DCTPQ: Dynamic Cloud Gaming Traffic Prioritization Using Machine Learning and Multi-Queueing for QoE Enhancement

Alireza Shirmarz[1], Carlos Henrique de França Marques[1], Fábio Luciano Verdi[1], Roberto Silva Netto[2], Suneet Kumar Singh[3], Christian Esteve Rothenberg[4],

[1]Dep. of Computer Science, Federal University of São Carlos (UFSCar), Sorocaba, Brazil

[2]Dep. of Computer Science, FACENS, Sorocaba, Brazil

[3]Dep. of Computer Science, Norwegian University of Science and Technology, Norway

[4]Dep. of Computer Engineering, Universidade Estadual de Campinas (UNICAMP), Campinas, Brazil

`{ashirmarz,carlos.franca,verdi}@ufscar.br, roberto.netto@facens.br`

`Suneet.k.singh@ntnu.no, chesteve@dca.fee.unicamp.br`

***Abstract.** Cloud gaming (CG) traffic requires high bandwidth and low latency to ensure Quality of Experience (QoE). We propose DCTPQ, an ML-based edge solution that dynamically identifies and prioritizes CG traffic on-the-fly, achieving 97.6% classification accuracy using packet-based and RTP frame-based features with Decision Tree (DT) and Random Forest (RF) models. DCTPQ employs separate queues for CG, UDP (Non-CG), and TCP traffic, with varied lengths and rates, implemented using P4 on the data plane. Leveraging Inband Network Telemetry (INT) and Device-in-the-Loop (DIL) techniques, we evaluate QoS (throughput, latency, packet sojourn time) and QoE (VMAF score) under congestion. The system is tested with three distinct CG games (Fortnite, Forza, Mortal Kombat) on the Xbox platform, while users play online, ensuring a realistic assessment of the deployed model's impact on QoS and QoE.*

1. Introduction

In recent years, the video gaming industry has experienced substantial growth, evolving from a niche hobby into one of the most significant sectors within the entertainment industry. Consequently, CG has emerged as a focal point of both industry and academic research [Kougioumtzidis et al. 2024]. CG requires high bandwidth for video streaming from servers to clients and low latency to ensure interactivity [Graff et al. 2024]. The downlink typically involves video streaming from the server to the client, while the uplink consists of the user's real-time commands. In CG, response time is the total time from when a player sends a command to when they receive a frame from the server, while latency refers to the one-way delay, the time it takes for a packet to travel from the source to the destination (and vice-versa). Short latency and quick response times are essential to maintain a seamless gaming experience [Shirmarz et al. 2024, Graff et al. 2023].

To effectively manage and minimize latency, application-level solutions dynamically configure client-side video frame buffers to ensure smooth video frame display (fps), adapting seamlessly to fluctuations in network latency. Despite substantial advancements in CG frameworks such as Google Stadia (STD), Nvidia GeForce Now (GFN), Microsoft Xbox Cloud Gaming (XC), Sony Playstation Now (PSN), and Amazon Luna, the implementation of robust queue buffering mechanisms remains critical for mitigating packet bursts and ensuring low-latency performance. This is particularly crucial due to the inherent variability in wireless network resources and capacity, especially in 4G, 5G, and Wi-Fi-based Access Points (APs), which serve as the final connection point for gamers access-

ing CG over the Internet. Moreover, the phenomenon of bufferbloat may arise within network infrastructure queues [Gettys and Nichols 2011], leading to increased latency, a key QoS metric, for CG traffic. QoS is measured through metrics such as latency, jitter, and throughput, while QoE represents the end-user perception, influenced by received frame quality, audio fidelity, and response time [Carvalho et al. 2023]. This study focuses on CG applications, emphasizing the differentiation of CG traffic at the edge (last hop) to enhance QoS metrics and, consequently, QoE. To evaluate user experience, we analyze the quality of received frames in the CG downlink using the VMAF score [Nádas et al. 2024], examining its variation under different queue settings implemented with DCTPQ.

We tackle the challenge of CG performance by ensuring high throughput and low latency through real-time traffic classification, multi-queue resource allocation, and edge-based DIL evaluation. Our high-accuracy model, trained on a pre-collected dataset and tested in real-time cloud gaming, classifies CG vs. non-CG traffic using DT and RF with both packet-level and RTP-based features—marking the first use of RTP features in CG classification. To optimize fluctuating bandwidth, we introduce a multi-queue architecture with dedicated CG, UDP (Non-CG), and TCP queues, adjusting queue rate and length to prioritize CG traffic while ensuring fair resource allocation. Implemented on P4Pi as an AP, DCTPQ integrates DIL methodology for on-the-fly classification and resource allocation. We evaluate QoS and QoE in real-time Xbox cloud gaming, uniquely analyzing Fortnite, Forza, and Mortal Kombat to understand game-specific network impacts. This is the first work to combine classification, queue allocation, and real-time QoS/QoE evaluation, offering a novel framework to enhance cloud gaming at the network edge. To sum up, our contributions are as follows:

- We achieved 97.6% accuracy in identifying CG flows from other (Non-CG) UDP-based application flows using RF and DT models using RTP features (IFI and FS) in addition to packet-based features (IPI, PS);
- We propose a multi-queue architecture with dynamic queue allocation, prioritizing CG traffic while ensuring smoothness for misclassified flows, ultimately enhancing both QoS and QoE.
- We propose a multi-queue architecture with dynamic queue allocation, prioritizing CG traffic while ensuring smoothness for misclassified flows, and evaluate QoS and QoE for each game separately to provide distinct performance insights;
- DCTPQ was developed on P4Pi as a proof of concept. The datasets, ML model (in pickle), P4 code for the data plane, and Python code for the controller are publicly available[1].

The paper is organized as follows: Section II reviews related work on CG traffic classification and queuing disciplines. Section III introduces DCTPQ. Section IV describes the DIL experiment setup. Section V presents QoS and QoE assessment results. Section VI discusses the advantages and disadvantages of the approach. Finally, Section VII concludes the study and outlines future work.

2. Related Works

Bufferbloat [Gettys and Nichols 2011]—caused by large network buffers filling up—severely impacts QoS for CG. Downlink traffic (video frames) and uplink traffic (user commands) both affect QoE, as larger packets on the downlink can exacerbate queue buildup [Jarschel et al. 2011]. To tackle bufferbloat, AQM schemes (RED, CoDel, PIE) drop or mark packets with ECN [Floyd and Jacobson 1993, Pan et al. 2013, Jacobson and Kathleen 2012, Ramakrishnan et al. 2001, de Almeida et al. 2022]. When streaming over UDP, the network must detect congestion and manage queues to avoid long delays and loss [Nokia 2024, Bell Labs]. L4S [De Schepper and Briscoe 2023] reduces latency and packet loss by marking congestion early, but network-side mechanisms can further

---

[1]https://github.com/dcomp-leris/DCTPQ

improve CG packet sojourn times [Marchal et al. 2023]. Multi-queue setups and tailored queue management (e.g., HTB, Dual Queue Coupled AQM) help separate CG from other traffic [Graff et al. 2024, Bondarenko et al. 2016, Albisser et al. 2019, Rhee et al. 2018, Alizadeh et al. 2010]. Studies with TCP Cubic, TCP BBR, and CG flows [Xu and Claypool 2022] show differing competition behaviors; loss-based CCAs can cause excessive buffering, while BBR constrains queue growth.

Our approach, DCTPQ, addresses both CG traffic classification and dedicated queue allocation with improved queue rate and queue length at the edge, which connects users to the Internet for online gaming. Unlike previous works that consider end-to-end effects, we focus specifically on evaluating the edge device role in QoS and QoE, independent of Internet-side factors. DCTPQ, deployed on P4Pi, classifies CG traffic with 97.6% accuracy using *packet-based* and *RTP frame-based features*, considering uplink/downlink behavior. It assigns a dedicated queue for CG while managing UDP (non-CG) and TCP traffic separately, ensuring improvement even when classification errors (false positives) occur (2.4%). To mitigate misclassification impact, UDP traffic is allocated a queue with a rate and length close to CG, covering potential misclassified flows. Using a device-in-the-middle setup, we evaluate QoE (VMAF) and QoS metrics in real-time while gamers play online. Our findings highlight the impact of AP-based classification and CG queue allocation on QoS and QoE, demonstrating the effectiveness of DCTPQ beyond simulation-based studies.

3. Proposed DCTPQ for Cloud Gaming

In this paper, we propose DCTPQ which is a dynamic queue allocation system for CG traffic, deployed on a programmable device (P4Pi) where gamers connect to the CG server through the Internet. The APs face challenges with encryption and dynamic ports to use deterministic identifiers like IP addresses and port numbers for CG traffic classification [Shirmarz et al. 2024, Graff et al. 2023], so DCTPQ uses ML to identify the CG traffic on the fly and allocate it to dedicated queue. DCTPQ comprises two components: a *Traffic Classifier (TC)* in the controller, distinguishing CG from other UDP flows, and a *Queue Allocation Engine (QAE)* in the data plane on P4Pi (running BMv2), which assigns traffic to three specialized queues according to the TC's decisions.

We configure queue length and rate according to $Sojourn\ Time = \frac{Q_{length}}{Q_{Rate}}$ to deliver differentiated service levels. Consequently, we slice traffic into three distinct queues: (1) CG—gaming traffic with strict delay requirements, (2) Non-CG UDP—delay-sensitive but loss-tolerant and lacking congestion control, and (3) TCP—delay-tolerant, loss-sensitive, typically using Cubic or BBR. Separating Non-CG UDP from TCP avoids QoS and QoE degradation, since TCP's congestion control can aggressively dominate buffer resources.

We leverage Little's Law and PASTA principles [Allen 2014] to estimate sojourn times and ensure switch stability while prioritizing CG traffic. Queue lengths are set for Q_0, Q_1, and Q_2, with fixed processing rates, to evaluate their effects on QoS metrics for TCP, non-CG UDP, and CG traffic under congestion. Four queue sizes (64, 128, 256, 512 packets) are tested, balancing burst traffic support with queuing delay trade-offs. TCP and UDP queues are separated due to differing QoS needs, enabling effective resource allocation and improved user satisfaction. Our ML model achieves 97.6% accuracy, with CG traffic tolerating brief misclassification (up to 2.4%). By prioritizing CG traffic within the UDP queue, we ensure higher QoS and QoE for interactive applications while maintaining fairness and efficiency for all users. DCTPQ meets TCP and UDP requirements, enhancing CG player experience during congestion.

3.1. CG Traffic Classification

In DCTPQ, TCP and UDP traffic are separated based on protocol numbers in the data plane. UDP traffic is further classified in the control plane into CG or Non-CG using a ML model, as illustrated in
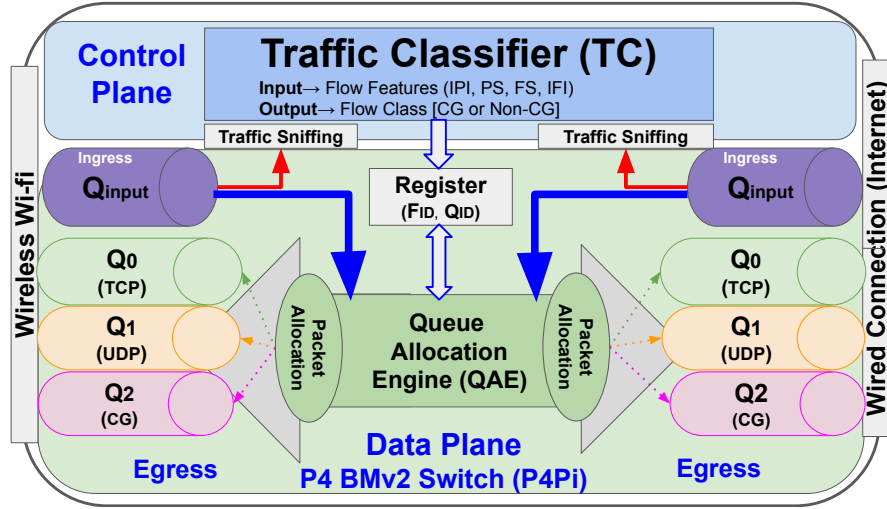
Figure 1. DCTPQ architecure deployed on P4Pi. Three level queues: Queue (0) for default best-effort traffic (TCP), Queue (1) for Non-CG UDP-based traffic, and Queue (2) for CG traffic with higher priority. It is equipped with a Wi-Fi 802.11ac (Downlink) interface and an Ethernet (Uplink) interface with a speed of 1 Gbps.

Fig.1. This ML model offers flexibility and ease of development, accommodating diverse CG frameworks and configurations, as well as various Non-CG applications that exhibit different network features. The process steps to train and deploy the model includes: (1) collecting datasets for training and evaluation, (2) selecting relevant and effective features, (3) performing pre-processing, (4) choosing a training algorithm, (5) evaluating the model, and (6) deploying the model.

Dataset: We use the dataset collected in [Graff et al. 2023] related to different CG platforms and the Xbox CG datasets from [Shirmarz et al. 2024]. We allocate 10% for evaluation and 90% for training. To enhance the model's ability to identify the 'Non-CG' class, we include UDP-based application traffic (such as Streaming, VoIP, and P2P) collected in [Gil et al. 2016], along with Non-CG datasets from [Graff et al. 2023]. The 'Non-CG' traffic includes applications such as Remote Desktop, Video Conferencing, Video Streaming, Live Video, Facebook, Navigation, VoIP, and P2P applications.

Feature Selection: According to [Graff et al. 2023], STD, GFN, and XC platforms use RTP over UDP to transfer video and audio, making it suitable for downlink CG flow, which involves video streaming to the client. Therefore, we consider the average PS, IPI, FS, and IFI as features, which can be extracted from network traffic at the interface rate and used for training the model.

- PS (Packet Size): The average length of the packets belonging to the same flow;
- IPI (Inter-Packet Interval): The average time between two consecutive packets;
- FS (Frame Size): The average RTP frame size. An RTP frame begins with an RTP marker flag set in the RTP header. The frame size is the sum of the lengths of UDP packets between two consecutive packets with the RTP marker flag set to 1;
- IFI (Inter-Frame Interval): The average time between two consecutive UDP packets with the RTP marker flag set to 1. It is calculated by subtracting the timestamps of these consecutive packets.

Pre-processing: We used Python and the Scapy package to parse PCAP files collected in the dataset for the training phase. In the PCAP files, we utilized packet data such as source and destination IPs, source and destination ports, and transport protocols to identify network flows. Timestamps and packet lengths were used to extract key features, namely the averages of PS, IPI, FS, and IFI, as detailed

in [Shirmarz et al. 2024]. For model training, we organized, labeled, merged, and shuffled the samples across datasets methodically. To address the class imbalance between 'CG' and 'Non-CG' samples, we adopted a weighted class strategy to maintain dataset balance and ensure fairness among two categories ('CG' and 'Non-CG').

Training the ML model: We train DT and RF models in the controller to classify traffic into 'CG' and 'Non-CG' based on PS, IPI, FS, and IFI features.

Evaluation of the model: To evaluate the classification model, we tested different feature combinations for DT and RF models. As expected, the RF model was more accurate, and we chose it for our deployment. The highest accuracy of 97.6% was achieved with the features PS, IPI, and FS, while adding IFI did not improve the results. This highlights the effectiveness of PS, IPI, and FS in classifying traffic into 'CG', and 'Non-CG' as shown in Table 2.

Deployment: After training and evaluation, the model is deployed in the controller using the TC component, which is implemented in the P4Pi controller for our proof of concept. The TC component performs four key functions: sniffing interfaces, computing features at line rate, classifying network flows, and setting registers in the data plane to forward traffic.

- TC sniffs incoming and outgoing packets from the Wi-Fi and Ethernet interfaces to collect data without hindering the data plane's forwarding process, thus avoiding latency.
- TC computes features using RTP frames identified by the marker bit in the RTP header. Feature computation begins after receiving at least two RTP frames, with the threshold set to three in this work. On average, an RTP frame is carried by 5.5 UDP datagrams whose extraction taking 1.28 ms to 6.48 ms depending on the rate.
- Flow classification is triggered upon receiving the features and performed using the trained RF model with features PS, IPI, and FS. To avoid hindering traffic forwarding, the TC classifies ingress traffic concurrently and independently of the data plane forwarding. To mitigate negative impacts of misclassification on long flows, a reset time of 5 seconds is defined to reclassify flows after a threshold.
- Once a flow is classified, the hash of the flow ID (destination IP and port) and queue ID (Q_0: TCP, Q_1: UDP, Q_2: CG) are stored in a data plane register to ensure subsequent packets are forwarded without delay.

The flow class is stored using a 16-bit $flow_{ID}$, with each register variable using 2 bits to define the Q_{id}. The register size is $2^{16} \times 2$ bits. This register in the data plane allows packets to be forwarded without waiting for classification. Periodic classification updates in the control plane ensure accurate flow handling without degrading QoS.

3.2. Traffic Forwarding & Dynamic Queue Allocation

The forwarding multi-queue architecture, developed on P4Pi using P4 V1 model defines three egress queues: Q_0 for TCP traffic, Q_1 for (Non-CG) UDP-based traffic, and Q_2 for CG. We focus on improving egress queues, leveraging the P4 pipeline's rapid processing capabilities. As far as we know, most CG platforms use UDP, so we consider CG traffic as UDP and extract it from other UDP traffic. Consequently, CG platforms using TCP fall outside the scope of this research. Incoming packets are directed to Q_0 if TCP, or Q_1/Q_2 if UDP, based on application type. For UDP, Non-CG traffic goes to Q_1 and CG traffic to Q_2. The $flow_{ID}$, derived from the destination IP and port, is hashed with CRC_{16}, and packets are forwarded based on the Q number (0 or 1 or 2) stored in the register by the TC component. Queues are prioritized using sojourn time, considering queue length and processing rate. This component, developed in P4, can be customized and easily deployed to other P4 targets such as Tofino Native Architecture (TNA) and Portable Switch Architecture (PSA).

The Raspberry Pi 4 setup includes a Wi-Fi IEEE 802.11 interface and an Ethernet interface. The wired connection links to the Internet, while game players and traffic generators use the wireless interface. This setup is ideal for CG players needing mobility. Packets are managed and prioritized in the QAE component in the developed in P4 according to Algorithm 1. This enhances CG user experience and robustness by providing reliable dynamic queue allocation, ensuring user satisfaction with distinct queues for UDP and TCP, and a high-priority queue for CG. This traffic allocation ensures CG flows receive high-priority service with the proposed ML model's accuracy. Even with inevitable misclassifications, CG service remains acceptable until reclassification. This approach makes ML models deployable in AP, providing robust service despite occasional true negatives which make the edge network smarter to mitigate the Wi-fi traffic fluctuation causing higher latency.

Algorithm 1 Queue Allocation Engine Algorithm (QAE)

Require: Network packet
Ensure: Forwarding packet to appropriate queue (Q_0, Q_1, or Q_2)
 1: **Receive Packet**
 2: **Extract Transport Protocol:** Determine if TCP or UDP
 3: **if** protocol == TCP **then**
 4:     Forward(packet, Q_0)
 5: **else if** protocol == UDP **then**
 6:     **Calculate Flow_ID:**
 7:     Compute `Flow_ID=CRC((`IP_{dst}`),(`$Port_{dst}$`))`
 8:     **Search `Flow_ID` in register for flow class:**
 9:     Get flow class from register
10:     (stored by TC component in controller)
11:     **if** flow_class for current `Flow_ID` exists **then**
12:         **if** flow_class == 'Non-CG' **then**
13:             Forward(packet, Q_1)
14:         **else if** flow_class == 'CG' **then**
15:             Forward(packet, Q_2)
16:         **end if**
17:     **else**
18:         Forward(packet, Q_1)
19:     **end if**
20: **end if**

As outlined in Algorithm 1, QAE starts by identifying the transport protocol from the IP header (protocol number 6 for TCP, 17 for UDP). TCP packets are forwarded to Q_0. For UDP packets, the algorithm calculates $Flow_{ID}$ using the CRC_{16} of the destination IP address and port number, referring to the defined register position to forward to the appropriate queue classified and registered by the TC component. If the classification changes during flow forwarding, subsequent packets are forwarded based on the new classification. Packets are forwarded to Q_1 (default UDP queue) until classified by the TC. Once classified, packets are forwarded based on their classification: Q_2 for CG and Q_1 for Non-CG. This QAE mechanism ensures that CG traffic benefits from specialized and prioritized queue QoS, avoiding the TCP queue. CG traffic experiences UDP-level QoE even during short misclassification periods. To ensure stable queues and prevent overloading, packet loss, and infinite sojourn times in the DCTPQ multi-queue model, we use Little's and PASTA laws. This ensures each queue remains stable despite the stochastic behavior of ingress traffic and processing rates.

3.3. Queue Modeling (M/M/1)

To predict sojourn time based on queue length, we model the queue to handle workload in congested network conditions using queuing theory. Since QAE forwards flows based on traffic type, we can model each queue (Q_0, Q_1, Q_2) as M/M/1. According to Little's Law, the queue system is modeled. λ is the ingress rate of the packets, and μ is the processing rate. W represents the waiting time a packet spends in the queue, and S is the processing time. The total time, $W + S$, represents the sojourn time a packet spends in the system.

The average number of packets in a stable system is estimated using $E[N] = \lambda \times E[Sojourn\ Time]$. E[N] represents the average number of packets, $E[Sojourn\ Time]$ represents the average sojourn time in seconds, λ is the arrival rate in PPS, and μ is the queue servicing rate in PPS. The average number of packets in the switch includes both those waiting in the queue E[N_q] and those being serviced E[N_s]. Utilization U is defined based on μ and λ as $U = \frac{\lambda}{\mu}$. According to Little's Law, if $U > 1\ or\ U = 1$, the queue is overloaded, causing more packet loss and higher sojourn times and E[Sojourn Time], E[N], and E[N_q] approach infinity. Therefore, a stable queue has $U < 1$. The average service time is E[S] = $\frac{1}{\mu}$, and the average Inter-Packet Interval (IPI) is E[A] = $\frac{1}{\lambda}$. Hence, $E[N]$ can be defined based on the utilization as $E[N_q] = \frac{U^2}{1-U}$. Therefore, the expected average sojourn time is calculated with $E[Sojourn\ Time] = \frac{U}{(1-U)\cdot\mu}$. Finally, by simplifying and calculating the expected sojourn time in each queue, we obtain $E[Sojourn\ Time] = \frac{1}{\mu-\lambda} = \frac{E(s)}{1-U}$.

Queue Stability & Constraints. In P4Pi, we can define a multi-queue architecture, setting the queue length in packets and the maximum processing rate in packets per second. In this work, we consider queue length (E[N_q]) and estimate utilization. Therefore, for E[N_q] = 64, 128, 256, and 512 packets, the utilizations are U_{64}=0.9848, U_{128}=0.9923, U_{256}=0.9961, and U_{512}=0.9981. These utilizations for different queue lengths represent the proportion of λ to μ in the system. While the ingress packet rate (λ) is stochastic, we can estimate it based on the Wi-Fi IEEE 802.11 rate for CG downlink, theoretically 54 Mbps. To keep the queue stable in congested conditions, we estimate the maximum μ in P4Pi, considering packet sizes between 64 Bytes and 1500 Bytes. This yields a μ_{max} range of 45,00 to 105,468 packets per second. Our findings show downlink average packet sizes for Fortnite, Forza, and Mortal Kombat are 875 Bytes, 890 Bytes, and 1050 Bytes, respectively, while uplink average packet sizes for all three games range from 150 to 400 Bytes. Considering these principles and P4Pi constraints, we define the queue length and maximum processing rate to maintain stable queues for various lengths and rates.

4. Experiment Methodology & Requirements

To evaluate DCTPQ, we use a DIL method under congested conditions, assessing QoS for CG, UDP, and TCP traffic, as well as QoE for CG. This section is organized into two parts: the first details the topology, devices, and software used for traffic generation and data collection, while the second introduces six DCTPQ configurations (S_1-S_6).

4.1. Topology & Hardware/Software Requirements

The deployed DCTPQ in P4Pi is used as a device in the loop within a network topology, as shown in Fig. 2. This topology includes three clients and a server to generate the required traffic and collect data for QoS and QoE evaluation. We use four machines in the testbed. Client (1) (Windows 10, Intel i5-4210U, 4 GB RAM) runs Google Chrome, Python3, Tshark, and OBS for game playback. Client (2) (Ubuntu 24.04, Intel i7-13700T, 16 GB RAM) and Client (3) (Ubuntu 24.04, Intel i7-5500U, 12 GB RAM) both run Iperf3 for UDP/TCP (Cubic or BBR). Finally, the Iperf Server (Ubuntu 18.04, Intel Xeon D-1518, 64 GB RAM) handles incoming traffic.
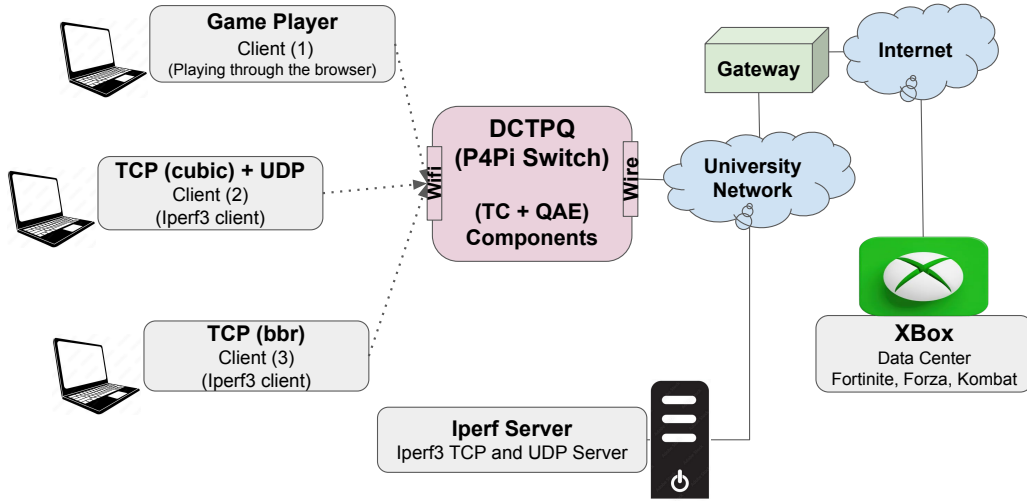
Figure 2. Topology for DCTPQ evaluation.

The P4Pi's Ethernet interface connects to the university network, which links to the lab server and the Internet via a gateway. The P4Pi's wireless interface connects to three clients, which generate different types of traffic and collect network data for evaluation. To generate CG traffic, we use the Xbox Cloud gaming service, selecting three games—Fortnite, Forza Horizon, and Mortal Kombat—to represent various genres. Client (1) logs network traffic in PCAP format via Tshark while a human uses Google Chrome to play on the Xbox online service. It also uses two Python modules to collect INT data, such as queue occupancy and sojourn time of queues (Q_0, Q_1, and Q_2), at a rate of one packet per second: (1) Sender, which sends INT packets, and (2) Receiver, which sniffs and logs these packets. The P4 program in the BMv2 switch processes and fills INT variables, sending packets back to the host with queue performance metrics. To evaluate video quality, we use OBS (Open Broadcaster Software) to capture the player's screen at 1366 x 768 resolution and 30 frames per second, mirroring our Xbox Cloud gaming settings. This captured video is compared with non-congested video frames using FFmpeg and VMAF [Netflix] tool to assess quality degradation due to congestion in different DCTPQ setups. The latency between the client and the Xbox server is challenging to evaluate due to UDP's nature and blocked ICMP packets in the path to the Xbox cloud. To compare fairly with other traffic (UDP and TCP using Iperf3) within the DCTPQ device, we measure the latency as RTT/2 between the client (1) and the Internet gateway (inside the University network) as shown in Fig. 2. The P4 code is developed to direct specific ICMP packets from Client (1) to Q_2 to measure latency, ensuring a fair comparison with UDP and TCP traffic, as end-to-end latency involves many other devices through the internet to the Xbox server.

To generate UDP and TCP traffic, we use Iperf3 with client (2) and client (3) acting as clients and the server in our lab as the Iperf3 server. For UDP traffic, representing delay-sensitive applications requiring constant throughput, client (2) sends traffic at a constant bitrate of 10 Mbps with a default packet size of 1470 bytes. For TCP traffic, which is delay-tolerant and loss-sensitive, we utilize two common CCAs: Cubic and BBR. Client (2) generates TCP traffic using the Cubic CCA, while client (3) uses the BBR CCA. The greedy behavior of these CCAs causes TCP to aggressively occupy the queue compared to other types of traffic.

To evaluate the DCTPQ performance, we set different queue lengths for each of the three queues (Q_0, Q_1, Q_2) while maintaining a fixed maximum processing rate to assess the effect of queue

length on QoS and QoE under congested conditions. Six scenarios (S_1 - S_6) with varying queue length combinations are tested. Each experiment runs for 300 seconds with three different games (Fortnite, Forza, and Mortal Kombat), resulting in 18 runs (six per game).

4.2. Queue Setup & Scenarios

We assume a constant processing rate (R) of 20,000 pkt/s per queue (30.72 Mbps - 720 Mbps total), higher than theoretical support, to focus on the effect of queue length (L) on QoS and QoE in DCTPQ based on the stochastic behavior of ingress packets. The evaluation scenarios, with a fixed processing rate and varying queue lengths, are organized into six scenarios as presented in Table 1.

Table 1. Queue Length (L) (number of packets) and Processing Rate (R) (R=20,000 Pkts/Second) Setup

Interface	Queue (L)	S1	S2	S3	S4	S5	S6
Ethernet	L_{Q0}	One	64	256	256	512	512
	L_{Q1}	Queue	64	128	128	256	256
	L_{Q2}	64	64	64	64	128	128
Wi-Fi	L_{Q0}	One	64	256	512	256	512
	L_{Q1}	Queue	64	128	256	128	256
	L_{Q2}	64	64	64	128	64	128

Table 2. DT & RF Model Performance with Different Features

Features				Algorithm	Accuracy (%)	Precision (%)		Recall (%)		F-Score (%)	
PS	IPI	FS	IFI			CG	Non-CG	CG	Non-CG	CG	Non-CG
✓	✓	-	-	DT	93.5	96	91	92	95	94	93
✓	✓	-	-	RF	94.5	95	94	95	94	95	94
✓	✓	✓	-	DT	95.5	96	95	96	95	96	95
✓	✓	✓	-	**RF**	**97.6**	**98**	**97**	**98**	**97**	**98**	**97**
✓	✓	-	✓	DT	93.5	94	93	94	93	94	93
✓	✓	-	✓	RF	94.5	95	94	95	94	95	94
✓	✓	✓	✓	DT	93.5	96	91	93	95	94	93
✓	✓	✓	✓	RF	94.5	95	94	95	94	95	94

5. QoS & QoE Assessment

We assess DCTPQ performance by measuring and reporting QoS and QoE metrics. Due to the higher significance of downlink traffic on user QoE in CG, we focus on downlink traffic. We conducted 18 experimental runs, each lasting 300 seconds, involving simultaneous traffic from CG, UDP, and TCP (BBR, CUBIC). The experiments were organized into six different DCTPQ setups, each repeated for another 300 seconds. For CG traffic, we used three games: Fortnite, Forza, and Mortal Kombat. Each game's results are organized into separate tables, reflecting the three different experiments. Thus, we have six runs for each game (Fortnite, Forza, Mortal Kombat), totaling 18 runs. Each group of six runs represents one of the three games, with all six DCTPQ setups tested for each game.

5.1. QoS

QoS is measured using throughput, latency (ms), and sojourn time (µs) for CG, UDP, and TCP traffic, with TCP retransmissions reported for TCP only. Throughput for CG is extracted from PCAP files on client (1), and latency is measured as RTT/2 between client (1) and the Internet gateway using ICMP packets. UDP and TCP throughput and latency are obtained from Iperf3 data on client (2) and client (3), with latency derived as RTT/2 from Iperf3. Sojourn time for each queue (Q_0, Q_1, Q_2) is extracted from

INT probes. We report the average, variance, and 95th percentile for a clear comparison of QoS during each experiment run. The results are organized into three tables, Table 3 (Fortnite), Table 4 (Forza), and Table 5 (Mortal Kombat), and covering all six DCTPQ setups. For convenience, we show all the numbers for all the metrics, including variance, min and max values, and average at 95th percentile. To facilitate, we highlight in bold the best values for each scenario.

Table 3. QoS Metrics for Fortnite Downlink, UDP & TCP Traffic

S_i	Traffic Types	Throughput (Mbps)			Latency (ms)			Queue Sojourn Time (μs)			TCP Retr
		Avg.	Var.	95th	Avg.	Var.	95th	Avg.	Var.	95th	
S1	CG Fortnite	7.22	10.42	**8.72**	65.92	155.54	91				-
	UDP (10 Mbps)	9.99	0.16	10.20	1.47	0.21	1.56				-
	TCP [CCA=bbr]	17.26	30.81	20.63	1080	309000	1050.35	**27.73**	**59.51**	**38.60**	**582**
	TCP [CCA=cubic]	10.41	17.96	13.23	1.190	6.75E+06	1049.75				1310
S2	CG Fortnite	5.15	**2.94**	6.31	38.84	45.06	54.5	63.06	12657	155.6	-
	UDP (10 Mbps)	9.67	1.40	10.30	0.91	0.07	0.98	85.32	20827	228.70	-
	TCP [CCA=bbr]	3.59	12.21	4.17	1050	48300	1050	64.67	6240	153	**1389**
	TCP [CCA=cubic]	8.06	17.21	10.68	1057.18	54300	1050				3669
S3	CG Fortnite	6.19	5.67	7.66	19.83	45.28	22.87	60.19	4436	160.50	-
	UDP (10 Mbps)	9.68	1.32	10.30	0.88	0.17	0.93	93.45	145000	176.75	-
	TCP [CCA=bbr]	5.86	20.90	8.17	1050	11100	1050	67.86	9696.47	228.35	2300
	TCP [CCA=cubic]	7.44	15.42	9.94	1050	10600	1050				**2221**
S4	CG Fortnite	4.89	3.95	5.94	19.73	58.10	24.57	79.26	26800	167	-
	UDP (10 Mbps)	9.88	0.81	10.30	1.01	0.40	1.02	108.64	50500	562.80	-
	TCP [CCA=bbr]	4.83	11.23	5.84	1050	56000	1050	92.42	25139.13	447	**563**
	TCP [CCA=cubic]	9.13	13.36	11.20	1060	65500	1050				1395
S5	CG Fortnite	**7.33**	4.75	8.56	**19.58**	**22.51**	**21.70**	79.34	5470	247.80	-
	UDP (10 Mbps)	9.54	0.98	10.10	0.91	0.41	0.92	67.13	7520	216.25	-
	TCP [CCA=bbr]	3.83	12.85	4.14	1037.77	6951.91	1049.26	116.03	110E+03	201.05	**1388**
	TCP [CCA=cubic]	6.06	13.45	9.24	1034.20	3691.25	1048.71				1949
S6	CG Fortnite	6.14	4.80	7.71	24.49	38.28	35.69	71.33	8690	229.15	-
	UDP (10 Mbps)	9.60	1.48	10.30	0.94	0.52	1.01	75.12	14300	279.10	-
	TCP [CCA=bbr]	4.04	9.97	4.99	1110	17400	1050	265.22	368E+04	348.15	**1639**
	TCP [CCA=cubic]	6.43	11.35	8.69	1050	37400	1050				2402

Table 4. QoS Metrics for Forza Downlink, UDP & TCP Traffic

S_i	Traffic Types	Throughput (Mbps)			Latency (ms)			Queue Sojourn Time (μs)			TCP Retr
		Avg.	Var.	95th	Avg.	Var.	95th	Avg.	Var.	95th	
S1	CG Forza	2.84	1.63	3.35	38.43	30.57	52.49				-
	UDP (10 Mbps)	9.85	4.02	10.20	1.46	0.51	1.48	**37.21**	**851.66**	**57.4**	-
	TCP [CCA=bbr]	15.12	58.89	20.20	1044.71	3628.87	1049.80				716
	TCP [CCA=cubic]	10.71	31.83	14.30	1122.69	1843511.00	1049.74				**571**
S2	CG Forza	2.50	0.73	2.95	26.61	22.69	33	1411	1142E+03	224.4	-
	UDP (10 Mbps)	9.90	0.71	10.30	0.91	0.20	0.96	148.97	742E+03	206.45	-
	TCP [CCA=bbr]	5.70	20.73	7.76	1100	953E+03	1050	185	142E+04	237	1574
	TCP [CCA=cubic]	9.93	18.78	12.05	1110	131E+04	1050				**857**
S3	CG Forza	2.52	**0.68**	3.01	13.7	**19.91**	9.39	63.40	7587.79	181.80	-
	UDP (10 Mbps)	9.89	1.17	10.50	1.02	0.31	1.04	79.96	20100	273.00	-
	TCP [CCA=bbr]	4.69	15.13	5.84	1060.67	124022.91	1049.39	89.28	18300	295.2	**422**
	TCP [CCA=cubic]	12.00	20.07	14.80	1064.63	121952.20	1050.24				2305
S4	CG Forza	2.58	1.03	3.01	**7.09**	38.96	9.69	69.34	10500	294.2	-
	UDP (10 Mbps)	9.89	0.85	10.40	0.98	0.54	0.97	194.55	343E+04	228.20	-
	TCP [CCA=bbr]	5.85	18.67	8.12	1090	739E+03	1049.44	74.47	17900	186.6	1273
	TCP [CCA=cubic]	9.33	22.00	11.80	1160	377E+04	1049.72				**1166**
S5	CG Forza	2.43	0.84	2.94	16.62	80.61	**8.63**	70.17	17000	258	-
	UDP (10 Mbps)	9.96	1.03	10.43	0.92	0.22	0.98	78.47	13500	327.90	-
	TCP [CCA=bbr]	5.82	17.35	7.56	1050	45900	1049.47	66.43	7438.20	209.4	871
	TCP [CCA=cubic]	9.95	19.05	12.40	1080	297E+03	1050				**414**
S6	CG Forza	**3.45**	3.53	**3.94**	22.09	36.28	36.91	99.14	131000	178.5	-
	UDP (10 Mbps)	9.96	1.03	10.43	1.05	0.57	1.03	84.79	59800	242	-
	TCP [CCA=bbr]	5.82	17.35	7.56	1040	21400	1048.98	76.76	24000	215	**449**
	TCP [CCA=cubic]	9.95	19.05	12.40	1070	167E+03	1050				986

5.2. QoE

Player experience during gameplay is measured using VMAF scores, which compare video quality under different conditions. The reference video, recorded without congested TCP and UDP traffic, has a resolution of 1366x768 and a frame rate of 30 fps. Using the default VMAF model, combining various metrics to predict perceived video quality, we compare captured videos from scenarios S_1 to S_6, extracting VMAF scores from 0 to 100. A VMAF score above 70 is generally considered acceptable. The VMAF scores for different scenarios are shown in Fig. 3.

6. Discussion

In this section, we discuss the findings based on the QoS and QoE metrics presented in the previous section.

Table 5. QoS Metrics for Mortal Kombat Downlink, UDP & TCP Traffic

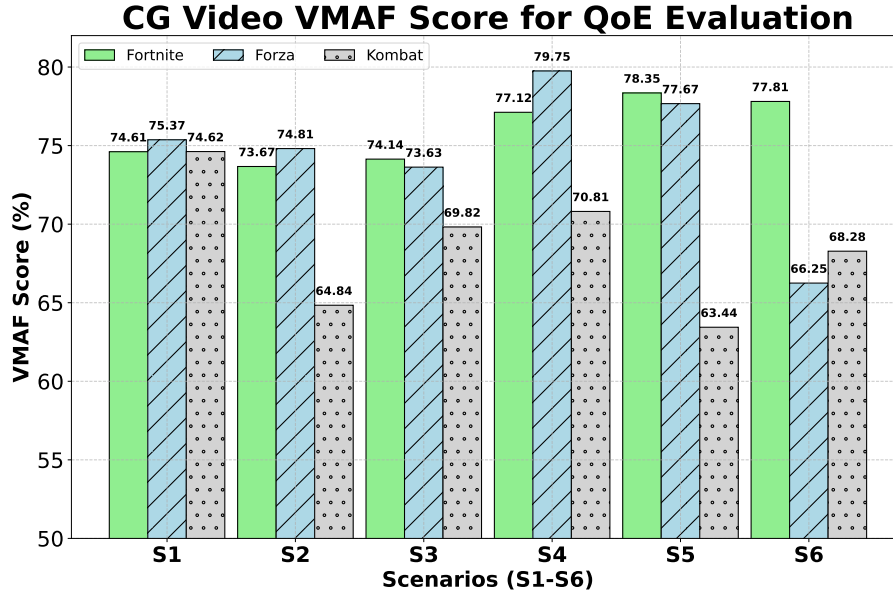| S_i	Traffic Types	Throughput (Mbps)			Latency (ms)			Queue Sojourn Time (μs)			TCP Retr
		Avg.	Var.	95th	Avg.	Var.	95th	Avg.	Var.	95th	
S1	CG Kombat	5.55	26.96	**9.49**	**48.36**	**20.88**	55.75	2919.80	788E+06	389	-
	UDP (10 Mbps)	9.32	11.68	10.20	1.44	0.27	1.46				-
	TCP [CCA=bbr]	6.70	55.01	9.60	1038.47	3707.98	1050				**463**
	TCP [CCA=cubic]	7.92	33.24	12.40	1219.30	9625553	1050				1252
S2	CG Kombat	**7.38**	10.05	8.90	106.5	40.35	160.2	**69.02**	**5910**	267	-
	UDP (10 Mbps)	9.62	1.77	10.40	0.98	0.41	0.98	107.20	113000	226.50	-
	TCP [CCA=bbr]	2.60	6.08	2.82	1029.56	3655.58	1047.75	75.35	7897.07	245.45	**574**
	TCP [CCA=cubic]	6.55	19.66	9.60	1047.74	13421.56	1049.59				3204
S3	CG Kombat	6.69	10.29	8.80	63.68	55.96	**36.29**	87.64	42400	217.45	-
	UDP (10 Mbps)	9.49	1.44	10.10	1.04	0.16	1.10	77.23	12500	229.50	-
	TCP [CCA=bbr]	3.74	9.73	4.88	1110	1.50E+06	1050	76.06	26600	234.20	**1541**
	TCP [CCA=cubic]	7.28	27.43	10.40	1140	286E+04	1050				2231
S4	CG Kombat	6.87	**9.12**	8.72	50.02	52.01	61.02	113.34	198E+03	339.6	-
	UDP (10 Mbps)	9.68	1.11	10.30	0.94	0.23	1.00	84.53	21300	263.80	-
	TCP [CCA=bbr]	5.06	20.04	6.80	1050	54400	1050	87.79	23000	322.60	**1034**
	TCP [CCA=cubic]	6.96	21.48	9.83	1080	412E+03	1050				1660
S5	CG Kombat	6.87	**9.12**	8.72	116.29	58.25	126.02	70.66	18400	**166.75**	-
	UDP (10 Mbps)	9.48	1.55	10.20	0.94	0.29	0.96	87.41	41900	225.70	-
	TCP [CCA=bbr]	3.99	15.01	4.76	1043.90	29600	1.05E+03	119.72	173E+03	231.70	**1350**
	TCP [CCA=cubic]	5.55	15.58	8.67	1185.85	667E+04	1050				2284
S6	CG Kombat	6.14	9.60	8.16	123.25	30.29	184.94	220.48	243E+04	205.2	-
	UDP (10 Mbps)	9.48	1.55	10.20	0.89	0.19	0.98	455.37	232E+05	282.15	-
	TCP [CCA=bbr]	3.99	15.01	4.76	1044.42	22600	1050	78.27	21200	197.40	**1744**
	TCP [CCA=cubic]	5.55	15.58	8.67	1077.28	439E+03	1050				2400

Figure 3. VMAF score for different games with different Queue setup

6.1. CG Downlink Traffic QoS Analysis.

QoS metrics, including throughput (Mbps), latency (ms), and sojourn time (μs), measure the overall QoS. Higher throughput and lower latency typically indicate higher QoS. Lower variance in throughput and latency ensures smooth and stable traffic, which is critical for CG applications. Comparing the single queue scenario (S_1) with multi-queue DCTPQ scenarios (S_2-S_6), we note that sojourn time is inversely related to throughput according to Little's law (sojourn time $\sim \frac{1}{\text{Throughput}}$). In S_1, all traffic types share a single queue, resulting in higher throughput and lower sojourn time. In contrast, S_2-S_6 separate TCP, UDP, and CG traffic into distinct queues, leading to potentially higher sojourn times for each type but improved overall QoS due to reduced congestion and better traffic distribution across different queues. This distinction is crucial for our analysis of Fortnite, Forza, and Mortal Kombat.

- **Fortnite.** According to the findings reported in Table 3, S_5 achieves the highest throughput (7.33 Mbps) compared to the single queue scenario S_1. The variance in throughput is significantly lower in S_5 (4.75 Mbps) than in S_1 (10.42 Mbps), indicating smoother traffic in S_5. Minimum latency (19.58 ms) and latency variance (22.51 ms) also in S_5 result in smoother

video streaming, as supported by the VMAF score. Although sojourn time increases in S_5 compared to S_1 (as expected), the queue length remains the same. The difference lies in the input rate, which is divided among three different rates λ_{max} in S_5 based on little's law, leading to better traffic distribution;

- **Forza.** According to the findings reported in Table 4, the highest throughput is observed in S_6 (3.45 Mbps), surpassing S_1 (2.84 Mbps). Although S_6 shows higher throughput variance (3.53 Mbps) compared to S_1 (1.63 Mbps), the lowest latency is achieved in S_4 (7.09 ms) and the lowest latency variance in S_3 (19.91 ms). This low latency is crucial for CG video quality. Sojourn time increases in DCTPQ scenarios compared to S_1 (as expected) due to higher overall throughput in one shared queue in S_1. The highest throughput in S_6, the lowest throughput variance in S_3, and the lowest latency in S_4 indicate that multi-queue DCTPQ provides better QoS than the single queue scenario. The higher VMAF score in S_4 emphasizes the importance of low latency for QoE, even with slightly lower throughput.

- **Mortal Kombat.** According to Table 5, the highest throughput for Mortal Kombat in the DCTPQ setup is seen in S_2 (7.38 Mbps), while S_4 and S_5 exhibit the lowest throughput variance (9.12 Mbps). Although S_1 exhibits the lowest latency (48.36 ms) and lowest latency variance (20.88 ms), S_4 shows a similar latency (50.02 ms). Unlike Fortnite and Forza, Kombat shows reduced sojourn time in S_2 compared to S_1, which is counter-intuitive and indicates different traffic behavior. Despite higher throughput with lower variance in multi-queue scenarios, the VMAF score is higher in S_1, suggesting that DCTPQ doesn't enhance QoS for Kombat as effectively, highlighting its higher resource requirements.

6.2. CG Traffic QoE Analysis.

The VMAF scores for single queue (S_1) and multi-queue setups (S_2-S_6) in Fig. 3 show the impact of network congestion on video quality. Different CG games yielded varying VMAF scores, highlighting the influence of game type and genre on QoE.

For Fortnite, S_5 maintained a high VMAF score of 78.35, with a 95th percentile throughput of 8.56 Mbps, the lowest latency of 19.58 ms, and the lowest variance of 22.51 ms. High average (79.34 μs) and 95th percentile (247.80 μs) sojourn times helped mitigate latency variance effects.

Forza achieved the highest VMAF score (79.75) in S_4, which had a larger CG queue, indicating sensitivity to jitter. S_4 had a bandwidth requirement of 2.58 Mbps, with low latency (7.09 ms) and variance (38.96 ms). Its average sojourn time (69.34 μs) is high, and its 95th percentile of sojourn was the highest (294.2 μs).

Mortal Kombat in S_1 had throughput ranging from 5.55 Mbps to 9.49 Mbps with high variance (29.96 Mbps), showing significant bandwidth demands. S_4 had the second-highest VMAF score (70.81), with the lowest throughput variance, leading to smoother bandwidth, latency, and sojourn time. Kombat required higher queue lengths and more resources, as indicated by its higher VMAF scores in S_4 and S_6.

Observations show a linear correlation between the 95th percentile sojourn time and VMAF scores for Fortnite and Forza, indicating that higher sojourn times correspond to higher VMAF scores and smoother, higher-quality video transfer in varying wireless rates. This pattern does not hold for Kombat.

7. Conclusion

This paper presents DCTPQ, a multi-queue architecture leveraging a RF model with 97.6% accuracy to classify CG traffic from non-CG UDP traffic in real-time. We evaluated both DT and RF models using different feature combinations (PS, IPI, FS, IFI) and achieved the best accuracy with RF using FS, PS,

and IPI. The model was deployed on P4Pi as a proof of concept, integrating dIL methodology to enable real-time classification and dynamic queue management at the edge. By assigning dedicated queues for TCP, UDP (non-CG), and CG traffic, the DCTPQ maintains smooth gameplay despite fluctuating bandwidth, dynamically adjusting queue rate and length to prioritize CG traffic while ensuring fair resource sharing. DCTPQ was evaluated across six scenarios, demonstrating improved QoS (higher throughput, lower latency) and better QoE for games like Fortnite and Forza, while Kombat performed better with a single queue due to its higher variance. VMAF scores revealed that longer sojourn times improve QoE, except for Kombat. Our methodology enables real-time QoS and QoE evaluation across different games over the duration of gameplay. Future work will integrate feature extraction and classification within the data plane and extend DCTPQ to Tofino with TNA architecture, addressing network operator bottlenecks alongside AP limitations for CG gamers.

Acknowledgment

References

Albisser, O., De Schepper, K., Briscoe, B., Tilmans, O., and Steen, H. (2019). Dualpi2-low latency, low loss and scalable throughput (l4s) aqm. *Proc. of Netdev*.

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. (2010). Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63–74.

Allen, A. O. (2014). *Probability, statistics, and queueing theory*. Academic press.

Bell Labs. L4s (low latency, low loss, scalable throughput). Accessed: 2024-07-26.

Bondarenko, O., De Schepper, K., Tsang, I.-J., Briscoe, B., Petlund, A., and Griwodz, C. (2016). Ultra-low delay for all: Live experience, live analysis. In *Proceedings of the 7th International Conference on Multimedia Systems*, pages 1–4.

Carvalho, M., Soares, D., and Macedo, D. F. (2023). Transfer learning-based qoe estimation for different cloud gaming contexts. In *2023 IEEE 9th International Conference on Network Softwarization (NetSoft)*, pages 71–79.

de Almeida, L. C., Matos, G., Pasquini, R., Papagianni, C., and Verdi, F. L. (2022). iRED: Improving the DASH QoS by dropping packets in programmable data planes. In *2022 18th International Conference on Network and Service Management (CNSM)*, pages 136–144.

De Schepper, K. and Briscoe, B. (2023). The explicit congestion notification (ecn) protocol for low latency, low loss, and scalable throughput (l4s). Technical report, RFC 9331. https://doi.org/10.17487/RFC9331.

Floyd, S. and Jacobson, V. (1993). Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, 1(4):397–413.

Gettys, J. and Nichols, K. (2011). Bufferbloat: Dark buffers in the internet: Networks without effective aqm may again be vulnerable to congestion collapse. *Queue*, 9(11):40–54.

Gil, G. D., Lashkari, A. H., Mamun, M., and Ghorbani, A. A. (2016). Characterization of encrypted and vpn traffic using time-related features. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*, pages 407–414. SciTePress Setúbal, Portugal.

Graff, P., Marchal, X., Cholez, T., Mathieu, B., and Festor, O. (2023). Efficient identification of cloud gaming traffic at the edge. In *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, pages 1–10. IEEE.

Graff, P., Marchal, X., Cholez, T., Mathieu, B., Tuffin, S., and Festor, O. (2024). Improving cloud gaming traffic qos: a comparison between class-based queuing policy and l4s. In *Network Traffic Measurement and Analysis Conference (TMA 2024)*, page 10. IEEE.

Jacobson, V. and Kathleen, N. (2012). Controlling queue delay-a modern aqm is just one piece of the solution to bufferbloat. *Asscociation for Computing Machinery (ACM Queue)*.

Jarschel, M., Schlosser, D., Scheuring, S., and Hoßfeld, T. (2011). An evaluation of qoe in cloud gaming based on subjective tests. In *2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 330–335. IEEE.

Kougioumtzidis, G., Vlahov, A., Poulkov, V. K., Lazaridis, P. I., and Zaharis, Z. D. (2024). Qoe prediction for gaming video streaming in o-ran using convolutional neural networks. *IEEE Open Journal of the Communications Society*, 5:1167–1181.

Marchal, X., Graff, P., Ky, J. R., Cholez, T., Tuffin, S., Mathieu, B., and Festor, O. (2023). An analysis of cloud gaming platforms behaviour under synthetic network constraints and real cellular networks conditions. *Journal of Network and Systems Management*, 31(2):39.

Nádas, S., Ernström, L., Szilágyi, L., Patra, G., Krylov, D., and Lynam, J. (2024). To qoe or not to qoe. In *Proceedings of the 2024 Applied Networking Research Workshop*, ANRW '24, page 38–44, New York, NY, USA. Association for Computing Machinery.

Netflix, I. Vmaf: The video multi-method assessment framework. accessed 2024-07-29.

Nokia (2024). Nokia and vodafone conduct world's first trial of l4s technology over an end-to-end pon network. Accessed: 2024-07-26.

Pan, R., Natarajan, P., Piglione, C., Prabhu, M. S., Subramanian, V., Baker, F., and VerSteeg, B. (2013). Pie: A lightweight control scheme to address the bufferbloat problem. In *IEEE 14th international conference on high performance switching and routing (HPSR)*, pages 148–155. IEEE.

Ramakrishnan, K., Floyd, S., and Black, D. (2001). The addition of explicit congestion notification (ecn) to ip. Technical report.

Rhee, I., Xu, L., Ha, S., Zimmermann, A., Eggert, L., and Scheffenegger, R. (2018). Cubic for fast long-distance networks. Technical report.

Shirmarz, A., Verdi, F. L., Singh, S. K., and Rothenberg, C. E. (2024). From pixels to packets: Traffic classification of augmented reality and cloud gaming. In *IEEE 10th International Conference on Network Softwarization (NetSoft)*, pages 195–203. IEEE.

Xu, X. and Claypool, M. (2022). Measurement of cloud-based game streaming system response to competing tcp cubic or tcp bbr flows. In *Proceedings of the 22nd ACM Internet Measurement Conference*, pages 305–316.