresponds to an application class. We use a flow identifier to distinguish among flows in each class.

The key challenges of designing an RL policy in Maestro are the large search space and the lack of sufficient training data. We therefore design an off-policy RL approach that can explore possibilities in the search space while efficiently utilizing collected state. We control the exploration process to speed up model convergence. Further, we employ a deep neural network to handle the high-dimensional state-action space in the Wi-Fi setting, caused by the multiple application classes, dynamic network conditions, and diverse client capabilities. The proposed policy agent uses the double deep Q-network (DDQN) RL method [24] coupled with a feed-forward neural network to handle the high-dimensional space [37], and a replay buffer to increase sample efficiency. We give further details on components of DDQN in Appendix B and compare it to other RL algorithms in Appendix C.2.

Action space. Maestro aims at controlling the parameters related to the radio link between Wi-Fi client devices and the AP. Most WLAN controllers enable configuring flow-level, radio-level, and device-level parameters. Since configuring radio-level and device-level parameters requires restarting the AP or devices, we design Maestro to only configure flow-level parameters that can be modified during run-time.

Maestro jointly assigns access category [4] and traffic priority [3, 7, 9, 56, 60] for each flow. When assigning access categories, Maestro does not limit a given application class to a specific access category. Instead, it assigns incoming flows to any access category based on the learned policy in order to adapt to application heterogeneity and diverse network conditions. Flows are assigned to any of the three access categories: Best Effort (BE), Video (VI), and Voice (VO). In addition, Maestro assigns either a low priority (LQ) or a high priority (HQ) to each flow, which controls the drain rate of packets. Maestro executes these actions for each application every 16 seconds.

By combining the low and high priority queues (LQ/HQ) with the access categories, the action space in Maestro consists of the following six possible action combinations: LQ+BE, LQ+VI, LQ+VO, HQ+BE, HQ+VI, HQ+VO.

State space. The state space is composed of the estimated application QoE and the telemetry data [42]. The smallest unit on which a learning agent takes action is a 5-tuple application flow traversing a Wi-Fi AP. Given the complexity and volume of data available within the controller database, comprising thousands of metrics, a primary challenge is identifying which of these metrics significantly impact QoE estimation. To ensure comprehensive coverage, we used all available metrics allowing the QoE estimator to select a pertinent subset and accurately estimate the QoE.

We categorize telemetry data extracted from the controller's database [42] into three types: (1) application-related data, (2) client-specific data, and (3) access point-specific data. Application-related data includes metrics such as flow-specific throughput, transport protocol of the flow, and TX/RX bytes. Client-specific data encompasses the wireless capabilities of the client, signal-to-noise ratio (SNR), PHY/MAC interface details, channel width, and access category. Access point data involves system-specific metrics such as overall packet loss, packets dropped, buffer TX/RX metrics, and radio metrics.

Reward function. We design the reward function for the DDQN to maximize overall QoE and to maximize QoE fairness [26]. In addition to the state defined earlier, each RL agent receives (1) the mean QoE of all other applications and (2) the deviation of the particular application's QoE from the mean QoE. After taking an action a_t at state s_t, we define the reward function as:

$$R(s_t^{app}, a_t^{app}) = w_{\mathrm{QoE}} \cdot \mathrm{QoE}_{\mathrm{app}} - w_{\mathrm{mean}} \cdot \overline{\mathrm{QoE}}_{\mathrm{others}} - w_{\mathrm{deviation}} \cdot \mathrm{deviation}, \tag{1}$$

where $\mathrm{QoE}_{\mathrm{app}}$ is the QoE of the current application flow, $\overline{\mathrm{QoE}}_{\mathrm{others}}$ is the mean QoE of all other application flows, and deviation $= |\mathrm{QoE}_{\mathrm{app}} - \overline{\mathrm{QoE}}_{\mathrm{system}}|$ is the deviation of the current application's QoE from the system-wide mean QoE, where $\overline{\mathrm{QoE}}_{\mathrm{system}} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{QoE}_i$ is the system-wide mean QoE for N application flows. The weights w_{QoE}, w_{mean}, and $w_{\mathrm{deviation}}$ are used to adjust the sensitivity of the application QoE to the Wi-Fi environment, allowing the reward function to be tailored to the specific needs of applications. This function maximizes the QoE of the current application, balances it against the mean QoE of all other applications, and maximizes QoE fairness by minimizing deviation in QoE metrics [26].

Algorithm 1 Train the QoE-aware policy agent.

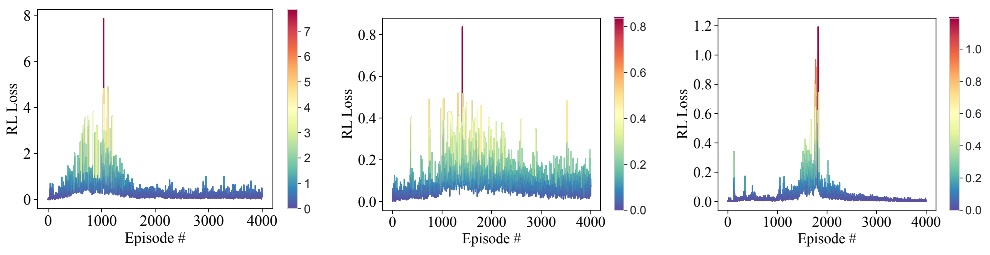
```
1: Initialize knowledge base $\mathcal{K}$ with telemetry data and QoE values
2: Initialize RL agent with selected network parameters and random policy $\pi_\theta$
3: Set initial state $s_0 \in \mathcal{K}$ randomly
4: for each episode do
5:     $s \leftarrow s_0$
6:     for each step $t$ do
7:         Select action $a_t$ using the Boltzmann exploration mechanism (Appendix Eq. B.3)
8:         Evaluate action $a_t$ based on $\mathcal{K}$.
9:         Identify best matching vector $\mathbf{v} = \arg\max_{\mathbf{v}' \in \mathcal{K}} \rho\left((s_t, a_t), \mathbf{v}'\right)$
10:        Observe reward $r_t$ and next state $s_{t+1}$ from $\mathbf{v}$
11:        Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer
12:        Update state $s \leftarrow s_{t+1}$
13:        Calculate MSE loss (Appendix Eq. B.4)
14:        Perform Q-learning update to policy $\pi_\theta$ (Appendix Eq. B.2)
15:    end for
16: end for
```

2.4 Simulation Environment for Training the Policy Agent

Training the policy agent in Maestro poses several challenges. The training process requires access to state variables, execution of actions, and observation of rewards and next states. Training on a real Wi-Fi network requires deploying multiple client devices, generating traffic through creating application sessions, and recording both QoE and telemetry data. Obtaining a sufficient number of diverse samples for the RL agent to learn a policy is impractical due to the complexity and variability of realistic conditions. In addition, these real Wi-Fi conditions cannot be easily integrated with current RL environments [23, 33, 41]. Finally, exploring all possible state-action combinations is infeasible, as it requires an excessive amount of time and resources.

We thus design and build a simulation environment by leveraging a knowledge base collected during QoE estimator training. The knowledge base includes telemetry data and corresponding QoE values. The simulation environment reflects Wi-Fi network dynamics through initializing the RL agent with states derived from the knowledge base and through executing policy actions that are assessed using historical data. This approach creates a realistic training loop without deploying physical devices during training, thus expediting learning. The iterative learning process is described in Algorithm 1, where the RL agent transitions through states by deriving the next state vector from the knowledge base using the Pearson correlation coefficient ρ [61] and selecting actions using Boltzmann exploration. This ensures that transitions in the simulation closely reflect real network behavior, allowing the agent to iteratively learn effective policies based on realistic scenarios drawn from empirical data.

We estimate the reward function weights (Equation 1) and the Boltzmann exploration temperature values (Equation B.3) by testing five representative sets of weights and ten temperature values ranging from 0 to 1 in increments of 0.1, feeding each combination into Algorithm 1. The optimal values are selected by fitting the RL training loss to the function $L(t) = \alpha \cdot t \cdot e^{-\beta t}$, which models the expected loss behavior during training [19]. As shown in Figures 3(a), 3(b), and 3(c), the training loss initially rises during exploration, peaks, and gradually decreases to zero as the agent learns and refines its policies. Table 5 in Appendix B lists the optimized weights and temperature values for each application class.



(a) Video streaming (b) Video conferencing (c) File transfer

Fig. 3. RL training loss for different applications classes.

3 Evaluation

We extensively evaluate Maestro in our Wi-Fi testbed with different classes of applications. Our evaluation aims to answer the following questions: (1) How much QoE and QoE fairness improvement does Maestro achieve compared to the widely-deployed WMM policy and state-of-art learning-based policy? (2) To what extent does Maestro improve the individual quality metrics of applications? (3) How well does Maestro perform with popular over-the-top (OTT) services? and (4) What is the resource footprint of Maestro?

3.1 Testbed Setup

Our testbed emulates a small campus deployment, with a controller-based Wi-Fi setup. It contains an Aruba 555 Wi-Fi 6 AP [27] connected to the Internet and egress network via an Aruba 7010 Mobility Controller [28], as shown in Figure 4. The Wi-Fi devices are managed by an Aruba Mobility Conductor [29] for policy enforcement, resource allocation, and wireless/RF optimization.

We deploy seven wireless devices to represent a diverse set of clients. We use four Acer Aspire laptops, each of which runs Ubuntu 20.04 as its OS and is equipped with a Wi-Fi 6E Intel AX210 wireless card [68]. In addition, we use a PC running Windows 10, a laptop running macOS 15.0.1, and an iPhone XS. The wireless clients maintain a clear line of sight with the AP. The clients are connected over the 2.4 GHz frequency band. The PHY/MAC mode is set to High Throughput (HT) with a channel width of 20 MHz. We configure our Wi-Fi deployment with these parameters to saturate the network, ensuring that even a small number of clients or applications fills the queues and creates resource contention.

We set up four servers, specifically HPE ProLiant DL20, DL60, DL120, and DL360 to serve content to the clients. The servers run Ubuntu 20.04 with kernel version 5.15.0.107-generic. The servers and controller are connected to a switch over a Gigabit Ethernet interface. We do not limit the uplink or downlink bandwidth within the Wi-Fi environment. The Internet downlink capacity to our lab is 35 Mbps.

We run the Maestro resource manager on one of the four servers. The resource manager executes the telemetry processing, QoE estimation, and RL policy agent. The server sends the selected policy to the Wi-Fi controller to apply to the network flows.

3.2 Applications

Our evaluation uses *local applications* between local clients and local servers hosted in our lab, and *OTT services* between local clients and servers on the Internet. Local applications are fully managed within our lab, whereas OTT services fetch content from a remote server over the Internet. We use the following local applications in our experiments.

Table 2. Enhanced distributed channel access parameters for WMM-enabled AP.

Category	AIFSN	ECWmin	ECWmax	TXOP
Best-Effort (BE)	3	4	6	0
Background (BK)	7	4	10	0
Video (VI)	2	3	4	94
Voice (VO)	2	2	3	47

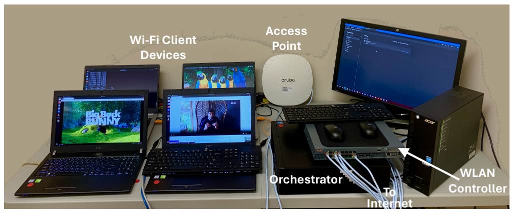


Fig. 4. Our Wi-Fi evaluation testbed.

(1) Video Streaming Application: We leverage an open-source application [5] that uses the Dynamic Adaptive Streaming over HTTP (DASH) protocol. We configure an HTTP server to stream the "Big Buck Bunny" video [52] to the wireless client in 4-second segments over HTTP. The video is encoded at 24 frames per second using the H.264 codec for video and the AAC codec for audio.

The application employs a buffer-based rate adaptation algorithm [30]. The algorithm dynamically selects the quality of each video chunk and requests the appropriate chunk from the server. This varies the required bandwidth from 3.4 to 47 Mbps.

(2) Video Conferencing Application: We run an open-source application [66] that starts a video conferencing session between two clients. The clients exchange audio and video streams in real-time. The control plane is established using the STUN protocol, and a TURN server is used for the media exchange.

To simulate a realistic conferencing session, we follow a similar methodology as described in [18] wherein a pre-recorded video of a talking head from YouTube [74] streams both audio and video at a variable frame rate using the H.264 codec for video and the OPUS codec for audio. The video conferencing application [66] uses the WebRTC APIs and protocols [21] to deliver RTP packets and adjust the frame rate and video resolution based on network conditions. Consequently, the maximum bandwidth requirement is 20 Mbps.

(3) File Transfer Application: We use the iPerf3 tool [34] to simulate a file transfer application, where the server sends data to the client using TCP. The TCP congestion control protocol is set to the default CUBIC algorithm. No application bandwidth limit is imposed, allowing file transfer to utilize as much bandwidth as is available.

(4) Audio Conferencing Application: This application is the same as video conferencing, but only audio is enabled in both directions. The audio component utilizes the OPUS codec.

3.3 Performance Metrics

We report QoE values for each application, and also compute QoE fairness. Although subjective studies can yield the user-perceived QoE, they are infeasible to conduct for the large number of system configurations in our experiments. Therefore, we use existing objective QoE metrics that are known to capture the QoE perceived by users. We now describe these metrics.

(1) Video Streaming QoE: We use the QoE metric proposed in prior work [72] where the QoE is a function of the quality of each video chunk, the quality variation between successive chunks, rebuffering time, and startup delay. The rationale behind this QoE metric is that users prefer uninterrupted video playback without stalls once the video starts, and frequent changes in video quality between chunks are undesirable. The best possible QoE is achieved with minimal startup delay, zero rebuffering, zero quality variations, and the highest video resolution throughout the streaming session. Each chunk's QoE is normalized against this best possible QoE score. We set the metric weights to balance between rebuffering and quality instability, which is referred to as "balanced mode" [72].

The QoE values range from $-\infty$ to 1. However, our evaluation shows that the QoE rarely goes below -1. Thus, we restrict the range of QoE values from -1 to 1, and normalize these values between 0 to 1 using the min-max normalization method. This normalized QoE is reported for each four-second video chunk, which is the same duration of our collection interval.

(2) Video Conferencing QoE: We compute audio jitter, audio packet loss, video packet loss, and video packets sent [32]. The rationale is that users find it most annoying when real-time sound is choppy or when video frames are out of sync with audio [32].

The best possible QoE for the video conferencing application is achieved with the highest resolution video, the maximum possible frame rate, and minimal jitter and packet loss. QoE values are normalized against this best possible QoE score. We collect the values using the WebRTC APIs [67] every second. To map these QoE metrics to a four-second interval, we average the QoE values over 4 seconds and report a single QoE value.

(3) File Transfer QoE: We cannot use the total completion time for a file transfer since we need to periodically compute QoE. Thus, we record the number of bytes transferred from server to client in one second [65]. This data is then averaged over a four-second interval to compute the QoE for that period, and align it with our four-second telemetry collection interval. The rationale is that users prefer file transfers to complete in the shortest time possible, which can be achieved if the server consistently sends large amounts of data.

The best achievable QoE for file transfer is determined using the optimal Wi-Fi configuration, with a 160 MHz channel using the 5 GHz frequency band and HE physical interface. At this setting, the server can transfer the maximum number of bytes to the client, representing the best achievable QoE. We normalize the QoE for each four-second interval against this score.

(4) Audio Conferencing QoE: We report the jitter and packet loss of the audio stream as the QoE metrics. Normalization follows the same approach we use in the video conferencing application.

(5) QoE Fairness: We compute the QoE fairness score using Jain Fairness Index [36]. We average the QoE for each client over the entire session, and apply the Jain Index to these per-client mean QoE values to report the overall QoE fairness of the system.

3.4 Experimental Methodology

Each application session starts with the default configuration with access category set to best effort (BE) and priority queue to low (LQ). We run each application session for five minutes. Each experiment is repeated four times, resulting in a total runtime of 20 minutes per application. We report the mean score across these 20-minute sessions.

We compare Maestro with static and dynamic policies. For the static policy, we employ the widely-used WMM policy that assigns video traffic to the VI (Video) access category, audio/video conferencing traffic to the VO (Voice) access category, and file transfer traffic to the BK (Background) access category. Since a DPI module is required to support the WMM static policy, APs that do not have this functionality simply assign the Best Effort (BE) category to all traffic. The parameters of WMM access queues are configured as shown in Table 2. We disable the admission control policy on the Wi-Fi AP to ensure that no flow is rejected due to resource availability constraints. For the dynamic policy, we compare Maestro with QFlow [7] that implements an RL-based approach. The implementation of QFlow is described in Appendix E.

3.5 Results with Local Applications

To evaluate how Maestro handles multiple application sessions, we first conduct experiments with multiple instances of the same local application. Then, we extend our experiments to run multiple instances of different local applications. All application sessions in our experiments are executed concurrently. We analyze the performance of Maestro using comprehensive QoE metrics and detailed case studies of critical individual quality metrics. We also demonstrate the operation of Maestro through a specific example in Appendix D.

3.5.1 Same Application Class. In this scenario, we evaluate the effectiveness of Maestro with one active RL agent at a time. Two clients run a local application while the remaining five clients generate background traffic. The background traffic creates resource contention to stress Maestro and other approaches. Background traffic is assigned an access category per WMM standards and a low priority queue. The background traffic is generated by running video streaming, video conferencing, file transfer, web browsing, and a point cloud application on five clients. This experiment is conducted for each of the local applications.

As shown in Figure 5, Maestro demonstrates substantial improvements across applications, particularly excelling for file transfer, where it achieves over 25× the QoE of WMM and nearly double the QoE of QFlow. For video streaming, Maestro offers a 14% gain over WMM and a 69% gain over QFlow. For video conferencing, Maestro achieves a 6% improvement over WMM and 16% over QFlow, while for audio conferencing, it shows a 4% increase over WMM and QFlow. When applications share access category queues with background traffic, resource contention and congestion become critical. Maestro has a significant advantage by dynamically allocating resources based on estimated QoE and the current system state, optimizing both channel access and priority allocation to maximize QoE. Unlike QFlow that only adjusts priority queues, and WMM that uses fixed priority queue and access categories, Maestro's adaptive approach to managing both channel access and priority queues ensures consistently high QoE.

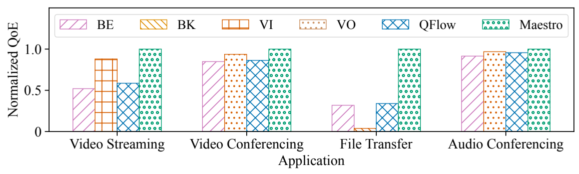


Fig. 5. Concurrent instances of the same application class.

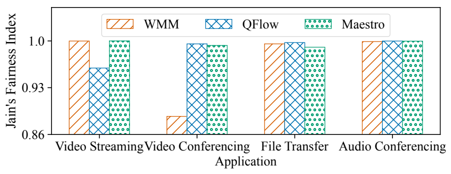


Fig. 6. QoE fairness.

We observe that for video streaming, the large disparity between low and high priority queues in QFlow can lead to significant QoE variations between clients, reducing fairness as seen in Figure 6. For video conferencing, WMM's fairness suffers due to limitations in the GCC algorithm; when audio or video quality is disrupted, it adjusts to a lower quality only briefly instead of adapting continuously, resulting in one client achieving a consistently higher QoE over another. For file transfer, Maestro achieves much higher data throughput than the other policies, leading to a slightly higher fairness index for WMM and QFlow. This is expected due to the significant throughput advantage achieved by Maestro for file transfer QoE.

3.5.2 Different Application Classes. We evaluate the performance of Maestro in a multi-agent environment by concurrently executing multiple local applications of different classes. We create all combinations of three local applications: video streaming (VS), video conferencing (VC), and file transfer (FT). We exclude the audio conferencing application from this set of experiments due to its low bandwidth requirements and low sensitivity to resource adjustments.

When video streaming and file transfer (VS+FT) run concurrently, we deploy two instances of video streaming on two clients and two instances of file transfer on two other clients. The same setup is used for video streaming and video conferencing (VS+VC) and video conferencing and file transfer (VC+FT). We also experiment with all three local applications running simultaneously (VS+VC+FT), where each client executes an instance of each local application. For better legibility, we aggregate the QoE of multiple application instances by first normalizing each application QoE with respect to its maximum value and then calculating the average across all applications.

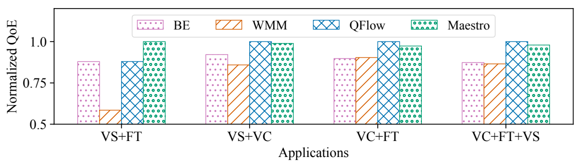


Fig. 7. Concurrent instances of different application classes.

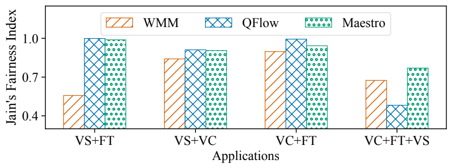


Fig. 8. QoE fairness.

Maestro consistently outperforms WMM in all scenarios, as shown in Figure 7. The most significant QoE gains occur when video streaming and file transfer run concurrently, where Maestro achieves a 71% improvement over the WMM policy. For video streaming and video conferencing, Maestro provides a 15% higher QoE than WMM. For video conferencing with file transfer, Maestro performs 8% better than WMM. When all three applications run concurrently, Maestro outperforms WMM by 13%. Maestro avoids airtime contention that arises when multiple applications compete within the same access category and priority queue under the static policy of WMM. By learning application characteristics and queue behavior, Maestro dynamically determines the optimal queue assignment for each application flow.

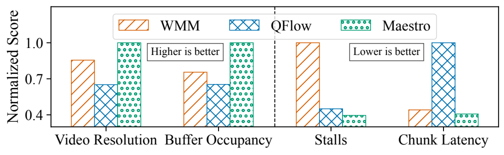
QFlow achieves similar or slightly higher QoE and fairness than Maestro in specific cases. QFlow's advantage lies in its direct access to real QoE feedback, providing it with perfect knowledge for decision making. In contrast, Maestro does not require access to actual user QoE metrics, and thus Maestro is easy to deploy. Despite some QoE estimation errors, Maestro achieves high QoE across all scenarios, while WMM and QFlow yield inconsistent QoE results as they perform poorly in certain cases.

Figure 8 shows that the highest fairness gain with Maestro is observed when video streaming and file transfer run concurrently. Maestro improves fairness by 78% over WMM. In the scenario with three applications, Maestro achieves a 60% improvement in fairness over QFlow. The substantial differences in resource access methods between the BK static policy for file transfer and the VI static policy for video streaming, listed in Table 2, are prominently reflected in their respective fairness indices. A similar pattern is observed in scenarios involving other mixed sets of applications. In contrast, Maestro seeks to optimize the QoE for each application as dictated by the reward function given in Equation (1), maximizing individual QoE and QoE fairness.

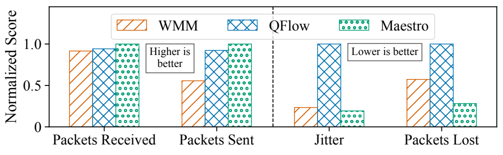
3.5.3 Individual Quality Metrics. To gain a deeper understanding of the different aspects of user experience, we analyze the individual quality metrics of video streaming and video conferencing applications aggregated over four runs and compare it with both WMM and QFlow.

Figure 9(a) presents aggregate video resolution, stalls, chunk latency and buffer occupancy for the video streaming application. Maestro improves video resolution by 17% over WMM and 53% over QFlow, reduces stalls by 60% and 12%, respectively, and lowers chunk latency by 8% and 59%. Buffer occupancy also increases by 32% compared to WMM and 53% compared to QFlow, which indicates a smoother playback experience.

We examine four individual quality metrics for video conferencing: audio jitter, audio packet loss, video packets received, and packets sent, which are chosen for their significant impact on QoE as established through prior statistical analysis of WebRTC metrics [32]. As shown in Figure 9(b), Maestro reduces audio jitter by 18% over WMM and 81% over QFlow, while lowering audio packet loss by 51% and 72%, respectively. It also increases video packets received by 9% compared to WMM and 6% compared to QFlow, with video packets sent rising 80% and 8%, respectively. These



(a) Video streaming



(b) Video conferencing

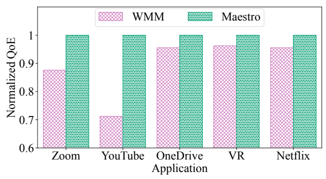
Fig. 9. Individual quality metrics.

improvements indicate that the QoE estimator and policy agent modules effectively learn application characteristics and their impact on user QoE from the system state.

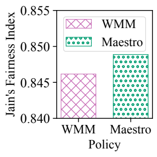
3.6 Results with OTT Services

We analyze the performance of Maestro when the Wi-Fi clients are accessing popular OTT services including YouTube, Netflix, Zoom, virtual reality (VR), and OneDrive. Our goal is to show the performance, usefulness, and feasibility of our approach in real Wi-Fi deployments with unseen applications by leveraging characteristics of an applications class (Section 2.2). Specifically, in these experiments, we cannot control the clients or servers or background traffic. We also do not retrain the Maestro models as we do not have access to the actual QoE values. We compare Maestro against the WMM policy, which assigns an application flow to a fixed queue: YouTube, VR, and Netflix to LQ-VI, OneDrive to LQ-BK, and Zoom to LQ-VO per Wi-Fi industry standards [69]. We cannot compare Maestro against QFlow here because QFlow requires modifying the clients to provide real-time QoE feedback.

For YouTube, we stream a video [75] at 4K resolution. Netflix streams episodes of the series "Bridgerton" on the iPhone. For Zoom, we conduct a video conferencing session between a Linux laptop and a MacBook Pro over the Internet using a talking head video. For the VR application, we interact with a soldier object [2] hosted on AWS with six degrees of freedom using Potree [57] with simulated actions. For OneDrive, we downloaded a 2 GB file to the Windows desktop.

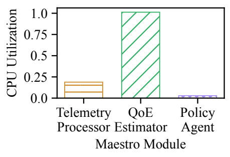


(a) QoE

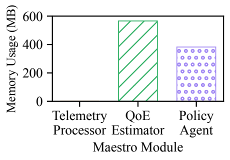


(b) QoE fairness

Fig. 10. OTT services.



(a) CPU



(b) Memory

Fig. 11. Resource footprint.

We normalize the QoE for each application against the maximum value observed such that the highest mean QoE attains a value of 1. Figure 10(a) shows that Maestro performs on average 12% better across all applications compared to the WMM policy with up to 29% improvement seen in case of YouTube. Maestro QoE fairness is also slightly higher as seen in Figure 10(b). This validates that the models trained on local applications effectively manage OTT services, and that Maestro can learn application characteristics and select appropriate policies.

3.7 Resource Footprint of Maestro

We measure the resource footprint of the resource manager of Maestro under high load. Specifically, we load three QoE estimation models and run the corresponding RL policy agents. The Wi-Fi network runs five clients accessing OTT services over the Internet. We observe similar performance when running local applications. In this experiment, we deploy the resource manager on an HPE DL360 server with 40 CPUs and 377 GB of DRAM. We show the resource usage (CPU and memory) of all three modules in Figure 11.

The telemetry processor uses approximately 0.2 CPU with negligible memory footprint. This is because it primarily sorts the raw data collected from the controller and stores it on disk. The QoE estimator is the most resource-intensive module, using almost 1 CPU and around 550 MB of memory. The current implementation of the QoE estimator is single-threaded and 1 CPU is sufficient for our experiments. The experiment logs show that QoE estimations are completed within one second, which is much lower than the decision-making requirement of four seconds. For larger experiments with more application sessions, the QoE estimator can be scaled to multiple instances. The QoE estimator also runs the LSTM models for the three application classes. It loads the three LSTM models into memory and applies inference without GPU acceleration, demonstrating that our system can operate on compute-constrained infrastructure. The observed memory usage is typical for LSTM neural network models [77].

The main task of the policy agent is to generate configuration actions, which uses little CPU. However, the agent maintains a replay buffer, a feed-forward neural network, and a Q-network, which hold the entire system state. This results in memory consumption of around 400 MB. Overall, the results demonstrate that Maestro does not require extensive CPU and memory resources.

4 Related Work

QoE estimation. A number of past studies have estimated user QoE for specific applications from passive measurements. eMIMIC [43], YOUQMON [12], VideoNOC [44], and QoEBoX [51] use DNS data, application-layer data or network-layer log data to understand application performance, mostly focusing on video streaming applications. ExBox [13] uses IQX hypothesis to estimate QoE for admission control purposes. Sharma et al. [59] developed an ML-based method which uses network telemetry and flow statistics to infer WebRTC video QoE. In contrast, the proposed QoE estimation approach does not require modifications to Wi-Fi clients or applications.

Traffic prioritization. Several studies have employed flow prioritization over wireless links by allowing an end user to configure a home router [8, 45, 58]. Recent prioritization approaches, such as PIAS [6] and QFlow [7], run a separate process on an end-user device to communicate with the network and prioritize flows based on usage or network telemetry. This prior work primarily considers a single type of application and specific types of networks. Kulkarni et al. [39] showed the impact of different Wi-Fi configuration options on performance, but did not propose a prioritization method. In contrast, we propose a dynamic resource allocation method that supports multiple application classes and does not restrict application traffic to specific access categories.

QoE-aware resource allocation. Xu et al. developed a framework [70] that applies QoE-centric metrics specifically for video quality to improve spectral and energy efficiency in heterogeneous networks. A learning-based model from Tao et al. [62] incorporates human perception limits to enable dynamic adaptation of network resources to improve video QoE. Kulkarni et al. propose a QoE-aware path selection method [38] that considers load averages and variations to enhance QoE over standard routing techniques. These prior solutions do not dynamically optimize user-perceived QoE for diverse applications and network conditions.

RL-based networking. RL has been effectively used to optimize several network environments. In cellular networks, RL has improved radio resource management [16], association control [35], spectrum access [47], and content delivery [14, 73]. In other wireless networks, RL has improved medium access control [71], resource allocation in IoT [25], and routing in wireless sensor networks [46, 64]. For example, Zhao et al. [78] propose a system that performs IoT resource allocation using a model-free deep RL framework to optimize user experience through estimated QoE-based utility functions. Moura et al. [49] introduce a control loop for transmission power and channel selection using Q-learning to enhance web QoE metrics. In contrast, Maestro dynamically adapts resource allocation based on the estimated QoE for diverse application classes at a fine granularity.

5 Limitations and Practical Considerations

Although our results are promising, we acknowledge that our experiments were highly controlled. In this section, we outline limitations and deployment considerations for Maestro.

Mobility and variability. Our experiments used a fixed number of static clients with direct line-of-sight to the AP, mostly communicating with each other. Maestro includes mobility-related parameters in its training data, but factors such as client mobility and highly variable wide-area traffic were not explicitly evaluated. Such factors may add complexity to resource allocation.

Short-duration flows. Maestro optimizes long-duration flows such as audio/video streaming, conferencing, and file transfers, where QoE estimation and policy actions significantly enhance user experience. Short-duration flows, such as web browsing or instant messaging, are excluded from direct management due to their transient nature, which does not allow sufficient time for meaningful QoE optimization. However, these flows are included as background traffic in experiments.

Multi-agent training dynamics and convergence behavior. Maestro employs a multi-agent RL framework to address convergence challenges in large state-action spaces. These RL agents independently optimize resources for specific application classes but share system-level data, including telemetry and QoE metrics. This enables coordinated decisions. The unified reward function ensures alignment with overall network goals and prevents conflicting actions.

Policy update frequency. Maestro's policy update intervals are carefully chosen to balance responsiveness and stability. A 4-second telemetry cycle aligns with typical application patterns such as adaptive bitrate streaming, while the 16-second policy update interval accumulates sufficient data for effective decision-making. This design avoids excessive adjustments that could destabilize applications, but ensures responsiveness through adaptation to changing network conditions.

Generalizability. Maestro generalizes to unseen applications by leveraging common patterns such as greedy TCP for file transfers, bursty TCP for streaming, and real-time UDP for conferencing. By associating each application class with established traffic characteristics and corresponding QoE requirements, Maestro's models can be applied even if a specific application was not in the training dataset, as long as the application has been correctly classified (Section 2.2).

6 Conclusions

We presented the design, implementation, and evaluation of Maestro: a QoE-aware and adaptive resource manager for Wi-Fi networks. Maestro aims at maximizing the overall QoE and the QoE fairness even in diverse network and client conditions without accessing client devices. Maestro addresses resource allocation through (a) QoE estimation and (b) adaptive policy learning. Using a Partially Observable Markov Decision Process (PO-MDP) framework, Maestro leverages telemetry data from controllers to estimate QoE through an LSTM model, while optimizing resource allocation using Double Deep Q-Network (DDQN) reinforcement learning. Compared to the widely-deployed WMM policy, Maestro improves QoE by up to 25 times and QoE fairness by as much as by 78%. Maestro also improves QoE by up to 69% compared to a state-of-the-art learning approach.

Acknowledgments

We sincerely thank the anonymous CoNEXT reviewers for their insightful feedback. This research was supported in part by NSF grant 2212200.

References

[1] 2021. 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. *IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016)* (2021), 1–4379. https://doi.org/10.1109/IEEESTD.2021.9363693

[2] 8i 2024. 8i Voxelized Surface Light Field (8iVSLF) Dataset. https://mpeg-pcc.org/index.php/pcc-content-database/8i-voxelized-surface-light-field-8ivslf-dataset/.

[3] Air Slice 2024. Air Slice. https://www.arubanetworks.com/techdocs/ArubaOS_8.11.0_Web_Help/Content/arubaos-solutions/access-points/airslice.htm.

[4] Wi-Fi Alliance. 2024. WMM. https://www.wi-fi.org/discover-wi-fi/specifications.

[5] AStream 2024. AStream: A rate adaptation model for DASH. https://github.com/pari685/AStream.

[6] Wei Bai, Li Chen, Kai Chen, Dongsu Han, Chen Tian, and Hao Wang. 2017. PIAS: Practical Information-Agnostic Flow Scheduling for Commodity Data Centers. *IEEE/ACM Transactions on Networking* 25, 4 (2017), 1954–1967. https://doi.org/10.1109/TNET.2017.2669216

[7] Rajarshi Bhattacharyya, Archana Bura, Desik Rengarajan, Mason Rumuly, Bainan Xia, Srinivas Shakkottai, Dileep Kalathil, Ricky K. P. Mok, and Amogh Dhamdhere. 2022. QFlow: A Learning Approach to High QoE Video Streaming at the Wireless Edge. *IEEE/ACM Transactions on Networking* 30, 1 (2022), 32–46. https://doi.org/10.1109/TNET.2021.3106675

[8] Ilker Nadi Bozkurt and Theophilus Benson. 2016. Contextual Router: Advancing Experience Oriented Networking to the Home. In *Proceedings of the Symposium on SDN Research* (Santa Clara, CA, USA) *(SOSR '16)*. Association for Computing Machinery, New York, NY, USA, Article 15, 7 pages. https://doi.org/10.1145/2890955.2890972

[9] Raffaele Bruno, Marco Conti, and Enrico Gregori. 2004. Throughput Evaluation and Enhancement of TCP Clients in Wi-Fi Hot Spots. In *Wireless On-Demand Network Systems*, Roberto Battiti, Marco Conti, and Renato Lo Cigno (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 73–86.

[10] Lucian Busoniu, Robert Babuska, and Bart De Schutter. 2008. A Comprehensive Survey of Multiagent Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38, 2 (2008), 156–172. https://doi.org/10.1109/TSMCC.2007.913919

[11] Furkan Canbal, Y. Bahadir Ozgun, M. Sukru Kuran, Ganesh Venkatesan, and Necati Canpolat. 2023. Wi-Fi QoS Management Program: Bridging the QoS Gap of Multimedia Traffic in Wi-Fi Networks. *IEEE Communications Magazine* (2023), 1–7. https://doi.org/10.1109/MCOM.003.2300264

[12] Pedro Casas, Michael Seufert, and Raimund Schatz. 2013. YOUQMON: A System for on-Line Monitoring of YouTube QoE in Operational 3G Networks. *SIGMETRICS Perform. Eval. Rev.* 41, 2 (aug 2013), 44–46. https://doi.org/10.1145/2518025.2518033

[13] Ayon Chakraborty, Shruti Sanadhya, Samir R. Das, Dongho Kim, and Kyu-Han Kim. 2016. ExBox: Experience Management Middlebox for Wireless Networks. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies* (Irvine, California, USA) *(CoNEXT '16)*. Association for Computing Machinery, New York, NY, USA, 145–159. https://doi.org/10.1145/2999572.2999597

[14] Nesrine Changuel, Bessem Sayadi, and Michel Kieffer. 2012. Online learning for QoE-based video streaming to mobile receivers. In *2012 IEEE Globecom Workshops*. 1319–1324. https://doi.org/10.1109/GLOCOMW.2012.6477773

[15] Xi Chen, Zonghang Li, Yupeng Zhang, Ruiming Long, Hongfang Yu, Xiaojiang Du, and Mohsen Guizani. 2018. Reinforcement Learning based QoS/QoE-aware Service Function Chaining in Software-Driven 5G Slices. arXiv:1804.02099 [cs.NI]

[16] Ioan-Sorin Comsa, Ramona Trestian, and Gheorghita Ghinea. 2018. 360° Mulsemedia Experience over Next Generation Wireless Networks - A Reinforcement Learning Approach. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*. 1–6. https://doi.org/10.1109/QoMEX.2018.8463409

[17] Der-Jiunn Deng, Shao-Yu Lien, Jorden Lee, and Kwang-Cheng Chen. 2016. On Quality-of-Service Provisioning in IEEE 802.11ax WLANs. *IEEE Access* 4 (2016), 6086–6104. https://doi.org/10.1109/ACCESS.2016.2602281

[18] Sandesh Dhawaskar Sathyanarayana, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. 2023. Converge: QoE-driven Multipath Video Conferencing over WebRTC. In *Proceedings of the ACM SIGCOMM 2023 Conference* (New York, NY, USA) *(ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 637–653. https://doi.org/10.1145/3603269.3604822

[19] Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taiga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, Aviral Kumar, and Rishabh Agarwal. 2024. Stop Regressing: Training Value Functions via Classification for Scalable Deep RL. arXiv:2403.03950

[20] G. Finnie, Canadian Radio-Television, Telecommunications Commission, and Heavy Reading (Organization). 2009. *ISP Traffic Management Technologies: The State of the Art : Report.* Canadian Radio Television and Telecommunications Commission. https://books.google.com/books?id=zUisDAEACAAJ

[21] Google 2024. WebRTC. https://webrtc.org/.

[22] DSP Group Graham Smith. 2008. 802.11 QoS Tutorial. https://ieee802.org/1/files/public/docs2008/avb-gs-802-11-qos-tutorial-1108.pdf.

[23] Gymnasium 2024. Gymnasium. https://gymnasium.farama.org/.

[24] Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double Q-Learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence* (Phoenix, Arizona) *(AAAI'16)*. AAAI Press, 2094–2100.

[25] Xiaoming He, Kun Wang, Huawei Huang, Toshiaki Miyazaki, Yixuan Wang, and Song Guo. 2020. Green Resource Allocation Based on Deep Reinforcement Learning in Content-Centric IoT. *IEEE Transactions on Emerging Topics in Computing* 8, 3 (2020), 781–796. https://doi.org/10.1109/TETC.2018.2805718

[26] Tobias Hobfeld, Lea Skorin-Kapov, Poul E. Heegaard, and Martin Varela. 2017. Definition of QoE Fairness in Shared Systems. *IEEE Communications Letters* 21, 1 (2017), 184–187. https://doi.org/10.1109/LCOMM.2016.2616342

[27] HPE Aruba Networking 2024. Aruba 550 Series Indoor Access Points. https://www.arubanetworks.com/products/wireless/access-points/indoor-access-points/550-series/.

[28] HPE Aruba Networking 2024. Aruba 7000 Series Mobility Controllers and Gateways. https://www.arubanetworks.com/products/wireless/gateways-and-controllers/7000-series/.

[29] HPE Aruba Networking 2024. HPE Aruba Networking Mobility Conductor. https://www.arubanetworks.com/resource/hpe-aruba-networking-mobility-conductor/.

[30] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A buffer-based approach to rate adaptation: evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (Chicago, Illinois, USA) *(SIGCOMM '14)*. Association for Computing Machinery, New York, NY, USA, 187–198. https://doi.org/10.1145/2619239.2626296

[31] Weixin Huang, Yuming Lin, Borong Lin, and Liang Zhao. 2019. Modeling and predicting the occupancy in a China hub airport terminal using Wi-Fi data. *Energy and Buildings* 203, Article 109439 (Nov. 2019), 109439 pages. https://doi.org/10.1016/j.enbuild.2019.109439

[32] Jasmina Baraković Husić, Adna Alić, Sabina Baraković, and Mladen Mrkaja. 2021. QoE Prediction of WebRTC Video Calls Using Google Chrome Statistics. In *2021 20th International Symposium INFOTEH-JAHORINA (INFOTEH)*. 1–6. https://doi.org/10.1109/INFOTEH51037.2021.9400661

[33] Eugene Ie, Chih wei Hsu, Martin Mladenov, Vihan Jain, Sanmit Narvekar, Jing Wang, Rui Wu, and Craig Boutilier. 2019. RecSim: A Configurable Simulation Platform for Recommender Systems. arXiv:1909.04847

[34] iperf 2024. iPerf3. https://iperf.fr/.

[35] Mona Jaber, Muhammad Ali Imran, Rahim Tafazolli, and Anvar Tukmanov. 2016. A Multiple Attribute User-Centric Backhaul Provisioning Scheme Using Distributed SON. In *2016 IEEE Global Communications Conference (GLOBECOM)*. 1–6. https://doi.org/10.1109/GLOCOM.2016.7841518

[36] Rajendra K Jain, Dah-Ming W Chiu, William R Hawe, et al. 1984. A quantitative measure of fairness and discrimination. *Eastern Research Laboratory, Digital Equipment Corporation, Hudson, MA* 21 (1984), 1.

[37] Hajime Kimura, Kazuteru Miyazaki, and Shigenobu Kobayashi. 1997. Reinforcement Learning in POMDPs with Function Approximation. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 152–160.

[38] Umakant Kulkarni, Yufeng Chen, Patrick Melampy, and Sonia Fahmy. 2023. Toward QoE-based Routing Path Selection. In *2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR)*. 114–119. https://doi.org/10.1109/HPSR57248.2023.10147938

[39] Umakant Kulkarni, Khaled Diab, Shivang Aggarwal, Lianjie Cao, Faraz Ahmed, Puneet Sharma, and Sonia Fahmy. 2023. Understanding the Impact of Wi-Fi Configuration on Volumetric Video Streaming Applications. In *Proceedings of the 2023 Workshop on Emerging Multimedia Systems* (New York, NY, USA) *(EMS '23)*. Association for Computing Machinery, New York, NY, USA, 41–47. https://doi.org/10.1145/3609395.3610599

[40] Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. 2016. Temporal Convolutional Networks: A Unified Approach to Action Segmentation. In *Computer Vision – ECCV 2016 Workshops*, Gang Hua and Hervé Jégou (Eds.). Springer International Publishing, Cham, 47–54.

[41] Xiao-Yang Liu, Ziyi Xia, Jingyang Rui, Jiechao Gao, Hongyang Yang, Ming Zhu, Christina Dan Wang, Zhaoran Wang, and Jian Guo. 2022. FinRL-Meta: Market Environments and Benchmarks for Data-Driven Financial Reinforcement Learning. *NeurIPS* (2022).

[42] Maestro project. 2025. Telemetry Data Items. https://gist.github.com/UmakantKulkarni/1fe2ac6f82fdc65d6e9e87e9ca1ebd83.

[43] Tarun Mangla, Emir Halepovic, Mostafa Ammar, and Ellen Zegura. 2018. eMIMIC: Estimating HTTP-Based Video QoE Metrics from Encrypted Network Traffic. In *2018 Network Traffic Measurement and Analysis Conference (TMA)*. 1–8. https://doi.org/10.23919/TMA.2018.8506519

[44] Tarun Mangla, Ellen Zegura, Mostafa Ammar, Emir Halepovic, Kyung-Wook Hwang, Rittwik Jana, and Marco Platania. 2018. VideoNOC: Assessing Video QoE for Network Operators Using Passive Measurements. In *Proceedings of the 9th ACM Multimedia Systems Conference* (Amsterdam, Netherlands) *(MMSys '18)*. Association for Computing Machinery, New York, NY, USA, 101–112. https://doi.org/10.1145/3204949.3204956

[45] Jake Martin and Nick Feamster. 2012. User-Driven Dynamic Traffic Prioritization for Home Networks. In *Proceedings of the 2012 ACM SIGCOMM Workshop on Measurements up the Stack* (Helsinki, Finland) *(W-MUST '12)*. Association for Computing Machinery, New York, NY, USA, 19–24. https://doi.org/10.1145/2342541.2342548

[46] Ricardo Matos, Nuno Coutinho, Carlos Marques, Susana Sargento, Jacob Chakareski, and Andreas Kassler. 2012. Quality of experience-based routing in multi-service wireless mesh networks. In *2012 IEEE International Conference on Communications (ICC)*. 7060–7065. https://doi.org/10.1109/ICC.2012.6364944

[47] Fatemeh Shah Mohammadi and Andres Kwasinski. 2018. QoE-Driven Integrated Heterogeneous Traffic Resource Allocation Based on Cooperative Learning for 5G Cognitive Radio Networks. In *2018 IEEE 5G World Forum (5GWF)*. 244–249. https://doi.org/10.1109/5GWF.2018.8516939

[48] C. Molnar. 2020. *Interpretable Machine Learning*. Leanpub. https://books.google.com/books?id=jBm3DwAAQBAJ

[49] Henrique D. Moura, Daniel F. Macedo, and Marcos A.M. Vieira. 2020. Wireless control using reinforcement learning for practical web QoE. *Computer Communications* 154 (2020), 331–346. https://doi.org/10.1016/j.comcom.2020.02.032

[50] Priya Narasimhan, Utsav Drolia, Jiaqi Tan, Nathan D. Mickulicz, and Rajeev Gandhi. 2015. The next-generation in-stadium experience (keynote). In *Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences* (Pittsburgh, PA, USA) *(GPCE 2015)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/2814204.2814205

[51] Ashkan Nikravesh, Qi Alfred Chen, Scott Haseley, Xiao Zhu, Geoffrey Challen, and Z. Morley Mao. 2018. QoE Inference and Improvement Without End-Host Control. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. 43–57. https://doi.org/10.1109/SEC.2018.00011

[52] Peach Open Movie Project 2024. Big Buck Bunny. https://media.xiph.org/video/derf//y4m/big_buck_bunny_1080p24.y4m.xz.

[53] Quantile Transformer 2024. QuantileTransformer: Transform features using quantiles information. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.QuantileTransformer.html.

[54] Miguel Ribeiro, Nuno Nunes, Valentina Nisi, and Johannes Schöning. 2022. Passive Wi-Fi monitoring in the wild: a long-term study across multiple location typologies. *Personal and Ubiquitous Computing* 26, 3 (01 Jun 2022), 505–519. https://doi.org/10.1007/s00779-020-01441-z

[55] M. Angeles Santos, Jose Villalon, and Luis Orozco-Barbosa. 2012. A Novel QoE-Aware Multicast Mechanism for Video Communications over IEEE 802.11 WLANs. *IEEE Journal on Selected Areas in Communications* 30, 7 (2012), 1205–1214. https://doi.org/10.1109/JSAC.2012.120806

[56] Ben Schneider, Rute C. Sofia, and Matthias Kovatsch. 2022. A Proposal for Time-Aware Scheduling in Wireless Industrial IoT Environments. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 1–6. https://doi.org/10.1109/NOMS54207.2022.9789864

[57] Markus Schutz. 2016. *Potree: Rendering Large Point Clouds in Web Browsers*. Master's thesis. Institute of Computer Graphics and Algorithms, Vienna University of Technology.

[58] M. Said Seddiki, Muhammad Shahbaz, Sean Donovan, Sarthak Grover, Miseon Park, Nick Feamster, and Ye-Qiong Song. 2014. FlowQoS: QoS for the Rest of Us. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking* (Chicago, Illinois, USA) *(HotSDN '14)*. Association for Computing Machinery, New York, NY, USA, 207–208. https://doi.org/10.1145/2620728.2620766

[59] Taveesh Sharma, Tarun Mangla, Arpit Gupta, Junchen Jiang, and Nick Feamster. 2023. Estimating WebRTC Video QoE Metrics Without Using Application Headers. In *Proceedings of the 2023 ACM on Internet Measurement Conference* (Montreal, Canada) *(IMC '23)*. Association for Computing Machinery, New York, NY, USA, 485–500. https://doi.org/10.1145/3618257.3624828

[60] Jaykumar Sheth and Behnam Dezfouli. 2019. Enhancing the Energy-Efficiency and Timeliness of IoT Communication in WiFi Networks. *IEEE Internet of Things Journal* 6, 5 (2019), 9085–9097. https://doi.org/10.1109/JIOT.2019.2927588

[61] Stephen M. Stigler. 1989. Francis Galton's Account of the Invention of Correlation. *Statist. Sci.* 4, 2 (1989), 73 – 79. https://doi.org/10.1214/ss/1177012580

[62] Xiaoming Tao, Chunxiao Jiang, Jie Liu, Ailing Xiao, Yi Qian, and Jianhua Lu. 2019. QoE Driven Resource Allocation in Next Generation Wireless Networks. *IEEE Wireless Communications* 26, 2 (2019), 78–85. https://doi.org/10.1109/MWC.2018.1800022

[63] Arryon D. Tijsma, Madalina M. Drugan, and Marco A. Wiering. 2016. Comparing exploration strategies for Q-learning in random stochastic mazes. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. 1–8. https://doi.org/10.1109/SSCI.2016.7849366

[64] Hai Anh Tran, Abdelhamid Mellouk, Said Hoceini, and Brice Augustin. 2012. Global state-dependent QoE based routing. In *2012 IEEE International Conference on Communications (ICC)*. 131–135. https://doi.org/10.1109/ICC.2012.6364307

[65] Dimitris Tsolkas, Eirini Liotou, Nikos Passas, and Lazaros Merakos. 2017. A survey on parametric QoE estimation for popular services. *Journal of Network and Computer Applications* 77 (2017), 1–17. https://doi.org/10.1016/j.jnca.2016.10.016

[66] WebRTC Samples 2024. WebRTC Samples. https://github.com/webrtc/samples.

[67] WebRTC Stats 2024. Identifiers for WebRTC's Statistics API. https://w3c.github.io/webrtc-stats/.

[68] Wireless NIC 2024. Intel Wi-Fi 6E AX210. https://www.intel.com/content/www/us/en/products/sku/204836/intel-wifi-6e-ax210-gig/specifications.html.

[69] WMM Traffic Management 2024. WMM Traffic Management. https://www.arubanetworks.com/techdocs/Instant_85_WebHelp/Content/instant-ug/voice-and-video/wmm-traffic-mgmt.htm.

[70] Yiran Xu, Rose Qingyang Hu, Lili Wei, and Geng Wu. 2014. QoE-aware mobile association and resource allocation over wireless heterogeneous networks. In *2014 IEEE Global Communications Conference*. 4695–4701. https://doi.org/10.1109/GLOCOM.2014.7037549

[71] Jiechen Yin, Yuming Mao, Supeng Leng, Xiang Wang, and Huirong Fu. 2015. QoE Provisioning by Random Access in Next-Generation Wireless Networks. In *2015 IEEE Global Communications Conference (GLOBECOM)*. 1–7. https://doi.org/10.1109/GLOCOM.2015.7417656

[72] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. *SIGCOMM Comput. Commun. Rev.* 45, 4 (aug 2015), 325–338. https://doi.org/10.1145/2829988.2787486

[73] Faqir Zarrar Yousaf, Olli Mämmelä, and Petteri Mannersalo. 2014. Reinforcement learning method for QoE-aware optimization of content delivery. In *2014 IEEE Wireless Communications and Networking Conference (WCNC)*. 3390–3395. https://doi.org/10.1109/WCNC.2014.6953124

[74] YouTube 2024. Talking Head. https://www.youtube.com/watch?v=hWTT4J_xNwY.

[75] YouTube Video 2024. YouTube Video. https://www.youtube.com/watch?v=X1oyKzWAwYU.

[76] Jia Zhang, Yixuan Zhang, Enhuan Dong, Yan Zhang, Shaorui Ren, Zili Meng, Mingwei Xu, Xiaotian Li, Zongzhi Hou, Zhicheng Yang, and Xiaoming Fu. 2023. Bridging the Gap between QoE and QoS in Congestion Control: A Large-scale Mobile Web Service Perspective. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. USENIX Association, Boston, MA, 553–569. https://www.usenix.org/conference/atc23/presentation/zhang-jia

[77] Xingyao Zhang, Haojun Xia, Donglin Zhuang, Hao Sun, Xin Fu, Michael B. Taylor, and Shuaiwen Leon Song. 2021. n-LSTM: Co-Designing Highly-Efficient Large LSTM Training via Exploiting Memory-Saving and Architectural Design Opportunities. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 567–580. https://doi.org/10.1109/ISCA52012.2021.00051

[78] Jianan Zhao, Shaoyi Xu, and Dongji Li. 2019. QoE Driven Resource Allocation in Massive IoT: A Deep Reinforcement Learning Approach. In *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*. 1–6. https://doi.org/10.1109/ICCW.2019.8756710

A LSTM Training Insights

Telemetry data is normalized using a Quantile Transformer [53] which scales all features to a uniform distribution between 0 and 1. This step standardizes metrics with varying scales and formats, making them compatible with the LSTM model and managing outliers. By aligning both instantaneous and cumulative telemetry data, normalization enhances the LSTM's ability to learn by treating all features equally. To maintain consistency, the normalization scaler is saved post-training and loaded during inference, ensuring that incoming telemetry data is processed using the same scale as during training.

The LSTM model leverages its recurrent neural network architecture to effectively capture temporal dependencies within the telemetry data. Through its gradient-based learning process, the model implicitly assigns weights to features based on their contribution to the QoE estimation task, enabling it to prioritize the most influential features. Table 3 presents the key features identified by the LSTM model for QoE estimation across all applications, using the permutation feature importance method [48].

Table 3. Key features for QoE estimation.

Feature	Description
Delay	End-to-end delay.
RX Unicast Data Frames	Number of unicast frames received.
TX Data Frames MCS-X	Total number of data frames transmitted at rate of MCS X.
Last SNR	Last recorded signal-to-noise ratio.
Last RX SNR	SNR of the last data packet received from the client.
Last ACK SNR	SNR of the last acknowledgment packet sent by the client.
Current Noise Floor	Residual background noise detected by an AP.
Channel busy 1/4/64 Sec	Percentage of time the radio channel was busy in the last 1/4/64 seconds.
PS State	Power-save state, showing if the AP/channel/link is in the awake or power-save state.
TX Retries	Number of packets that the AP had to resend to the client due to a transmission failure.
TX RTS Failed	Number of Ready To Send (RTS) frames that were not successfully transmitted.
Health	Quality of the link between the client and the radio.

B Components of DDQN

The Q-network denoted by $Q(s, a; \theta)$ is a neural network that approximates the action-value function, where s represents the state, a represents the action, and θ represents the network parameters. The target Q-network Q' is a copy of the Q-network that is periodically updated to stabilize the learning process and minimize potential overestimation of the Q values. During training, the Q-value is updated as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma Q'(s_{t+1}, \text{argmax}_{a'} Q(s_{t+1}, a'; \theta); \theta') - Q(s_t, a_t; \theta) \right], \quad \text{(B.2)}$$

where s_t is the current state at time t, a_t is the action taken at time t, a' is the action maximizing the next state's Q-value, r_t is the reward received at time t, s_{t+1} is the next state at time $t+1$, θ' is the target Q-network parameters, α is the learning rate, and γ is the discount factor. We select actions using Boltzmann exploration, which assigns probabilities to each action based on their Q-values as:

$$P(a) = \frac{e^{Q_t(s_t, a)/T}}{\sum_{i=1}^{n} e^{Q_t(s_t, a^i)/T}}, \quad \text{(B.3)}$$

where T is the temperature parameter that controls the exploration-exploitation trade-off. Prior studies showed that Boltzmann exploration can effectively balance exploration and exploitation [63].

Finally, the loss function used to train the Q-network is the mean squared error (MSE) between the predicted Q-values and the target Q-values:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\left(r_t + \gamma Q'(s_{t+1}, \text{argmax}_{a'} Q(s_{t+1}, a'; \theta); \theta') - Q(s_t, a_t; \theta)\right)^2\right]. \tag{B.4}$$

Table 4 lists the selected hyperparameters for the RL agents.

Table 4. Hyperparameters for DDQN agent.

Hyperparameter	Value
State size	257
Action size	6
Hidden layer size	128
Discount factor	0.99
Batch size	32
Optimizer	Adam
Learning rate	0.001
Replay buffer	10000

Table 5. Optimized RL reward weights and temperature value.

Application	w_{QoE}	w_{mean}	$w_{\text{deviation}}$	T
Video streaming	0.2	0.6	0.2	0.4
Video conferencing	0.2	0.2	0.6	0.6
File transfer	0.34	0.33	0.33	0.8
Audio conferencing	0.6	0.2	0.2	0.3

C Alternate Approaches

C.1 QoE Estimation

While we used LSTM for estimating QoE in Maestro, we also evaluated logistic regression, random forest, gradient boost, AdaBoost, k-neighbors, and decision tree models to see if any classification model can estimate the QoE with higher accuracy. To identify the best parameters for all the machine learning models, we used GridSearchCV with 5-fold cross-validation.

We use binning to convert the continuous QoE values ranging from 0 to 1 to discrete bins. We varied the number of bins from 5 to 10 and also evaluated with both equal-width bins and quantile-based bins but none of these binning schemes significantly improved the accuracy of the models. When we used 3 bins, accuracy was above 90%. However, a few bins can be insufficient for fine-grained resource allocation in Wi-Fi networks, which necessitates a larger number of bins.

Examining the performance of classification models for estimating QoE in Table 6, we found that these models were not able to accurately estimate the QoE, with the highest reported accuracy being only 63%. As a result, we shifted focus to regression models to enhance prediction accuracy and avoid the cumulative errors associated with binning and discrete classifications.

In Table 6, we also include the mean absolute error (MAE) and mean squared error (MSE) for regression-based models random forest, gradient boost, linear regression, AdaBoost, decision tree and compare their performance to that of the LSTM model. Additionally, we explored other neural networks such as Artificial Neural Networks (ANN) and Temporal Convolutional Networks (TCN) [40]. TCNs, similar to LSTMs, are designed to handle sequential data and capture temporal dependencies. However, we found LSTM to be the most effective, achieving the lowest mean absolute error (0.21) and mean squared error (0.10) on the test dataset.

C.2 Reinforcement Learning

While we use DDQN in Maestro, we also explored other RL methods for the Wi-Fi resource allocation problem. The original Deep Q-Network (DQN) algorithm tends to overestimate Q-values, resulting in suboptimal policies that can negatively impact performance. This overestimation occurs because DQN uses the same network to select and evaluate actions, which can lead to biased updates.

Policy gradient methods, such as REINFORCE (Monte Carlo Policy Gradient), directly optimize the policy by adjusting actions to maximize expected rewards. Although these methods can effectively

Table 6. Performance of different ML models for estimating QoE.

Classification Model	Accuracy (%)	F_1 Score (%)
Logistic Regression	58	53
Random Forest	63	58
Gradient Boosting	63	59
AdaBoost	20	22
K-Nearest Neighbors	58	56
Decision Tree	62	59

Regression Model	Mean Square Error	Mean Absolute Error
Random Forest	0.14	0.27
Gradient Boosting	0.17	0.31
Linear Regression	0.12	0.26
AdaBoost	0.13	0.27
Decision Tree	0.14	0.27
LSTM	**0.10**	**0.21**

learn complex policies, they often struggle with high variance and require a substantial number of samples to converge. In the dynamic setting of Wi-Fi networks, where conditions and usage patterns change rapidly, gathering enough data for training can be challenging, making these methods less practical.

Actor-critic methods, like Advantage Actor-Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C), attempt to combine the strengths of both value-based and policy-based methods. These methods use a critic to estimate the value function and guide the actor's policy updates, aiming to reduce variance and improve stability. However, the dual-network structure makes the training process more complex and computationally demanding, which can be a drawback in the fast-paced environment of Wi-Fi networks. In comparison, DDQN, with its use of feedforward neural network, target networks and experience replay, offers a more straightforward and efficient approach for managing the high-dimensional state-action space and dynamic conditions of Wi-Fi resource allocation.

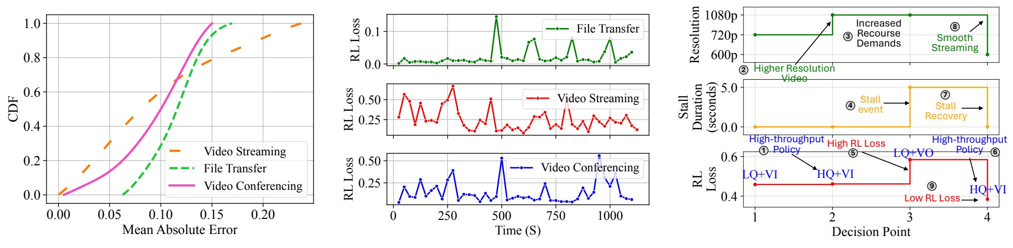
D Maestro Operation Closeup

The QoE estimator and RL agent components collectively enhance QoE. We now take a look at each of the two components individually: we analyze the prediction error of the QoE estimator, and the loss metrics of RL agents. Additionally, we provide an example to illustrate how Maestro reacts to dynamics in the environment and generates corresponding resource allocation decisions leading to higher QoE.

Figure 12(a) presents the CDF for the mean absolute error (MAE) between the actual and estimated QoE across the three local applications, computed from data from the experiments in Section 3.5.1. The results show that the maximum MAE is approximately 0.2, with 90% of the absolute errors being less than 0.15. This demonstrates the high accuracy of the LSTM-based QoE estimator. The slight imprecision in the QoE does not substantially hinder the effectiveness of the orchestrator as it does not rely solely on the QoE estimate, but it also considers the full telemetry state vector as part of its decision-making process. This means that even with an imperfect QoE estimate, the agent uses comprehensive system information to make robust policy choices that optimize resource allocation and client QoE. Figure 12(b) depicts the RL loss metric of three local applications. The losses exhibit variability and remain close to zero, indicating accurate predictions.

To observe how Maestro adapts to the changing conditions, we present a snapshot of Maestro operation in Figure 12(c). We focus on four decision points and the corresponding values of RL loss, stall duration and video resolution due to the selected actions (annotated in blue).

At *Decision Point 1*, the RL agent assigns a low priority queue with the video access category to the video streaming application. This policy results in no stalls and sets the video resolution to 720p. At *Decision Point 2*, the agent determines that the application can achieve a higher QoE by switching to a higher priority queue with the video access category. This change allows the application to increase the video resolution from 720p to 1080p without any stalls. At *Decision Point 3*, the agent opts for a low priority queue with the voice access category. This decision, combined with a sudden change in the environment–specifically, increased resource demands from



(a) MAE for real vs. estimated QoE (b) Client RL loss (c) Video streaming behavior

Fig. 12. Maestro operation closeup.

other applications–leads to a stall duration of 5 seconds when the application attempts to fetch high-quality chunks. The policy agent detects this stall through the increase in RL loss caused by the corresponding state-action combination. To recover from this loss and improve the client QoE, the RL agent decides to move the application flow to a high priority queue with the video access category at *Decision Point 4*. This eliminates stalls and continues the smooth flow of the video stream as shown in Figure 12(c).

Contrasting this to the WMM policy where all video streaming flows are assigned the video access category, we observe resource contention which slows down the video streaming application. This cycle leads to repeated stalls, as applications request higher-quality chunks when they gain channel access but fail to maintain this quality due to subsequent access denials.

E QFlow Implementation

QFlow [7] is a policy framework that directly utilizes real-time QoE feedback from client devices to make precise policy decisions. Unlike Maestro, which estimates QoE through machine learning models to navigate partial observability, QFlow benefits from accessing direct, accurate QoE metrics, giving it a more complete view of application quality in real time. The QFlow action space consists of only two actions: select a low-priority queue and select a high-priority queue. The specific access category used for video streaming applications such as YouTube is not disclosed, so we assume it to be Best Effort.

For a fair comparison with Maestro, we extended the evaluation to include several open-source multimedia applications, not just YouTube. This expansion required adapting QFlow's state variables to align with the characteristics of other application classes. For video streaming, we retained state variables such as buffer status, quality, interruptions, and startup delay. For video conferencing, we defined new state variables, including audio jitter, audio packet loss, and the number of video packets sent and received. File transfer states were based on throughput and bytes transferred, while audio conferencing used audio jitter and packet loss. We adhered to the methodology outlined in Section 2.3 for training QFlow's RL agents.

A key challenge in this process was addressing QFlow's scalability when accommodating varying numbers of clients and types of application classes. QFlow aggregates the state of all clients into a combined system state for decision-making. This approach means that as the number of clients and the diversity of applications increase, the combined state size changes dynamically. For example, the state size may be eight for two video streaming clients, but it could expand to ten for three clients running video streaming, conferencing, and file transfer applications.

In summary, while QFlow can be extended beyond YouTube, it is less scalable due to the exponential growth of the combined state size with increasing numbers of clients and application diversity.

Received June 2024; revised November 2024; accepted December 2024