# Practical Latency-aware Scheduling for Low-latency Elephant VR Flows in Wi-Fi Networks

Shao-Jung Lu, Wei-Xun Chen, Yu-Shao Su,
Yu-Shou Chang, Yao-Wen Liu, Chi-Yu Li
*Computer Science, National Yang Ming Chiao Tung University*
Hsinchu City, Taiwan, ROC
nctu@louie.lu, {chrissy81527, shao.cs09g, ljk25679}@nctu.edu.tw,
{yaowenliu.cs07, chiyuli}@nycu.edu.tw

Guan-Hua Tu
*Computer Science and Engineering*
*Michigan State University*
East Lansing, MI, USA
ghtu@msu.edu

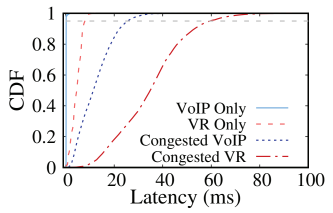
***Abstract*—Virtual reality (VR) applications are increasingly popular. With high-quality video streams and interactive content, they require both low-latency and high-bandwidth performance demands on the communication from the edge-based VR server to the VR headsets. Although most VR headsets are equipped with dedicated wired or wireless modules connected to the VR server, using common Wi-Fi networks to support them can be a promising trend due to convenience and low cost. However, current Wi-Fi Access Points (APs) cannot meet latency demands of low-latency elephant VR flows, especially in traffic congestion cases. We thus design a practical Wi-Fi scheduling solution, designated as LAST-PQ (Latency-Aware Scheduler with Two-level Priority Queueing), to support VR flows at the Wi-Fi AP. It monitors the runtime latency performance of VR flows while prioritizing scheduling for urgent flows, whose latency demands are at risk of violation. We implement LAST-PQ in Linux on a commodity Wi-Fi platform using an open-source Wi-Fi driver; it is compliant to the current Wi-Fi scheduling framework. The evaluation result shows that it can reduce latency by up to 79.89% in various congested scenarios; moreover, it consistently meets the latency demands of VR flows in cases of mobility at runtime.**

***Index Terms*—Wi-Fi, low latency, VR, scheduling**

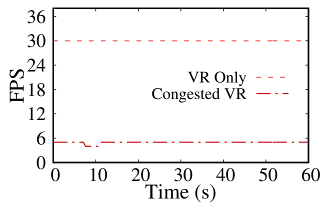
## I. Introduction

Virtual reality (VR) applications are increasingly popular in recent years [1], with more stringent demands than conventional applications on the communication performance between the VR server and headset. Specifically, their high-quality video streams with real-time interactive content require joint low-latency and high-bandwidth performance. To ensure the performance, wired VR devices (e.g., Oculus Rift) and those with dedicated wireless modules (e.g., HTC Vive) are the mainstream VR platforms.

Although the Wi-Fi network has been a common network service deployed everywhere, there are still few wireless VR devices relying on the common Wi-Fi network. The main reason is that VR performance can be easily affected by Wi-Fi traffic congestion. However, enabling common Wi-Fi networks to support VR applications benefits both VR vendors and users, saving the cost of dedicated VR wired/wireless modules and providing more convenience when using VR applications with ubiquitous Wi-Fi networks.



(a) Wi-Fi per-packet latency.



(b) VR frames per second.

Fig. 1: Comparing an elephant VR flow with a mouse VoIP flow in non-congested and congested cases.

We thus examine the VR performance on the Wi-Fi network. We conduct a case study by streaming a VR traffic flow from a VR server to a smartphone's VR application through Wi-Fi. The Wi-Fi Access Point (AP) connects to the VR server via a wired network and the smartphone, which serves as an emulated headset, through Wi-Fi. As the VR server is deployed at the Wi-Fi edge, the per-packet latency at the Wi-Fi AP dominates the overall latency performance. Therefore, we examine the latency values for the VR stream. We compare a VR flow with a conventional VoIP flow in non-congested and congested cases. Both flows are streamed to the same smartphone. The non-congested case does not have any other traffic, whereas in the congested case, there are two additional downlink TCP flows at the smartphone, along with two clients, each with one downlink TCP flow, in the same Wi-Fi network. More detailed settings are provided in Section III.

We observe that the Wi-Fi AP cannot satisfy the elephant VR flow's latency demand[1] in the congested case. In the non-congested case, as shown in Figure 1, the VR and VoIP flows exhibit Wi-Fi latency as small as 8.11 ms and 0.19 ms, respectively, at the 95th percentile. However,

[1]In this paper, VR and VoIP flows are designated as elephant and mouse flows, respectively. The former usually has a requirement of several tens of Mbps, while the latter requires only below 1 Mbps.

in the congested case, their latency values increase to 60.37 ms and 25.97 ms, respectively. Such large latency increase prevents the VR flow from achieving the required number of frames per second (FPS), i.e., 30; its average FPS is only 4.94 during the traffic congestion. Thus, the Wi-Fi AP can indeed offer low latency to the mouse VoIP flow, but not to the elephant VR flow, which has 45 Mbps demand with 1920x1080 resolution in the experiment. The root cause is that the current Wi-Fi scheduling framework neither prioritizes low-latency elephant flows nor satisfies low-latency demands at runtime. In Section III, we conduct a comprehensive case study to examine why current Wi-Fi networks cannot adequately support VR flows.

In this work, we design a practical Wi-Fi scheduling solution, designated as LAST-PQ (Latency-Aware Scheduler with Two-level Priority Queueing), to support low-latency elephant VR flows at the Wi-Fi AP. It is compliant with the current Wi-Fi scheduling framework and ready for use. Additionally, it is latency-aware, allowing it to satisfy VR latency demands configured from the user space at runtime. The core concept of LAST-PQ is monitoring per-packet latency statistics and performing prioritized scheduling whenever there is a potential demand violation. It consists of three main components: two-level priority queueing, cross-layer delay controller, and latency-aware scheduler. With the priority queueing, LAST-PQ can prioritize each VR flow from the current Wi-Fi scheduling framework at both flow and device levels. The delay controller in the user space monitors latency statistics, dynamically adapting each VR flow's permitted queueing delay using a window-based delay gradient algorithm. Collaborating with the delay controller, the latency-aware scheduler in the kernel space identifies `urgent` VR flows for prioritized scheduling.

We implemented and evaluated LAST-PQ on a Linux-based PC with a Qualcomm Atheros card using the open-source ath9k driver. The evaluation considers both static and mobility cases, and the enhanced distributed channel access (EDCA) mechanism enhanced by LAST-PQ, under traffic congestion. Results demonstrate that LAST-PQ can satisfy a VR flow's demand in most cases, reducing latency by up to 79.89% in the most congested scenario. It consistently meets latency demands in mobility cases at runtime. Furthermore, LAST-PQ enhances the EDCA to be latency-aware while effectively supporting multiple VR flows with different latency demands.

The rest of this paper is organized as follows. Section II presents the current Wi-Fi scheduling architecture on Linux, and Section III conducts a case study on the VR performance in Wi-Fi networks. We then design, implement, and evaluate LAST-PQ in Sections IV, V, and VI, respectively. Section VII presents related work and Section VIII discusses issues about LAST-PQ. Finally, Section IX concludes the paper.

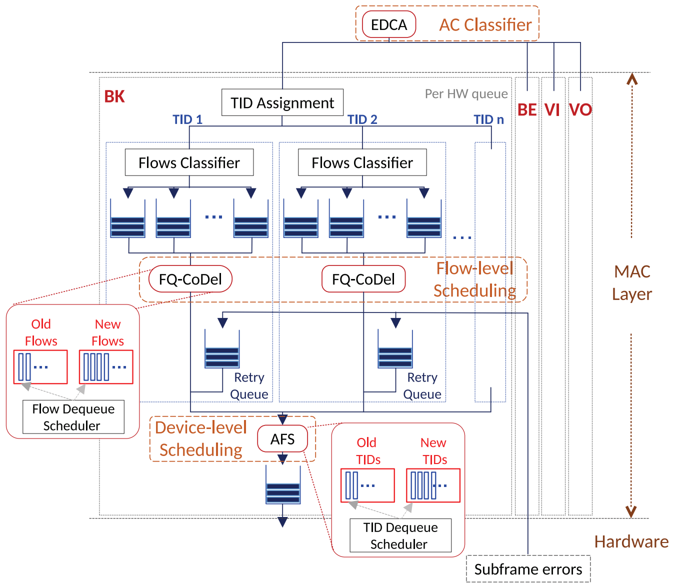


Fig. 2: Wi-Fi packet scheduling architecture on Linux.

## II. Wi-Fi Packet Scheduling Architecture

In this section, we present the Wi-Fi packet scheduling architecture employed on Linux-based APs. As shown in Figure 2, it comprises three major components from top to bottom: access category (AC) classifier, flow-level scheduling, and device-level scheduling. These components are currently enabled by updated methods in the Linux kernel: EDCA [2], fair queueing controlled delay (FQ-CoDel) [3], and airtime fairness scheduling (AFS) [4], respectively.

**AC Classifier.** EDCA defines four ACs in increasing order of traffic priority: background (BK), best effort (BE), video (VI), and voice (VO). Higher priority is achieved by adjusting two parameters: contention window (CW) boundary and arbitration inter-frame space (AIFS). Smaller values for these parameters allow faster data packet delivery. The CW boundary gives a range of time slots from which a station randomly selects a number to wait for the next transmission when the exponential back-off algorithm is executed. The AIFS is the time period that a station has to wait before it is allowed to transmit data. The smaller the CW boundary or the shorter the AIFS is used, the faster data packets can be delivered. To configure a high-priority AC for data packets, an application can specify the corresponding tag of the differentiated services codepoint (DSCP) in their IP headers.

At the MAC layer, each AC is associated with a set of traffic identifier (TID) queues, with each TID queue dedicated to one connected station. Figure 2 shows the scheduling architecture for the BK, to which those of the other ACs are similar. The TID queues of each AC are handled by the same hardware queue with the AC's parameters. Each TID queue has multiple flow queues where flows are classified based on flow information (e.g., IP address and port number). When a packet reaches the MAC layer, it is classified to an AC based on its DSCP tag, passed to a TID queue based on its destination station, and finally enqueued to a flow queue, ready to be scheduled by the FQ-CoDel.

**Flow-level Scheduling.** FQ-CoDel [3] schedules packets among the queues of each TID for transmission; each packet queue corresponds to a specific flow. FQ-CoDel is built on top of CoDel [5], a simple active queue management (AQM) algorithm designed to control bufferbloat-generated excess delay. It provides two major functions: ensuring flow fairness and preventing starvation of sparse flows (e.g., VoIP). Flow fairness is achieved through a byte-based deficit round-robin (DRR) scheduler. For starvation prevention, FQ-CoDel utilizes a two-tier queue structure with old and new flow queues to prioritize sparse flows. It schedules flows from the new queue with higher priority; flows in the old queue are scheduled only when the new queue is empty.

Sparse flows typically appear in the new queue to obtain high priority. This is because the flow is inserted into the new queue when the flow's queue state transitions from empty to non-empty, a situation that commonly occurs with sparse flows. The flow inserted into the new queue is assigned a default byte-based deficit value for DRR scheduling. This deficit value decreases as delivered bytes increase. In the new queue, when a non-empty flow becomes empty or has a negative deficit, it is transferred to the old queue.

**Device-level Scheduling.** AFS schedules downlink transmission for connected stations at the AP, considering airtime fairness [4] among TIDs. Each TID is associated with one station in each AC. It addresses a wireless anomaly [6] where slow stations may consume more airtime than fast stations, thus reducing overall bandwidth. AFS adopts two queues, old and new queues, to schedule TIDs, with operations similar to FQ-CoDel. The only difference is that the deficit is counted based on airtime instead of bytes. When a TID is scheduled for transmission, AFS requests the flow dequeue scheduler of the TID to dequeue packets based on the FQ-CoDel operation.

## III. Case Study on Elephant VR Flows

In this section, we assess the performance of elephant VR flows in Wi-Fi networks, considering both non-congested and congested cases, along with the impact of the EDCA.

**Experimental Setting.** We set up a Wi-Fi AP and connect two types of clients to it. The AP is based on IEEE 802.11n, utilizing hostapd 2.7 on an x86 PC running Linux kernel 4.2. It is equipped with a Qualcomm Atheros AR9280 wireless card. It operates on the 5GHz frequency, specifically on channel 132, employing a 3x3 MIMO antenna, and has a 40 MHz bandwidth. It uses the Linux default rate adaptation mechanism, Minstrel. We chose the traditional 802.11n AP platform because newer Wi-Fi AP platforms (e.g., 802.11ac/ax) have implemented a portion of the packet scheduling architecture in the Wi-Fi NIC firmware, which is not open-source. As a result, they do not allow the collection of latency results or modifications to the scheduling architecture.

The connected clients include 2 smartphones and 5 embedded Wi-Fi devices. The smartphones, which include the ASUS Zenphone 6 and Google Pixel 3, serve as VR clients. The embedded Wi-Fi devices are used to receive background (non-VR) downlink traffic. They are built based on the commercial 802.11ac AP, TP-Link C2600, by enabling the client mode with OpenWrt 18.06 [7].

We built two VR servers using PCs running Windows 10 and connected them to the AP via Ethernet. Each VR server is set up with two software programs, Steam VR [8] and Trinus VR [9]. Steam VR can provide VR content on various hardware platforms, while Trinus VR can turn smartphones into VR headsets. The Trinus VR application is installed on the smartphones to receive VR flows from the Trinus VR server. In a common setting with 30 FPS and a resolution of 1920x1080, a VR flow requires 45 Mbps of network bandwidth. These flows are delivered in the BK category unless explicitly specified. For background non-VR traffic, TCP flows are generated to Wi-Fi clients from a PC connected to the AP via Ethernet. Each experiment consists of 5 runs, each lasting 20 seconds. We focus on latency performance at the 95th percentile by collecting per-packet latency statistics (see implementation in Section V). Notably, per-packet latency is the period from the time a packet starts to be queued at the MAC layer to the time the packet's acknowledgement (ACK) is received.

All the clients are located close to the AP with good channel conditions. The physical data rates are fixed at 162 Mbps for smartphones with two antennas and 405 Mbps for embedded devices with three antennas, respectively. This static setting allows us to focus on Wi-Fi scheduling issues for elephant VR flows while minimizing the impact of rate adaptation and wireless channel dynamics. Once any scheduling issues are identified in this static scenario, they can only be aggravated by those dynamics.

Note that, although the Trinus VR application displays FPS information on its GUI panel, this data cannot be recorded as FPS statistics during experiments. We use a Python library called `memorpy` to trace the memory location where the Trinus VR application stores FPS values. Finally, we capture that memory address and record FPS values over time.

### *A. Non-congested Case: VR Flow Only*

We initially examine per-packet latency performance for a single VR flow in a non-congested scenario, without any other traffic. To account for different bandwidth demands, we vary its resolution from low (1152x648) to high (1366x770) and ultrahigh (1920x1080).

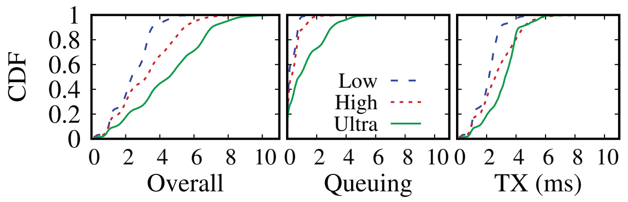


Fig. 3: Non-congested case: per-packet latency of a single VR flow with three different resolutions.

Based on our measurements, these resolutions require 19.97/25.57/45.38 Mbps bandwidths, respectively. In addition to overall per-packet latency, we decompose it into queueing and transmission latency, as shown in Figure 3. It is observed that latency increases with the resolutions of VR flows.

Specifically, at the 95th percentile of overall latency, the low, high, and ultrahigh resolution settings result in 4.29 ms, 6.07 ms, and 7.98 ms, respectively. Furthermore, latency values for over 99% packets in all settings are below 10 ms, meeting the requirements for most VR flows [10], [11]. While it can be anticipated that queueing delays may increase with other coexistent traffic flows, prioritizing the VR flow over others can ensure that the required latency is still satisfied.

### *B. Congested Cases: Multi-flow and Multi-client*

We consider two congested cases: multi-flow and multi-client, to examine the flow-level (i.e., FQ-CoDel) and device-level (i.e., AFS) scheduling mechanisms, respectively. In the multi-flow case, there is only one client, where a VR flow coexists with multiple data flows. In the multi-client case, there are not only one VR client with a VR flow and multiple data flows, but also multiple non-VR clients, each of which contains a data flow.

**Multi-flow Case.** We compare the latency performance of the elephant VR flow with that of a mouse VoIP flow when multiple data flows coexist with them. We vary the congestion level by considering 0, 1, and 3 background data flows. The overall and queueing latency results of the VR and VoIP flows are shown in Figures 4a and 4b, respectively. Almost all VoIP packets have latency below 10 ms. In contrast, for VR flows coexisting with 1 and 3 background flows, packet latency significantly increases. For example, the 95th latency increases from 7.85 ms in the case of no background flow to 19.87 ms and 39.97 ms, respectively.

It is further observed that VR flows in the cases of 0 and 1 background flow can consistently deliver 31 FPS while achieving an average throughput of 45 Mbps, as shown in Figures 4c and 4d. However, for the VR flow in the case of 3 background flows, FPS and average throughput decrease to 21.78 and 30.95 Mbps, respectively.

The significant increase in latency is primarily attributed to queueing latency. This is because FQ-CoDel fairly schedules VR and multiple background flows, and an increase in background flows causes VR packets to be queued for a longer duration. It is observed that FQ-CoDel can indeed ensure small latency for mouse VoIP flows, which are mostly in the new queue with higher priority than the old queue. However, most VR packets experience the old queue since, for most of the time when new packets arrive, the packet queue of each elephant VR flow is not empty, thus consistently remaining in the old queue.

**Multi-client Case.** We vary the congestion level in the multi-client case by considering three scenarios: 3 non-VR clients with 0 data flows at the VR client (tagged as 3C), 3 non-VR clients with 3 data flows at the VR client (tagged as 3F3C), and 4 non-VR clients without data flows at the VR client (tagged as 4C). As shown in Figure 5a, all three cases significantly increase the latency of the VR flow, and none of them can sustain its normal operation. For instance, they increase the 95th latency from 7.85 ms in the VR-only case to 36.96 ms, 48.54 ms, and 59.38 ms, respectively, exceeding the VR flow's required low-latency demand of 20 ms. The root cause is that AFS fairly schedules airtime among multiple clients, as shown in Figure 5b, where the values for all cases are almost 1 in terms of Jain's fairness index. When the VR client coexists with other clients, the fairly shared airtime may not be sufficient for the VR flow, thereby increasing latency.

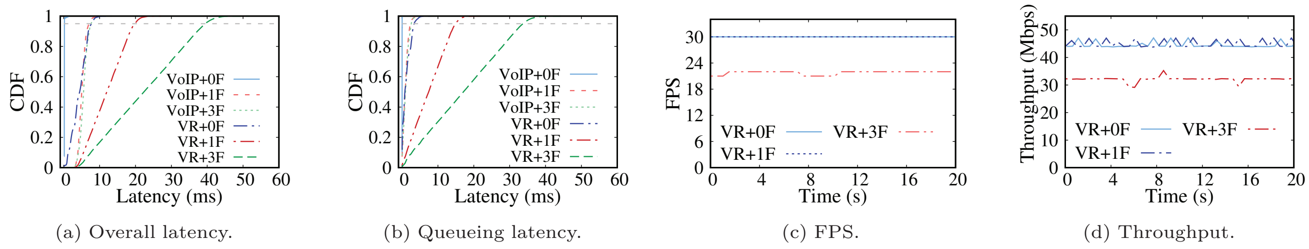
*The current joint flow-level and device-level scheduling framework can effectively support low-latency mouse flows while achieving fairness for coexistent flows and devices, but it has poor support for low-latency elephant VR flows.*

### *C. Does EDCA Work?*

EDCA can indeed enhance the latency performance of the elephant VR flow tagged with the VI category, but it only prioritizes its channel access. It may still face challenges when multiple latency-sensitive flows coexist. Therefore, we investigate multi-flow and multi-client scenarios where all flows are latency-sensitive and tagged with the VI category.

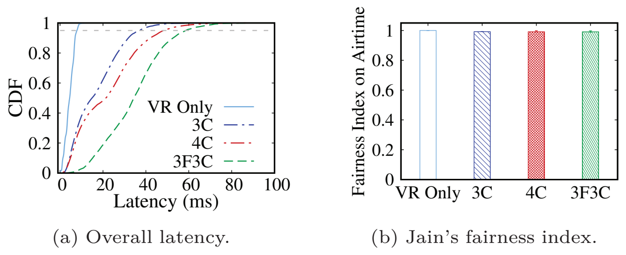
**Multi-flow Case.** We generate two VR flows and three TCP flows using the Trinus and iPerf software programs, respectively, to a single smartphone. In practice, different kinds of application flows may have varying latency requirements. In addition to the latency requirement of 20 ms for the VR flows, we assume that the TCP flows with 50 Mbps have a less stringent requirement of 30 ms.

Figure 6a shows the latency performance of these five flows scheduled in the VI category. It is observed that the 95th percentile latency values of the TCP flows are all below 20 ms, satisfying their requirement. However, for the VR flows, these values are 26.89 ms and 27.80 ms, respectively, failing to meet the 20 ms requirement. The better latency performance of the TCP flows can be attributed to the different traffic patterns generated by the Trinus and iPerf.



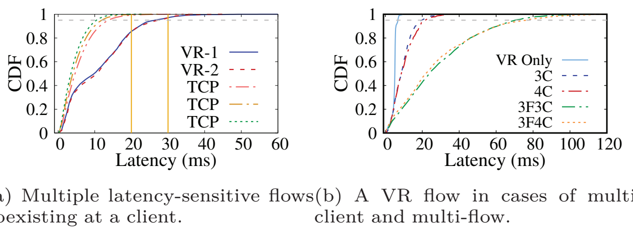
(a) Overall latency. (b) Queueing latency. (c) FPS. (d) Throughput.

Fig. 4: Performance of VR and VoIP flows in multi-flow cases (+$x$F stands for $x$ background flows).



(a) Overall latency. (b) Jain's fairness index.

Fig. 5: Performance of elephant VR flows in multi-client cases ($x$C stands for $x$ non-VR clients).



(a) Multiple latency-sensitive flows coexisting at a client. (b) A VR flow in cases of multi-client and multi-flow.

Fig. 6: Enabling EDCA for all latency-sensitive flows in the VI category.

It is evident that EDCA assigns the same priority setting to all flows within the same category, without considering their varying latency demands. The demands of latency-sensitive flows could likely all be satisfied with a latency-aware scheduler. In this case, VR flows could be prioritized over TCP flows, which do not require low latency performance. The latency-aware scheduler can manage the precedence among latency-sensitive flows over time based on their latency requirements, thus accommodating all demands.

**Multi-client Case.** We then examine the multi-client case with four different scenarios: 3 and 4 non-VR clients without any data flows on the VR client, and 3 and 4 non-VR clients with 3 data flows on the VR client. As shown in Figure 6b, the latency performance of the VR deteriorates with an increasing number of non-VR clients and data flows. Although all flows are in the VI category, EDCA treats them equally with only higher priority on the channel access. Consequently, the VR flow may still suffer from traffic congestion without being prioritized based on its latency requirement.

## IV. LAST-PQ Design

We design a closed-loop scheduling solution, LAST-PQ, to meet the latency demands of elephant VR flows in Wi-Fi networks. It schedules the transmission of each elephant VR flow based on its runtime latency statistics, considering its latency demand, denoted as $ld_{i,\beta}$, which indicates the packet latency for flow $i$ at a certain percentile $\beta$. The latency demand can be configured by the VR user or application through a given interface. Notably, most VR application servers are deployed at the network edge (e.g., next to the Wi-Fi AP) to provide stable low-latency, high-throughput performance. Therefore, Wi-Fi latency can be considered as the major performance bottleneck.

LAST-PQ adaptively ensures that the $\beta$th packet latency remains below the specified latency demand. The latency may be influenced by time-varying channels, user mobility, and dynamic traffic. Instead of consistently prioritizing latency-sensitive flows, LAST-PQ prioritizes them only when their demands are on the verge of being violated. This approach minimizes the impact of other traffic flows, allowing latency-sensitive flows to meet their demands without receiving excessive precedence. Since LAST-PQ satisfies latency-sensitive flows by suppressing other latency-insensitive flows at the same AP, they can be satisfied in congested cases only when their latency demands can be essentially met in the absence of any other flows.

To achieve this, three major challenges arise as follows. First, the existing two-level scheduling framework equipped with the FQ-CoDel and AFS mechanisms has evolved to address many issues, so LAST-PQ should retain this framework while giving priority to VR flows. Second, LAST-PQ needs to determine potential demand violations over time by obtaining statistics of packet latency from the kernel driver and comparing them with the corresponding latency demands configured from the user space. Third, LAST-PQ should prioritize VR flows for transmission by considering potential demand violations at runtime.

We propose three major components in LAST-PQ to address the above three challenges, respectively, as illustrated in Figure 7: two-level priority queueing, cross-layer delay controller, and latency-aware scheduler. The two-level priority queueing incorporates a prioritized flow queue and a prioritized TID queue into the flow-level and device-level schedulers, respectively, and lets them operate

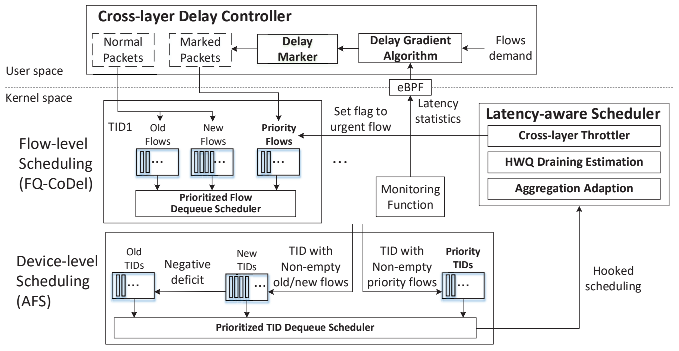


Fig. 7: LAST-PQ solution architecture.

on top of the original scheduling operation. The cross-layer delay controller dynamically adapts a permitted queueing latency for each VR flow to prevent potential demand violations in a closed-loop fashion. It considers both the latency demand and the feedback of latency statistics. The packets belonging to each VR flow are marked with high priority and the permitted queueing latency.

Based on the permitted queueing latency, below which the sum of the software and hardware queueing delays shall be maintained, the latency-aware scheduler determines whether VR packets should take precedence at runtime. This determination is made by considering the software queueing latency that the VR flow's queue has experienced and the estimated hardware queueing latency that will be experienced by the flow's next transmission. VR packets that are anticipated to violate the permitted queueing latency are prioritized, and their corresponding flow queues are then tagged as `urgent`, prioritizing them for transmission.

Notably, LAST-PQ focuses on enhancing the latency performance of downlink VR flows at the Wi-Fi AP. While uplink control packets are crucial for the interactive functions of VR applications, they inherently receive higher priority for channel access due to their small traffic volume, resulting in lower latency. We will now present each design component.

**Two-level Priority Queueing.** We introduce a prioritized queue into both the flow-level and device-level scheduling components. Each of these parts operates on top of a two-tier queue structure, which includes old and new queues. In each part, a packet queue for each flow marked with high priority is inserted into the prioritized flow queue, in addition to being placed in the old or new queue based on the original FQ-CoDel operation. Similarly, each TID with a non-empty prioritized flow queue is placed in the prioritized TID queue while also remaining in the old or new TID queue according to the default AFS mechanism. Therefore, the prioritized flow and TID queues are designated to hold the VR flows and their corresponding TIDs, allowing each VR flow to be prioritized for transmission. Simultaneously, these queues remain involved in the FQ-CoDel and AFS scheduling operations. Importantly, if a VR flow's latency demand can be consistently satisfied by the default scheduling operation (e.g., in the absence of other traffic flows), the prioritized transmission will not be triggered.

The two-level operation continues its sequence from bottom to top, now incorporating a prioritized scheduling action before proceeding with its standard operation on the old and new queues. When presented with a transmission opportunity at the device-level scheduling, the prioritized TID dequeue scheduler selects a prioritized TID with any `urgent` flow queue. Subsequently, the flow scheduler dequeues packets from that flow queue. In cases where multiple prioritized TIDs each have at least one `urgent` flow queue, or a prioritized TID has multiple `urgent` flow queues, the selection is based on the one with the greatest deficit value at the device-level or flow-level scheduling, respectively. If there is no prioritized TID, the schedulers revert to their original operation.

**Cross-layer Delay Controller.** The controller provides an interface in the user space for VR applications to prioritize VR flows based on flow information with specified latency demands (i.e., $ld_{i,\beta}$). For each VR flow $i$, the controller adapts a permitted queueing latency $l_{i,pq}$ using a window-based delay gradient algorithm. This algorithm takes into account the flow's latency demand and the latency statistics collected by a monitoring function per time window $t_w$ (20 ms in our implementation) in the kernel space. The latency statistics are transmitted from the kernel space to the user space via the extended Berkeley Packet Filter (eBPF) function, while the permitted queueing latency is communicated to the kernel space through the socket buffer.

At each time $t$, the $\beta$th latency, $l_{i,\beta}(t)$, is collected. Given a VR flow's latency demand $ld_{i,\beta}$, we maintain the runtime latency, $l_{i,\beta}(t)$, below this demand by adjusting $l_{i,pq}$. The rationale is that when $l_{i,\beta}(t)$ is approaching or exceeding $ld_{i,\beta}$, the VR packets should spend less time in the queue, necessitating a decrease in $l_{i,pq}$. Conversely, when $l_{i,\beta}(t)$ is too small, $l_{i,pq}$ is increased, ensuring the flow receives precedence minimally without disproportionately impacting other traffic flows.

We thus develop a window-based delay gradient algorithm. As shown in Figure 8, the algorithm aims to bind the runtime latency (i.e., $l_{i,\beta}(t)$) measured in each time window to oscillate within a desired range slightly lower than the latency demand (i.e., $ld_{i,\beta}$). The oscillation range is set between the high threshold ($th_{i,H}$) and low threshold ($th_{i,L}$), which are configurable with two parameters: a guard interval ($l_{i,GI}$) and an oscillation ratio ($\alpha_i$). Thus, $th_{i,H} = ld_{i,\beta} - l_{i,GI}$ and $th_{i,L} = (1 - \alpha_i) \times ld_{i,\beta}$.

The algorithm adjusts the permitted queueing latency (i.e., $l_{i,pq}$) based on the runtime latency (i.e., $l_{i,\beta}$) in three different cases as follows.

- $l_{i,\beta}(t) > th_{i,H}$: the latency demand is about to be violated; the adaptation undergoes a multiplicative

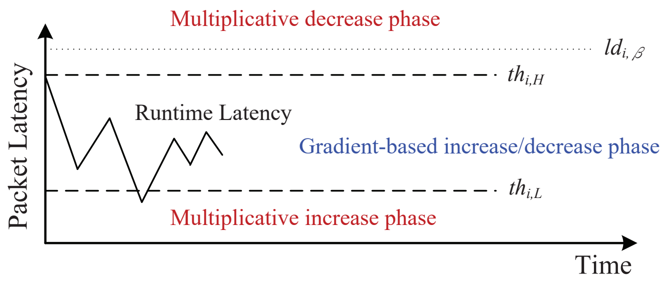

Multiplicative decrease phase
$ld_{i,\beta}$
$th_{i,H}$
Runtime Latency
Gradient-based increase/decrease phase
$th_{i,L}$
Multiplicative increase phase
Packet Latency
Time

Fig. 8: Window-based delay gradient approach.

decrease (MD) with a factor, denoted as $MD_i$, as $l_{i,pq}(t) = l_{i,pq}(t-1) * (1 - MD_i)$.

- $l_{i,\beta}(t) \leq th_{i,L}$: the obtained latency is too small; the adaptation undergoes a multiplicative increase (MI) with a factor, denoted as $MI_i$, as $l_{i,pq}(t) = l_{i,pq}(t-1) * (1 + MI_i)$.
- $th_{i,L} < l_{i,\beta}(t) \leq th_{i,H}$: the obtained latency remains in the safe oscillation range with the latency change $l_{i,ch}(t) = l_{i,\beta}(t) - l_{i,\beta}(t-1)$ and $l_{i,pq}(t)$ is calculated with gradients as follows:

$$l_{i,pq}(t) = \begin{cases} l_{i,pq}(t-1) \times (1 - \frac{MD_i \times l_{i,ch}}{th_{i,H} - l_{i,\beta}(t-1)}) & if\ l_{i,ch}(t) \geq 0 \\ l_{i,pq}(t-1) + \frac{AI_i \times \left|l_{i,ch}\right|}{l_{i,\beta}(t-1) - th_{i,L}}) & if\ l_{i,ch}(t) < 0 \end{cases},$$

where $AI_i$ is a factor of additive increase (AI).

Generally, $l_{i,pq}$ is adapted in the opposite direction of the runtime latency change. For the first two cases where $l_{i,\beta}$ exceeds the desired range, multiplicative factors are used to rapidly adapt $l_{i,pq}$. In the third case, to keep $l_{i,\beta}$ oscillating within the range, the adaptation is set to be proportional to the gradient of the runtime latency change. Moreover, a conservative approach with multiplicative decrease and additive increase on $l_{i,pq}$ is taken to prevent $l_{i,\beta}$ from going beyond the desired range.

**Latency-aware Scheduler.** The latency-aware scheduler notifies the flow-level scheduler of each priority flow potentially going to violate its latency demand to tag its flow queue as `urgent`. As depicted in Figure 7, it consists of three components: cross-layer throttler, hardware queue (HWQ) draining estimation, and aggregation adaptation. The cross-layer throttler determines any potential demand violations by checking whether any flow's expected queueing latency, denoted as $l_{i,e}$, has exceeded the permitted queueing latency ($l_{i,pq}$) marked by the delay controller. For each flow, when the violation occurs, its flow queue is tagged as `urgent` to be prioritized. The expected queueing latency is the sum of the software and hardware queueing delays, represented as $l_{sq}$ and $l_{hq}$, respectively: $l_{i,e} = l_{sq} + l_{hq}$.

The software queueing delay is calculated based on the time the earliest packet in the flow's queue has been queued, while the hardware queueing delay is estimated by the HWQ draining estimation module since it has not been experienced by the queued packets. The expected hardware queueing delay for a new frame depends on the time it takes for its preceding frames in the hardware queue to be drained. Given the number of the preceding frames, denoted as $N_{hq}$, the expected hardware queueing delay can be calculated as follows: $l_{hq} = T_{ctt} + \sum_i^{N_{hq}} (T_{ctt} + T_{mac} + T_{i,air}) + T_{guard}$, where $T_{ctt}$ is the average contention time, $T_{mac}$ is the MAC overhead, $T_{i,air}$ indicates the airtime needed for the transmission of frame $i$, and $T_{guard}$ is the guard period used to prevent the latency from being underestimated due to channel dynamics. Notably, the first $T_{ctt}$ represents the contention time of the new frame, whereas the summation calculates how much time the new frame needs to wait for its preceding frames in the hardware queue.

We estimate the average contention time using a moving average by considering two cases. First, when a frame is scheduled into an empty hardware queue, the contention time is calculated by subtracting the time the frame is enqueued to the hardware, the MAC overhead, and the frame's transmission airtime, from the time the frame's ACK is received. Second, when the hardware queue is not empty, the contention time is estimated by subtracting the frame's transmission airtime and the MAC overhead from the time interval between its ACK and the preceding ACK. The MAC overhead is calculated as the sum of the preamble, short interframe spaces (SIFS), and the frame's ACK transmission airtime. The airtime of a frame can be calculated based on the frame's size divided by the physical rate, for which the most recently used rate is chosen from the rate adaptation module.

However, in some cases where the estimated latency for a prioritized flow is only slightly smaller than the permitted latency, and a non-prioritized frame is then scheduled to the hardware queue, the prioritized flow may be scheduled late and suffer from long latency, especially when the scheduled non-prioritized frame has a large aggregation size and requires a long transmission time. To address this issue, we introduce aggregation adaptation, which reduces the allowable aggregation size for non-prioritized flows when any prioritized flows exist; otherwise, it reverts to the maximum value.

## V. Implementation

We implement LAST-PQ on a Linux-based PC with an Atheros Wi-Fi card using the open-source ath9k driver. Below, we describe four major implementation issues.

**Runtime Collection of Packet Latency Statistics.** The delay controller in the user space relies on eBPF and `kprobe` to collect per-packet latency results from the wireless MAC stack and driver in the kernel space. The BPF Compiler Collection (BCC) toolkit is employed to develop a Python program that uses eBPF and `kprobe` to insert hooks into some kernel functions corresponding to the latency results.

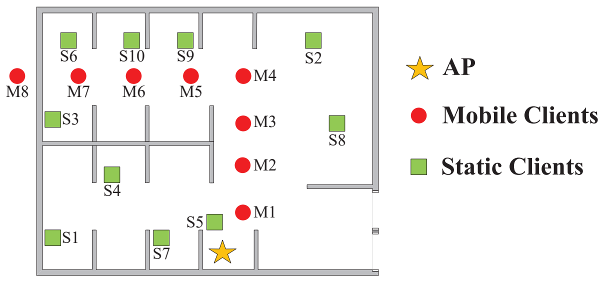
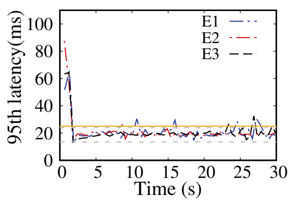
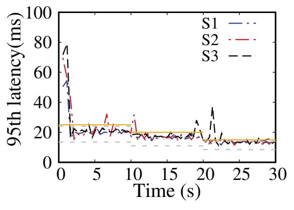


Fig. 9: The experimental floor plan.



(a) 25 ms latency.



(b) 25/20/15 ms latency.

Fig. 10: Evaluation of the LAST-PQ runtime control with different latency requirements.

**High-priority Mark with Permitted Queueing Latency.** We leverage an existing field of the socket buffer, `mark`, which is a 32-bit integer value, for the high-priority mark. The `mark` field can be configured by the delay controller at runtime using the `iptable` command as "`-j MARK --set-mark $priority_mark`", where the `priority_mark` is a specified 32-bit value serving as the high-priority mark. In this mark, the top 16 bits are chosen as a specific value different from other usages ("b45f" in our implementation), whereas the bottom 16 bits are used to carry the value of the permitted queueing latency.

**Priority Queues with Urgent Tags.** We add a prioritized flow queue and a prioritized TID queue to the data structures of the flow-level and device-level schedulers, respectively. These structures are located in the wireless MAC stack and `ath9k` driver, respectively. Additionally, the `urgent` tag is appended to the per-flow data structure.

**Parameters Determination.** We determine the values of the aforementioned parameters by conducting a mobility experiment in a multi-flow, multi-client case. This experiment involves varying channel conditions and congested traffic, requiring LAST-PQ to have timely scheduling adaptation. The obtained values from this challenging case should work for most scenarios. We vary MD, MI, and AI to examine which values of them can provide latency closest to 20 ms at the 95th percentile; they are 0.3, 0.3, and 5000, respectively. Notably, the values of the other parameters, i.e., $T_{guard}$, $l_{GI}$, and $\alpha$, are determined empirically; they are 1 ms, 1 ms, and 0.5, respectively.

## VI. Evaluation

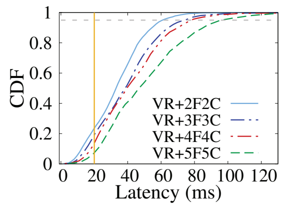
In this section, we evaluate the effectiveness of the runtime latency control for LAST-PQ and assess it in three cases, all with congested traffic: static clients, mobile clients, and static clients with the EDCA enhanced by LAST-PQ. We compare the performance of LAST-PQ with that of the updated scheduling method in the Linux kernel. Figure 9 shows the experimental floor plan, and the other experimental settings are similar to those in the case study (see Section III) unless explicitly specified. Notably, the focus of the latency demands is on the 95th percentile; however, LAST-PQ can also be applied to other percentiles.

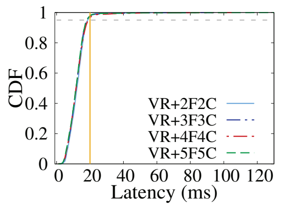
**Runtime Latency Control.** We examine whether LAST-PQ can meet various latency requirements at runtime, considering three locations with distinct channel conditions. As depicted in Figure 10, most latency results remain below the specified 25 ms latency requirement, with only a few exceptions. These exceptions are primarily attributed to sudden channel or traffic changes. However, the subsequent latency results are effectively managed to meet the requirement. To further assess LAST-PQ's adaptability, we modify the latency requirement of the VR flow sequentially to 25, 20, and 15 ms. LAST-PQ successfully controls the latency of the VR flow to align with the evolving latency requirements, and any occasional exceptions are promptly smoothed out.

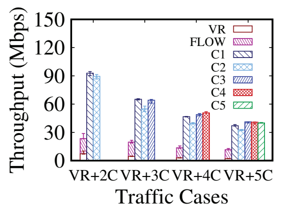
**Static Case.** We vary the number of coexistent non-VR clients while simultaneously adjusting the number of concurrent non-VR flows at the VR smartphone. The performance of a VR flow streaming to a smartphone at the farthest location, S6, is evaluated, and non-VR clients are deployed around S5. Figures 11a and 11b illustrate the 95th latency performance for the Linux and LAST-PQ scheduling methods, respectively. The Linux scheduling method can elevate the VR latency to as much as 99.39 ms, whereas LAST-PQ consistently maintains the VR latency below 20 ms, with a maximum of only 19.98 ms. LAST-PQ achieves a remarkable reduction in latency by 79.89%.

We also observe that the VR flow, exhibiting less aggressive traffic, can only achieve a throughput of 2.06-7.30 Mbps using the Linux scheduling, as demonstrated in Figure 11c. It is noteworthy that the VR smartphone is located farther away than the other clients, resulting in its aggregate throughput from both VR and non-VR flows being much lower than theirs. However, LAST-PQ can ensure the VR flow's throughput in such heavy congestion cases, maintaining a throughput ranging from 43.65 Mbps to 45.27 Mbps, as illustrated in Figure 11d.

**Mobility Case.** We then assess the real-time adaptability of LAST-PQ in mobility cases, where the wireless channel condition of the VR smartphone undergoes constant changes, resulting in variations in the VR flow's latency over time. The VR smartphone is moved from M1 to M8 at a speed of 1 m/s and then returns to M1, while concurrently having 5 non-VR flows at the VR smartphone and 5 non-VR clients. In contrast, the Linux scheduling method struggles to maintain the VR flow


(a) Linux scheduling.


(b) LAST-PQ scheduling.


(c) Linux scheduling.

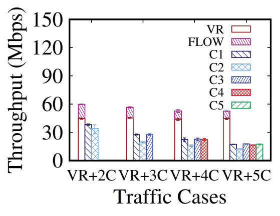
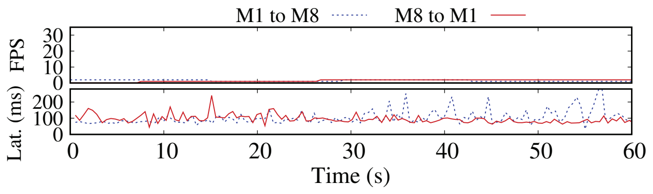

(d) LAST-PQ scheduling.

Fig. 11: Multi-client, multi-flow cases: the latency performance of a VR flow varies with number of coexistent non-VR flows (at the VR smartphone) and non-VR clients .


(a) Linux default scheduling.

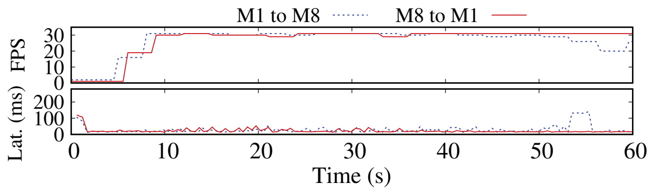

(b) LAST-PQ scheduling.

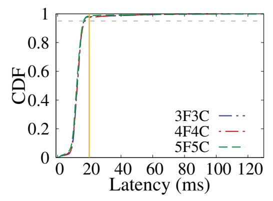
Fig. 12: Mobility case: the FPS and latency performance of a VR flow change over time while the VR smartphone is moving from M1 to M8 and from M8 to M1.

even in static congestion cases, performing even worse in scenarios involving mobility. As illustrated in Figure 12a, the average FPS values are only 1.41 and 1.50 for the M1-to-M8 and M8-to-M1 cases, respectively. The 95th latency requirement is consistently unmet, with 98.16% of values above 60 ms and the maximum value reaching 322.64 ms.

Figure 12b depicts the FPS and latency performance for LAST-PQ. It is observed that LAST-PQ can consistently meet the latency requirement below 20 ms, taking 1.70 s and 1.74 s from the start for the M1-to-M8 and M8-to-M1 cases, respectively. However, it takes 8.08 s and 12.12 s, respectively, to reach the full 30 FPS. Subsequently, LAST-PQ consistently satisfies the latency requirement with average FPS values, 26.86 and 27.56, respectively, except for the time period after 53.5 s in the M1-to-M8 case. Although this latency increase temporarily causes the FPS to drop to 20, it swiftly recovers within 9.1 s. The sudden latency rise is attributed to the deterioration of the VR smartphone's wireless channel while moving towards the farthest location, M8.

**Enhanced EDCA.** We further apply LAST-PQ to enhance the high-priority categories to be latency-aware. It can prioritize VR flows while satisfying different latency demands. In the following experimental results, we take the highest-priority access category, VI, as an example, following a setup similar to the EDCA case study in Section III-C.

In congested cases where all non-VR flows at the VR smartphone and non-VR clients' flows are configured to stay in the VI, LAST-PQ can still meet the latency requirement of a coexistent VR flow, as illustrated in Figure 13a. Moreover, by considering two VR flows and three 50 Mbps TCP flows with latency requirements of


(a) A VR flow with multiple non-VR flows/clients.

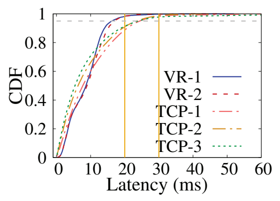

(b) Coexistent VR/non-VR flows with different demands.

Fig. 13: All traffic flows stay in the VI access category.

20 ms and 30 ms, respectively, LAST-PQ successfully meets the requirements of all these five flows, as shown in Figure 13b. The 95th latency values for the VR and TCP flows are 16.18-17.29 ms and 23.86-24.48 ms, respectively.

## VII. Related Work

Packet latency in Wi-Fi networks can accumulate from four major functions of the MAC layer and below: queue management [3], [5], [12]–[19], packet scheduling [4], [20]–[22], channel access [23]–[30], and transmission delay [31], [32]. In this section, we primarily focus on relevant studies within the queue management and packet scheduling functions, as these are closely related to the present study. Additionally, we explore current traffic scheduling solutions proposed for VR applications in the 5G network.

**Queue Management.** AQM [12], designed to address the bufferbloat issue [13], [14], is considered a best practice for queue management. It effectively reduces queue congestion and improves latency by dropping packets from congested queues. Numerous studies have introduced dropping policies, including RED [15], ARED [16], CoDel [5], and PIE [17]. More recently, for fairness in queueing, FQ-CoDel [3], [18] and FQ-PIE [19] have been proposed to enhance AQM with flow fairness. Notably, FQ-CoDel has become the default AQM algorithm in the mainline of Linux. However, it does not allow for the prioritization of VR flows.

**Packet Scheduling.** The packet scheduling operation at the AP can be decomposed into two parts: flow/client scheduling and frame aggregation scheduling. In a transmission opportunity, the former first schedules a client and its corresponding flows, and the latter then determines frame size and selects packets for frame aggregation. Various flow/client scheduling solutions [4], [20]–[22] have been proposed for Wi-Fi networks in recent years. Some aim to achieve airtime fairness among clients by considering transmission time [4], [20] and channel quality [21]. In an effort to enhance the Quality of Experience (QoE) for video streams, [22] proposes a scheduling algorithm that allocates resources to clients with video streams based on their congestion levels. However, this study is not practical without compliance to the current Wi-Fi scheduling architecture; it has not been implemented or evaluated on commodity devices.

As for frame aggregation scheduling, [33] dynamically adapts frame aggregation size based on the AC, taking QoS requirements into consideration. Several studies [31], [34], [35] have also been proposed to reduce retransmission overhead caused by losses in frame aggregation. While these studies can effectively reduce latency from the frame aggregation perspective, they may not guarantee latency performance for specific VR flows.

**Low-latency Scheduling for VR Flows.** There have been several low-latency scheduling solutions [36]–[39] introduced for VR flows in cellular networks. Two of them specifically focus on supporting VR applications with Ultra-Reliable and Low Latency Communications (URLLC) in 5G networks. [36] introduces a joint solution that can perform link adaptation and resource allocation simultaneously to achieve low latency for the URLLC. [37] develops a joint multi-user preemptive scheduling mechanism to cross-optimize spectral efficiency and packet latency. Additionally, [38] designs a multi-user MAC scheduling scheme for VR traffic in 5G networks, while [39] supports VR applications over LTE by presenting a client-side solution that leverages a cross-layer design and rich side channel information. These studies, primarily focusing on the cellular network, cannot be directly applied to the Wi-Fi network due to two major reasons. First, the cellular network has a different packet scheduling architecture from the Wi-Fi network, which contains the AC classifier and the two-level flow and device scheduling. Second, cellular network transmissions are based on centralized scheduling operations, while Wi-Fi network transmissions rely on distributed channel contention. In contrast, our proposed solution, LAST-PQ, is designed based on the Wi-Fi packet scheduling architecture and has been evaluated to be effective on practical off-the-shelf Wi-Fi platforms.

## VIII. Discussion

In this section, we discuss three major issues regarding LAST-PQ.

**Applicable to New Wi-Fi Standards.** While new Wi-Fi standards, such as IEEE 802.11ac/ax, have been introduced over the years to address contention problems to some extent, VR flows may still encounter challenges without a latency-aware scheduling solution like LAST-PQ. Despite the higher data rates provided by IEEE 802.11ac/ax compared to IEEE 802.11n, Wi-Fi devices, especially those with aggressive TCP flows and poor link quality, can congest a Wi-Fi AP and lead to severe contention. Additionally, although the 802.11ax/be standards incorporate OFDMA (Orthogonal Frequency Division Multiple Access) technology allowing multiple clients to communicate with the AP simultaneously, the proliferation of Wi-Fi devices, such as Wi-Fi IoT, in a Wi-Fi network can still result in severe contention. Hence, LAST-PQ remains essential to prioritize VR flows in new Wi-Fi networks, especially during instances of severe contention.

**Applied to OFDMA-based Wi-Fi Network.** The Wi-Fi network has supported the OFDMA technology since the IEEE 802.11ax standard, and our developed LAST-PQ can also be applied to it. It needs to cooperate with the OFDMA scheduling solution deployed at the Wi-Fi AP. During each transmission opportunity, LAST-PQ functions normally, emphasizing urgent flow queues, and the OFDMA scheduler considers them with high priority for participation in an OFDMA transmission. If there is only one prioritized TID with urgent flow queues, the OFDMA scheduler can perform a traditional single-user transmission for the prioritized client. However, if there are multiple prioritized clients, possibly with non-prioritized ones, the OFDMA scheduler can schedule them in an OFDMA transmission to transmit data concurrently.

**Limitations of LAST-PQ.** LAST-PQ can support multiple VR devices and multiple low-latency flows on a single device, based on the scheduling of prioritized TID and flow queues. However, the number of supported devices and flows is limited. This limitation arises because LAST-PQ prioritizes low-latency devices/flows by suppressing other flows at the same AP. Therefore, the supported capacity approximates the amount where the required low-latency demands can be met assuming no other intra-AP flows. However, the actual capacity might be lower than expected, as non-prioritized transmissions may cause head-of-line blocking against urgent transmissions. This occurs when non-prioritized transmissions are scheduled into the hardware queue at a time without any urgent requests, but they are not completed before an urgent request appears and cannot be interrupted according to the wireless NIC operation framework. It is important to note that the capacity can be affected by wireless channel conditions where the low-latency devices are located and the traffic demands from neighbor APs. As a result, the capacity can vary over time.

## IX. Conclusion

Enabling common Wi-Fi networks to support VR applications holds promise due to its low-cost and convenient merits. However, our experimental case study reveals that the current Wi-Fi scheduling framework poorly supports VR flows in congested cases. To address this, we have developed LAST-PQ, a latency-aware scheduling solution designed to ensure the latency performance of elephant VR flows. LAST-PQ is compliant with the current Wi-Fi scheduling framework, making it applicable to commodity Wi-Fi devices. Its effectiveness has been confirmed through evaluations based on a prototype. While the experiments in this work are conducted on conventional IEEE 802.11n platforms, mainly due to a part of the scheduling framework being implemented in non-released firmware for current Wi-Fi platforms (e.g., 802.11ac/ax), the proposed LAST-PQ can also be applied to them.

## Acknowledgment

We appreciate the insightful and constructive comments from the anonymous reviewers. This work is supported in part by the National Science Foundation (NSF) under Grants No. CNS-2246050, CNS-2246051, and CNS-2321416, and by the National Science and Technology Council (NSTC) under Grants No. 110-2221-E-A49-031-MY3, 112-2628-E-A49-016-MY3, 112-2218-E-A49-023, and 112-2634-F-A49-001-MBK. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors only and do not necessarily reflect those of the NSF and NSTC.

## References

[1] "Virtual & augmented reality: Understanding the race for the next computing platform," Report, Goldman Sachs, Jan. 2016.

[2] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications - Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements*, IEEE Std. 802.11e, 2005.

[3] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "The flow queue CoDel packet scheduler and active queue management algorithm," RFC 8290, 2018.

[4] T. Høiland-Jørgensen, M. Kazior, D. Täht, P. Hurtig, and A. Brunstrom, "Ending the anomaly: achieving low latency and airtime fairness in WiFi," in *Proc. USENIX Annual Technical Conference (ATC'17)*, Santa Clara, CA, Jul. 2017.

[5] K. Nichols, A. McGregor, V. Jacobson, and J. Iyengar, "Controlled delay active queue management," RFC 8289, 2018.

[6] M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance anomaly of 802.11b," in *Proc. IEEE International Conference on Computer Communications (INFOCOM'03)*, San Francisco, CA, USA, Mar. 2003.

[7] "OpenWrt," 2024. [Online]. Available: https://openwrt.org/releases/18.06/start

[8] "Steam VR," 2024. [Online]. Available: https://steamcommunity.com/steamvr

[9] "Trinus virtual reality," 2024. [Online]. Available: https://www.trinusvirtualreality.com/

[10] L. Liu, R. Zhong, W. Zhang, Y. Liu, J. Zhang, L. Zhang, and M. Gruteser, "Cutting the cord: designing a high-quality untethered vr system with low latency remote rendering," in *Proc. ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'18)*, NY, USA, Jun. 2018.

[11] "Cloud AR/VR whitepaper," White Paper, GSMA Future Networks, Apr. 2019.

[12] E. F. Baker and E. G. Fairhurst, "IETF recommendations regarding active queue management," RFC 7567, 2015.

[13] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet: Networks without effective aqm may again be vulnerable to congestion collapse," *ACM Queue*, vol. 9, no. 11, pp. 40–54, Nov. 2011.

[14] V. Cerf, V. Jacobson, N. Weaver, and J. Gettys, "Bufferbloat: What's wrong with the internet? a discussion with vint cerf, van jacobson, nick weaver, and jim gettys," *ACM Queue*, vol. 9, no. 12, pp. 10–20, Dec. 2011.

[15] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[16] S. Floyd, R. Gummadi, and S. Shenker, "Adaptive RED: An algorithm for increasing the robustness of RED's active queue management," AT&T Center for Internet Research at ICSI, Tech. Rep., Aug. 2001.

[17] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg, "PIE: A lightweight control scheme to address the bufferbloat problem," in *IEEE International Conference on High Performance Switching and Routing (HPSR'13)*, Taipei, Taiwan, Jul. 2013.

[18] T. Høiland-Jørgensen, P. Hurtig, and A. Brunstrom, "The good, the bad and the wifi: Modern AQMs in a residential setting," *Computer Networks*, vol. 89, pp. 90–106, Oct. 2015.

[19] G. Ramakrishnan, M. Bhasi, V. Saicharan, L. Monis, S. D. Patil, and M. P. Tahiliani, "FQ-PIE queue discipline in the linux kernel: Design, implementation and challenges," in *Proc. IEEE LCN Symposium on Emerging Topics in Networking (LCN Symposium'19)*, Osnabrück, Germany, Oct. 2019.

[20] R. Garroppo, S. Giordano, S. Lucetti, and L. Tavanti, "Providing air-time usage fairness in IEEE 802.11 networks with the deficit transmission time (dtt) scheduler," *Wireless Networks*, vol. 13, pp. 481–495, Aug. 2007.

[21] K. Gomez, R. Riggio, T. Rasheed, and I. Chlamtac, "On efficient airtime-based fair link scheduling in IEEE 802.11-based wireless networks," in *Proc. IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIRMC'11)*, Toronto, ON, Canada, Sep. 2011.

[22] S. Nosheen and J. Y. Khan, "High definition video packet scheduling algorithms for IEEE 802.11ac networks to enhance QoE," in *Proc. IEEE Vehicular Technology Conference (VTC'20)*, Antwerp, Belgium, May 2020.

[23] P. Serrano, A. Banchs, P. Patras, and A. Azcorra, "Optimal configuration of 802.11e EDCA for real-time and data traffic," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 5, pp. 2511–2528, Feb. 2010.

[24] C. Cano, B. Bellalta, and M. Oliver, "Adaptive admission control mechanism for IEEE 802.11e WLANs," in *Proc. IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'07)*, Athens, Greece, Sep. 2007.

[25] G. Cena, S. Scanzio, L. Seno, and A. Valenzano, "A fixed-priority access scheme for industrial Wi-Fi networks," in *Proc. IEEE Industrial Electronics Society (IECON'16)*, Florence, Italy, Oct. 2016.

[26] I. Syed, S. Shin, B. Roh, and M. Adnan, "Performance improvement of QoS-enabled WLANs using adaptive contention window backoff algorithm," *IEEE Systems Journal*, vol. 12, no. 4, pp. 3260–3270, May 2018.

[27] Y. Xiao, F. H. Li, and S. Choi, "Two-level protection and guarantee for multimedia traffic in IEEE 802.11e distributed WLANs," *Wireless Networks*, vol. 15, no. 2, pp. 141–161, Feb. 2009.

[28] Hyun-Jin Lee and Jae-Hyun Kim, "A optimal CF-Poll piggyback scheme in IEEE 802.11e HCCA," in *Proc. International Conference on Advanced Communication Technology (ICACT'06)*, Phoenix Park, South Korea, Feb. 2006.

[29] A. Grilo, M. Macedo, and M. Nunes, “A scheduling algorithm for QoS support in IEEE 802.11 networks,” *IEEE Wireless Communications*, vol. 10, no. 3, pp. 36–43, Jun. 2003.

[30] G. Cecchetti, A. L. Ruscelli, A. Mastropaolo, and G. Lipari, “Dynamic TXOP HCCA reclaiming scheduler with transmission time estimation for IEEE 802.11e real-time networks,” in *Proc. ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM’12)*, New York, USA, Oct. 2012.

[31] C. Li, C. Peng, S. Lu, X. Wang, and R. Chandra, “Latency-aware rate adaptation in 802.11n home networks,” in *Proc. IEEE Conference on Computer Communications (INFOCOM’15)*, Kowloon, Hong Kong, Apr. 2015.

[32] A. Chan, H. Lundgren, and T. Salonidis, “Video-aware rate adaptation for MIMO WLANs,” in *Proc. IEEE International Conference on Network Protocols (ICNP’11)*, Vancouver, BC, Canada, Oct. 2011.

[33] N. Hajlaoui, I. Jabri, M. Taieb, and M. Benjemaa, “A frame aggregation scheduler for QoS-sensitive applications in IEEE 802.11n WLANs,” in *Proc. International Conference on Communications and Information Technology (ICCIT’12)*, Hammamet, Tunisia, Jun. 2012.

[34] B. Maqhat, M. D. Baba, and R. A. Rahman, “A-MSDU real time traffic scheduler for IEEE 802.11n WLANs,” in *Proc. IEEE Symposium on Wireless Technology and Applications (ISWTA’12)*, Bandung, Indonesia, Sep. 2012.

[35] A. Saif and M. Othman, “SRA-MSDU: enhanced A-MSDU frame aggregation with selective retransmission in 802.11n wireless networks,” *Journal of Network and Computer Applications*, vol. 36, no. 4, pp. 1219–1229, Jul. 2013.

[36] G. Pocovi, K. I. Pedersen, and P. Mogensen, “Joint link adaptation and scheduling for 5G ultra-reliable low-latency communications,” *IEEE Access*, vol. 6, pp. 28 912–28 922, May 2018.

[37] A. A. Esswie and K. I. Pedersen, “Multi-user preemptive scheduling for critical low latency communications in 5g networks,” in *Proc. IEEE Symposium on Computers and Communications (ISCC’18)*, Natal, Brazil, Jun. 2018.

[38] M. Huang and X. Zhang, “MAC scheduling for multiuser wireless virtual reality in 5G MIMO-OFDM systems,” in *Proc. IEEE International Conference on Communications Workshops (ICC Workshops’18)*, Kansas City, MO, USA, May 2018.

[39] Z. Tan, Y. Li, Q. Li, Z. Zhang, Z. Li, and S. Lu, “Supporting mobile vr in lte networks: How close are we?” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 1, pp. 1–31, Apr. 2018.