DaMPS: A Deadline-aware Multipath Packet Scheduler for Mobile Applications

Zhuoyue Chen*, Kechao Cai*, Jinbei Zhang*, John C.S. Lui†
*School of Electronics and Communication Engineering, Sun Yat-sen University, China
†Department of Computer Science & Engineering, The Chinese University of Hong Kong, Hong Kong
chenzhy225@mail2.sysu.edu.cn, {caikch3,zhjinbei}@mail.sysu.edu.cn, cslui@cse.cuhk.edu.hk

2024 IEEE 21st International Conference on Mobile Ad-Hoc and Smart Systems (MASS) | 979-8-3503-6399-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/MASS62177.2024.00031

***Abstract*—Many popular mobile applications, such as online gaming and live video streaming, impose strict deadlines on packets arrivals. Modern mobile devices are equipped with both Wi-Fi and cellular interfaces, facilitating multipath transmissions. However, many packet schedulers overlook the deadline requirements associated with multipath transmissions, which is crucial for enhancing the overall user experience of mobile applications. In this paper, we propose a *Deadline-aware Multipath Packet Scheduler*, DaMPS, designed specifically to prioritize packet delivery before their deadlines in mobile networks. DaMPS dynamically adjusts the sending order and path allocation for packets while adhering to deadline requirements, thus enhancing its performance in dynamic network environments. We implement DaMPS in MPQUIC and evaluate its effectiveness across various network conditions using Mininet. Our extensive experiment results demonstrate that DaMPS improves the deadline adherence performance by 27%∼149% with comparable high throughput and low latency compared with other schedulers.**

***Index Terms*—Deadline-aware, multipath, packet scheduler**

I. Introduction

Many mobile applications, such as video conferencing, online gaming, and virtual reality (VR), demand timely data delivery, where packets arriving after the deadlines become utterly worthless. Such deadline requirements significantly impact the user experience. For example, a video conference requires that the packet delay remains within 100ms to ensure seamless host-guest interaction [1]. Similarly, maintaining latency below 25ms is imperative to prevent user discomfort in VR applications [2]. Meanwhile, mobile devices equipped with both Wi-Fi and cellular interfaces can transmit packets across multiple paths using multipath transmission protocols. These protocols are designed to aggregate bandwidth on multiple paths and provide reliable and low-latency data transmission. Numerous studies have investigated these protocols across various layers, including multi-link PPP [3] at the link layer, MPTCP [4] at the transport layer, and MPQUIC [5] at the application layer.

Multipath packet schedulers in the multipath transmission protocols play a critical role as they determine the proper path for each data packet, thereby impacting protocol performance. For instance, in the widely adopted protocols, MPTCP and MPQUIC, the default packet scheduler, known as minRTT [6], selects the path with the minimum Round-Trip Time (RTT) to reduce data transmission latency. Another scheduler, ReMP [7], aims to enhance reliability by transmitting redundant packets over all paths. Additionally, various schedulers pursue diverse objectives, including achieving high throughput [8], adapting to network fluctuations [9], [10], and facilitating seamless handoff [11], by using deep reinforcement learning and online learning techniques.

However, previous multipath packet schedulers neglect the deadline requirements in mobile applications and lack consideration of the heterogeneity in Wi-Fi and cellular paths in mobile networks. This oversight can lead to performance degradation, particularly due to the Head-of-Line (HoL) problem [12], where early-arriving packets on the Wi-Fi path are held up because of the delayed arrival of packets with smaller sequence numbers on the cellular path. While schedulers such as ECF [13], BLEST [14] and ETC [15] have attempted to mitigate the impact of HoL blocking by estimating packet arrival times on different paths, they still fall short in ensuring timely packet delivery before the packet deadlines. Besides, the characteristics of paths, such as bandwidth and latency, vary over time due to the stochastic nature of wireless networks. This results in varying bandwidth and RTT of Wi-Fi and LTE paths [9], [16]. The performance of schedulers can degrade because of this dynamic nature of paths. It is important to note that improperly designed packet schedulers can significantly impair transmission performance [17], [18]. Therefore, there is a critical need to design a tailored scheduler that considers both the deadline requirements and the varying conditions of different paths for mobile applications.

In this paper, we propose a novel deadline-aware multipath packet scheduler, DaMPS, to achieve high throughput while adhering to the deadline requirements in multipath data transmissions. DaMPS aims to make optimal scheduling decisions to ensure the delivery of more packets before their deadlines. To achieve this, DaMPS can drop packets unlikely to meet their deadlines, thereby reallocating bandwidth more efficiently to other packets. Moreover, DaMPS can strategically defer the transmission of packets with less stringent deadlines and prioritize those with tighter deadlines to increase the proportion of timely delivered packets. We formulate the multipath packet scheduling problem with deadline requirements as an optimization problem and introduce three modules within DaMPS shown in Fig. 1 to efficiently tackle this challenge. First, we incorporate a *Fluctuation Monitor* module into DaMPS to adapt to network variations. Then, we invoke a *Packet Scheduling Solver* module to find an optimal packet scheduling matrix. Finally, we refine the scheduling

2155-6814/24/$31.00 ©2024 IEEE
DOI 10.1109/MASS62177.2024.00031

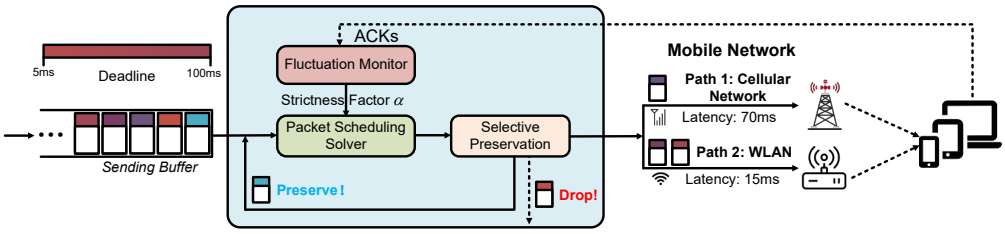


Fig. 1: An overview of DaMPS.

decisions for certain packets with a *Selective Preservation* module. Specifically, in the *Fluctuation Monitor* module, we model the fluctuation monitoring problem on each path as a Multi-Armed Bandit (MAB) problem, where arms represent the parameters tuning the one-way delay estimates for each path, and the reward function takes account of the packets that adhere to the deadline requirements and the packets that are decided not to transmit. To fine-tune the one-way delay estimates for each path, we utilize the UCB algorithm [19], facilitating parameter selection for optimal performance.

Our main contributions in this paper can be summarized as follows:

- We propose a deadline-aware multipath packet scheduler, DaMPS, which aims at delivering more packets before their deadlines.
- We introduce three modules into DaMPS for finding proper multipath packet scheduling decisions. Especially, our MAB-based *Fluctuation Monitor* module can facilitate effective adaptation to network variations.
- We implement DaMPS in MPQUIC and conduct extensive experiments to evaluate its performance alongside existing schedulers across dynamic and heterogeneous network environments. Our experiment results show that DaMPS outperforms existing schedulers in terms of deadline adherence and achieves 27%$\sim$149% improvements compared with other schedulers in different levels of network variations.

The rest of this paper is organized as follows. In Sec. II, we present the formulation of our multipath packet scheduling problem with deadline requirements and elaborate on the details of DaMPS. We describe the implementation details of DaMPS in Sec. III. In Sec. IV, we evaluate the performance of DaMPS and other schedulers. Sec. V reviews related works in the field. Finally, Sec. VI concludes this paper.

II. Problem Formulation and Scheduler Design

In this section, we present the details of DaMPS. We first formulate the multipath packet scheduling problem with deadline requirements as an optimization problem. Then we describe the three modules within DaMPS for tackling this multipath packet scheduling problem. Finally, we detail our novel *Fluctuation Monitor* module for adapting to network variations.

A. Deadline-aware Multipath Packet Scheduling Problem

In our multipath scheduler, once the packets accumulated in the sending buffer are packed as batches of certain size, the scheduler can determine the paths to send these batches. We denote the time for scheduling such a batch of packets as a *decision interval*.

High throughput objective. Let M represent the batch size and N denote the number of the available paths. At each decision interval, the scheduler computes a *scheduling matrix* for the M packets in the batch, denoted by $\mathbf{S} = [s_{ij}]$ with $s_{ij} \in \{0, 1\}$, indicating whether packet i in the batch is scheduled to path j ($s_{ij} = 1$) or not ($s_{ij} = 0$) for $1 \leq i \leq M$ and $1 \leq j \leq N$. Denote d_i as the deadline for packet i and τ_j as the one-way delay estimate of path j. For each batch of packets, the objective of our deadline-aware multipath packet scheduler is to maximize the number of packets that can be delivered timely before their deadlines, i.e., $\max_{\mathbf{S}} \sum_{i=1}^{M} \sum_{j=1}^{N} s_{ij} \cdot \mathbb{1}(\tau_j < d_i)$ as shown in (1a), where $\mathbb{1}(\mathcal{E}) = 1$ if event $\mathcal{E}$ holds and $\mathbb{1}(\mathcal{E}) = 0$ otherwise. It means that only the packet whose delay is within its deadline needs to be scheduled.

Packet scheduling constraints. At each decision interval, to avoid network congestion, the number of scheduled packets to a path should not exceed the remaining size of the congestion window at that path. Let c_j denote the remaining size of the congestion window at path j. Thus, we have the constraint in (1b). Moreover, the total number of scheduled packets should not exceed the batch size M as shown in the constraint (1c) and each packet in the batch can be scheduled to at most one path as constrained in (1d).

In summary, our deadline-aware multipath packet scheduling problem can be formulated as an optimization problem as follows:

$$\max_{\mathbf{S}} \sum_{i=1}^{M} \sum_{j=1}^{N} s_{ij} \cdot \mathbb{1}(\tau_j < d_i) \tag{1a}$$

$$\text{subject to} \sum_{i=1}^{M} s_{ij} \leq c_j \text{ for } j = 1, \ldots, N, \tag{1b}$$

$$\sum_{i=1}^{M} \sum_{j=1}^{N} s_{ij} \leq M, \tag{1c}$$

$$\sum_{j=1}^{N} s_{ij} \leq 1 \text{ for } i = 1, \ldots, M. \tag{1d}$$

Note that the optimization problem in (1) is an integer programming problem due to the integer constraint, $s_{ij} \in \{0, 1\}$. By relaxing this constraint to $s_{ij} \in [0, 1]$, one can reframe the problem in (1) as a linear programming (LP) problem and efficiently compute its optimal solution. In particular, this optimal solution is exactly the optimal packet scheduling

matrix for the problem in (1) as it resides within the feasible set of the problem in (1) [20]. Next, we propose three modules within DaMPS to find proper multipath packet scheduling decisions.

B. Modules within DaMPS

Fig. 1 shows an overview of DaMPS. Upon receiving ACKs from the mobile client, the *Fluctuation Monitor* module estimates the one-way delay of the N paths. When the packets accumulated in the sending buffer can be packed as a batch, the *Packet Scheduling Solver* module computes an optimal packet scheduling matrix by solving a linear programming problem relaxed from (1). Then, the *Selective Preservation* module determines whether the packets with no feasible paths should be preserved or dropped.

Algorithm 1 shows how the three modules operate together in DaMPS at decision interval t, $t \geq 1$. At decision interval t, in Line 1, DaMPS estimates the one-way delays, $\boldsymbol{\tau}$, of all the N paths with *Fluctuation Monitor* (detailed later in Sec. II-C). In Line 2, using the delay estimates $\boldsymbol{\tau}$, the remaining sizes of congestion windows $\boldsymbol{c}$, and the packet deadlines $\boldsymbol{d}$, DaMPS invokes the *Packet Scheduling Solver* to compute an optimal scheduling matrix $\mathbf{S}$ for the M packets in the batch across N paths with a linear relaxation (described in Sec. II-A) from the problem in (1).

However, the optimal packet scheduling matrix $\mathbf{S}$ may render infeasible paths for certain packets. This could occur for two reasons. First, some packets may have already passed their deadlines regardless of the scheduled path in $\mathbf{S}$, making further transmission wasteful of bandwidth. In this case, it is more efficient to drop these packets. Second, the current congestion window may be insufficient to handle the packets in the batch. In this case, these packets should be preserved in the sending buffer until additional congestion window space becomes available.

To better decide to drop or preserve the packets with no feasible paths in $\mathbf{S}$, we devise the *Selective Preservation* module in DaMPS as shown in Line 3-17 in Algorithm 1. Specifically, DaMPS maintains two Boolean variables for each packet in Line 4. If packet i can be scheduled to path j ($s_{ij} = 1$), it is transmitted via path j as shown in Line 6-9. Otherwise, packet i cannot be scheduled to path j ($s_{ij} = 0$) and if its deadline is far behind path j's one-way delay estimate (e.g., $d_i \geq 3\tau_j$), packet i is preserved in the sending buffer and rescheduled in the next decision interval since it has high probability of being transmitted on time in the next decision interval as shown in Line 10-15. At last, if packet i is marked as not transmittable (*isTransmitted* is **false**), DaMPS drops it to avoid bandwidth wastage as demonstrated in Line 16-17.

In the next subsection, we introduce the essentials in the *Fluctuation Monitor* module to facilitate DaMPS adapting to different levels of network variations.

C. Fluctuation Monitor Module

Note that the efficacy of the *Packet Scheduling Solver* relies on the accuracy of the delay estimates, $\boldsymbol{\tau}$, for the

Algorithm 1 DaMPS at decision interval t

Input: packet deadlines $\boldsymbol{d} = (d_1, \ldots, d_M)$, remaining congestion window sizes $\boldsymbol{c} = (c_1, \ldots, c_N)$
Output: packet scheduling matrix $\mathbf{S}$
$\triangledown$ ***Fluctuation Monitor***: *Estimate the one-way delays, $\boldsymbol{\tau}$, of N paths. Details are in Sec. II-C.*
1: $\boldsymbol{\tau} \leftarrow$ FluctuationMonitor()
$\triangledown$ ***Packet Scheduling Solver***: *Find an optimal packet scheduling matrix by solving an LP relaxed from the problem in* (1) *with $s_{ij} \in [0, 1]$ given $\boldsymbol{d}$, $\boldsymbol{c}$ and $\boldsymbol{\tau}$.*
2: $\mathbf{S} \leftarrow$ PacketSchedulingSolver($\boldsymbol{d}, \boldsymbol{c}, \boldsymbol{\tau}$)
$\triangledown$ ***Selective Preservation***: *Preserve or drop the packets with no feasible paths.*
3: **for** $i = 1, \ldots, M$ **do**
4: *isTransmitted* $\leftarrow$ **false**, *isPreserved* $\leftarrow$ **false**
5: **for** $j = 1, \ldots, N$ **do**
6: **if** $s_{ij} = 1$ **then**
7: Transmit packet i via path j
8: *isTransmitted* $\leftarrow$ **true**, *isPreserved* $\leftarrow$ **false**
9: **break**
10: **else if** $d_i \geq 3\tau_j$ **then**
11: *isPreserved* $\leftarrow$ **true**
12: **end if**
13: **end for**
14: **if** *isPreserved* **then**
15: Preserve packet i in the sending buffer and reschedule it in the next decision interval
16: **else if** !*isTransmitted* **then**
17: Drop packet i
18: **end if**
19: **end for**

N paths. Underestimating delays may result in transmitting packets that are unlikely to meet their deadlines. Conversely, overestimation may lead to unnecessary discarding of packets that could have been delivered on time.

To provide accurate delay estimates in a time-varying network environment, we design the *Fluctuation Monitor* module. For path j ($1 \leq j \leq N$), the module first measures path j's one-way delay OWD_j by using an exponentially weighted moving average (EWMA) of the delay information in the ACKs. Let $\mathcal{K} = \{\alpha_1, \ldots, \alpha_K\}$ denote a set of candidate strictness factors and $\alpha_k > 0$ for $1 \leq k \leq K$. Then the module selects a *strictness factor* $\alpha_k^j \in \mathcal{K}$ and gives the one-way delay estimate by scaling the measured one-way delay as follows:

$$\tau_j = \alpha_k^j \times OWD_j. \tag{2}$$

Note that the strictness factor controls the conservativeness of the scheduling decision. A larger strictness factor (e.g., $\alpha_k^j > 1$) can enforce the *Packet Scheduling Solver* module using delay estimates larger than the measured delays and thereby ensuring stricter adherence to deadlines. Each path can exhibit a unique degree of fluctuation, requiring a tailored strictness factor. For a path with varying one-way delays, the

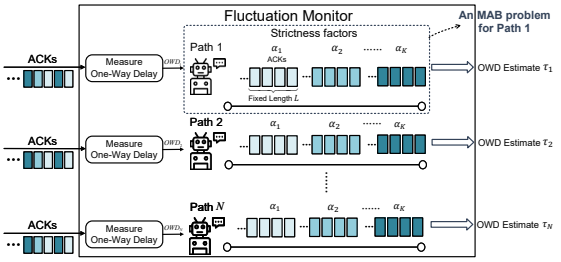


Fig. 2: MAB problems in the *Fluctuation Monitor* module.

Fluctuation Monitor module should select a larger strictness factor to better prevent packets on the path from missing their deadlines. On the contrary, for a path with stable delays, the *Fluctuation Monitor* module should favor a smaller strictness factor for more likely on time packet deliveries. In a word, the *Fluctuation Monitor* module must dynamically assess the stability of each path and select an appropriate strictness factor for each path.

We cast the selection of strictness factors in the *Fluctuation Monitor* module at each path as a multi-armed bandit problem, as illustrated in Fig. 2. This results in N independent MAB problems, one for each of the N paths. In the following, we present the design of arms and the reward functions of the arms for these MAB problems.

Arms: In an MAB problem, arms represent the actions that an learning agent can take at each round. In the *Fluctuation Monitor* module, the arms correspond to the strictness factors. For each path, the module must select a strictness factor at each decision interval from the candidate strictness factor set $\mathcal{K}$. Then it can provide a new one-way delay estimate by scaling the actual one-way delay according to Eq. (2).

Reward function: The reward function of an arm measures the value of the action taken by the learning agent. In the *Fluctuation Monitor* module, we expect that the reward of a chosen strictness factor can accurately characterize the performance of the scheduler resulting from that choice.

One potential reward metric is the *instantaneous deadline meeting ratio*, which is defined as the proportion of packets successfully delivered before their deadlines to the total packets acknowledged between two consecutive ACKs. The rationale is that: by selecting a good strictness factor, the scaled delay estimates can be accurate and the instantaneous deadline meeting ratio can reach 100%, which means all packets sent from the server adhere to their deadlines; conversely, if the chosen strictness factor is bad, the instantaneous deadline meeting ratio can drop below 100% and there must be deadline misses. Thus, the instantaneous deadline meeting ratio can effectively characterize the relation between the goodness of a strictness factor and DaMPS's performance. To avoid the noises in the instantaneous deadline meeting ratio, we calculate the deadline meeting ratio over a fixed length of ACKs, rather than relying on the instantaneous ones. Specifically, we define the *deadline meeting ratio* for the strictness factor α_k^j on path j as:

$$r_j^{DM}(t) = \frac{\sum_{a=t-L+1}^{A(\alpha_k^j)} nPktsDM_a}{\sum_{a=t-L+1}^{A(\alpha_k^j)} nPkts_a}, \tag{3}$$

where $A(\alpha_k^j)$ is the number of ACKs corresponding to the strictness factor α_k^j via the reward interaction mechanism (See details in Sec. III) and L is the fixed length. Additionally, $nPktsDM_a$ and $nPkts_a$ represent the number of packets that adhere to their deadlines and the total packets received in the a-th ACK, respectively.

While a large strictness factor may secure a higher deadline adherence, it could cause DaMPS prematurely dropping or preserving packets that can meet their deadlines. To address this issue, we introduce the *unsent packet ratio* measuring the proportion of unsent (dropped or preserved) packets in the batch at each decision interval. In particular, the *unsent packet ratio* for path j at decision interval t is defined as:

$$r_j^{UP}(t) = \frac{M - \sum_{i=1}^{M}\sum_{j=1}^{N} s_{ij}}{M}, \tag{4}$$

where M is batch size and s_{ij} is the scheduling decision for packet i on path j.

Finally, we can define the reward function of a strictness factor α_k^j on path j at decision interval t as:

$$f_{\alpha_k^j}(t) = \max\{r_j^{DM}(t) - r_j^{UP}(t), 0\}. \tag{5}$$

The reward function in Eq. (5) can characterize the effectiveness of a strictness factor in impacting the overall performance of DaMPS. A large reward indicates that the *Fluctuation Monitor* module with the selected strictness factor can give accurate delay estimates and help DaMPS strike a balance between a high deadline meeting ratio and a low unsent packet ratio.

Furthermore, in a dynamic network environment, the recent rewards should carry greater weight compared to the past rewards. This adjustment ensures that the scheduling decisions accurately reflect the current network conditions in a timely manner. Therefore, we introduce a discount factor $\gamma \in (0, 1]$ and define the *cumulative discounted reward* of strictness factor α_k^j on path j at the decision interval t as:

$$F_{\alpha_k^j}(t) = \sum_{t'=1}^{t} \gamma^{t-t'} f_{\alpha_k^j}(t') = \gamma F_{\alpha_k^j}(t-1) + f_{\alpha_k^j}(t), \tag{6}$$

and $F_{\alpha_k^j}(0) = 0$ for $\alpha_k^j \in \mathcal{K}$.

A bandit algorithm for the *Fluctuation Monitor* module on path j: Given the arms and the reward function we described above, the MAB problem for the *Fluctuation Monitor* module on path j is to select the proper strictness factors and maximize the cumulative discounted reward over T decision intervals. To achieve this goal, we adopt the UCB (Upper Confidence Bound) [19] algorithm to select strictness factors. At each decision interval t, the UCB algorithm selects action $\alpha_k^j(t)$ to maximize the sum of two terms as follows,

$$\alpha_k^j(t) = \arg\max_{\alpha_k^j \in \mathcal{K}} \frac{F_{\alpha_k^j}(t-1)}{n_{\alpha_k^j}(t-1)} + \sqrt{\frac{2\ln\sum_{k=1}^{K} n_{\alpha_k^j}(t)}{n_{\alpha_k^j}(t)}}, \tag{7}$$

Algorithm 2 Fluctuation Monitor on path j at t

Input: strictness factors $\mathcal{K} = \{\alpha_1, \ldots, \alpha_K\}$, discount factor $\gamma \in (0, 1]$
Output: one-way delay estimate τ_j of path j
1: Get the one-way delay measurement OWD_j of path j
2: **Initialize:** $n_{\alpha_k^j}(0) = 0$ and $F_{\alpha_k^j}(0) = 0$ for $\alpha_k^j \in \mathcal{K}$
3: **if** $t \leq K$ **then**
4: $\quad \alpha_k^j(t) = \alpha_k$ /* *Explore each strictness factor* */
5: **else**
6: $\quad$ Select the strictness factor $\alpha_k^j(t)$ as per Eq. (7)
7: **end if**
8: Update the delay estimate τ_j as per Eq. (2) with $\alpha_k^j(t)$
9: Obtain the feedback: deadline meeting ratio $r_j^{DM}(t)$ as per Eq. (3) and unsent packet ratio $r_j^{UP}(t)$ as per Eq. (4), and calculate the reward $f_{\alpha_k^j}(t)$ as per Eq. (5)
10: Update the discounted cumulative reward $F_{\alpha_k^j}(t)$ as per Eq. (6)
11: $n_{\alpha_k^j(t)}(t) = n_{\alpha_k^j(t)}(t-1) + 1$

where $n_{\alpha_k^j}(t)$ stands for the number of times the strictness factor α_k^j has been selected up to the decision interval t. The first term in Eq. (7) accounts for the average discounted reward of the strictness factor α_k^j. The second term in Eq. (7) involves a confidence interval according to Chernoff-Hoeffding bounds, aiding the exploration of strictness factors.

Finally, we describe how the *Fluctuation Monitor* module estimates the one-way delay by selecting the strictness factor for each path. Algorithm 2 shows the detail of the *Fluctuation Monitor* on path j at decision interval t. In Line 3-4, the module selects each strictness factor once to explore the impact of the different strictness factor on the reward of path j. After the exploration, in Line 5-7, the module selects the strictness factor $\alpha_k^j(t)$ with the highest UCB index according to Eq. (7). Then it updates the delay estimate τ_j according to Eq. (2) with $\alpha_k^j(t)$ in Line 8. Upon receiving the feedback, namely, the deadline meeting ratio $r_j^{DM}(t)$ and the unsent packet ratio $r_j^{UP}(t)$ from path j, it calculates the reward $f_{\alpha_k^j}(t)$ from path j according to Eq. (5) as shown in Line 9. Finally, the *Fluctuation Monitor* updates the cumulative discounted reward $F_{\alpha_k^j}(t)$ and the number of times the strictness factor $\alpha_k^j(t)$ has been selected in Line 10-11. With the *Fluctuation Monitor* deployed on all the paths, DaMPS can dynamically optimize packet scheduling decisions and quickly adapt to dynamic network environment.

III. Implementation

We incorporate DaMPS into MPQUIC [5], leveraging the *quic-go* framework.[1] Note that the path manager in MPQUIC can track the path information, including the congestion window (CWND) and RTT. The variations of CWND are related to the integrated congestion control algorithm (CCA) in MPQUIC. In our implementation, we adopt the OLIA [21] CCA and we retrieve the CWND of each path through the `pth.sentPacketHandler.GetCongestionWindow()` function. To get the number of bytes in flight, we utilize the `pth.sentPacketHandler.GetBytesInFlight()` function for calculating the remaining CWND of each path, which is equal to the CWND minus the bytes in flight. Similarly, we obtain the RTT of each path using the `pth.rttStats.SmoothedRTT()` function, which employs an exponentially weighted moving average (EWMA) filter for RTT measurement. We approximate the one-way delay measurement by halving the RTT value.

Reward interaction mechanism. In the *Fluctuation Monitor* module of DaMPS, network variations are evaluated by the server using the feedback from the client. Thus, the client is required to monitor the number of packets adhering to the deadline and include the deadline metadata in the ACKs transmitted to the server. To achieve this, we implement a *reward interaction mechanism.* To accurately determine the reward associated with each strictness factor on any given path, we associate the *packet number* with the packet's corresponding strictness factor. Upon receiving an ACK, we match the ACK with the relevant strictness factor based on ACK's *packet number*, and use the deadline information within the ACK to compute the reward. These adjustments are relatively minor and non-intrusive, with no adverse effects on data transmission in MPQUIC.

Cold-start mechanism. In addition, to get the critical path information in the beginning of the multipath packet transmission, such as CWND and RTT of newly added paths, we implement a *cold-start mechanism*. This mechanism involves sending redundant packets across both the newly established path and the existing paths. It can effectively mitigate the impact of initial uncertainty in the path information.

Fail-safe mechanism. In networks with extreme conditions, characterized by rapid and substantial bandwidth fluctuations, measuring delay and bandwidth accurately is challenging. Such bandwidth variations can misguide DaMPS into making impractical decisions, potentially leading to a data transmission blockage by selecting no path for all packets. To address this issue, we implement a *fail-safe mechanism*, where DaMPS can fall back to the Earliest Deadline First (EDF) scheduler temporarily if it makes impractical decisions for five consecutive decision intervals to prevent data transmission blockages.

IV. Evaluation

In this section, we evaluate the performance of DaMPS and other schedulers in an emulated environment. The testbed setup is presented in Section IV-A. The deadline adherence performance of DaMPS is evaluated in Section IV-B. We further examine the performance of the *Fluctuation Monitor* module in Section IV-C and the impact of schedulers on throughput and RTT in Section IV-D. Finally, we explore the

[1]Our implementation of DaMPS is open-source and available at https://github.com/MLCL-SYSU/Deadline-Packet-Scheduler.

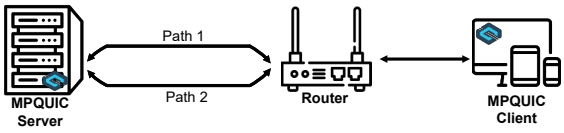


Fig. 3: Multi-path network topology setup.

DaMPS under the abruptly changing network conditions in Section IV-E.

A. Testbed Setup

Our testbed is based on Mininet 2.2.2 [22], which is installed and configured on Ubuntu 18.04. Our emulation setup adopts a typical network topology [5], [9], as illustrated in Fig. 3, where the router is equipped with two network interfaces connecting to the server and one to the client. To simulate different network conditions, we use the Linux Traffic Control (`tc` [23]) tool together with NetEM [24]. These tools allow us to configure bandwidth, one-way delay, RTT variation rate, and random loss rate of each path.

In our experiments, we set the batch size as $M = 6$, which indicates that the batch scheduler makes decisions for six packets at each decision interval. We use a synthetic trace as follows: the deadline of packets are set as the sum of $packed\ time$ and t_rand, where $packed\ time$ denotes the time when data is packed into packets, and t_rand is a random delay uniformly sampled from $[20, 50]$ms.

We compare the performance of DaMPS against other schedulers, including RR, minRTT [6], ECF [13], BLEST [14], Peekaboo[2] [9] and EDF. Specifically, RR sends packets in a round-robin manner across all available paths. MinRTT [6] prioritizes paths with the minimun RTT for packet transmission. ECF [13] focuses on minimizing the completion time of each packet. BLEST [14] attempts to avoid blocking by reducing the usage of slow paths. Peekaboo [9] leverages LinUCB [25] to make decisions based on network feedback, determining whether to wait for a better path or to transmit packets immediately. EDF prioritizes the packet with the tightest deadline and schedules it to the path with the smallest RTT. We do not include DAMS [26] in our comparison as it is designed for scheduling stream blocks rather than individual packets.

B. Deadline Adherence Performance

Table I: Parameters of Path 1 and Path 2

Parameter	Path 1	Path 2
Bandwidth (Mbps)	2	5
One-way Delay (ms)	30	10
RTT Variation (%)	0, 5, or 10	0, 10, 20, 30, or 40
Random Loss Rate (%)	1	1.5

We first study the deadline adherence performance of DaMPS *without* the *Fluctuation Monitor* module in bandwidth-limited scenarios, where the network resources are inadequate to deliver all packets before their deadlines. We configure the parameters for Path 1 and Path 2 as detailed in Table I, where the RTT variation is adjustable to characterize different levels of network dynamics. Note that only packets arriving before their deadlines are considered valid. Therefore, we use the *overall deadline meeting ratio* as the performance metric, calculated by dividing the number of packets received by the client before their deadlines by the total number of packets that have been received.

To illustrate DaMPS's performance under different levels of network dynamics, we compare the overall deadline meeting ratio of various schedulers under different RTT variation rates combinations, as shown in Fig. 4. The Y-axis is the overall deadline meeting ratio and the X-axis is the combination pairs of RTT variation rates on two paths: (0%, 0%), (5%, 10%), (5%, 20%), (10%, 30%), (10%, 40%), which respectively denote the RTT variation rates for Path 1 and Path 2. Fig. 4 shows that DaMPS improves the overall deadline meeting ratio by 27%$\sim$149% compared with other schedulers, which benefits from its deadline-aware scheduling decisions. In contrast, other schedulers such as RR, minRTT, ECF, BLEST and Peekaboo all have poor deadline adherence performance, because they all lack deadline awareness and send packets in a First-In-First-Out (FIFO) manner. Notably, Peekaboo, an online learning-based packet scheduler, exhibits the worst deadline adherence performance, potentially due to its excessive adoption of `wait` actions, which in turn leads to a substantial number of packets exceeding their deadlines. EDF achieves a slightly worse performance than DaMPS since it is unable to drop packets that are unlikely to meet their deadlines.

Moreover, we evaluate the deadline adherence performance of DaMPS under different RTT combinations. Specifically, we keep Path 1's RTT fixed at 20ms and vary Path 2's RTT from 20ms to 80ms with a step size 20ms. The bandwidth and random loss rate configurations remain the same as those specified in Table I, and RTT variation rates of both Path 1 and Path 2 are set at (0%, 0%). As shown in Fig. 5, DaMPS consistently outperforms other schedulers in different scenarios. For paths with the same RTT, all schedulers demonstrate a relatively good performance, as any path can ensure timely delivery for most packets. DaMPS and EDF outperform others in this case due to their ability to adjust the sending order for packet transmission. However, as the degree of path heterogeneity increases, DaMPS's overall deadline meeting ratio slightly declines (but is still higher than other schedulers). This is because fewer packets on Path 2 meet their deadlines, while most can only be delivered on time via Path 1. With more packets being transmitted on Path 1, an increased congestion can cause more packets to miss their deadlines, resulting in a decreased overall deadline meeting ratio.

In summary, as the path heterogeneity or the variations of both paths increase, the performance of DaMPS tends to degrade. Therefore, we incorporate the *Fluctuation Monitor* module into DaMPS to better adapt to network fluctuations and examine its impact in the following subsection.

[2]We evaluate Peekaboo using its open-source codebase available at [9].

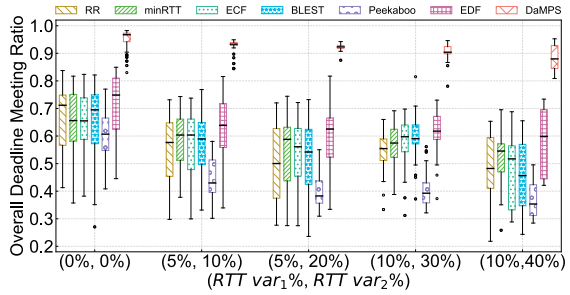


Fig. 4: Deadline adherence performance under different RTT variation rate combinations.

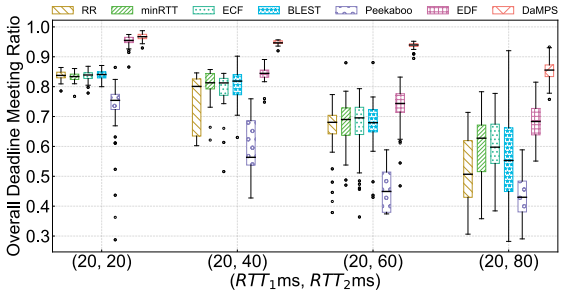
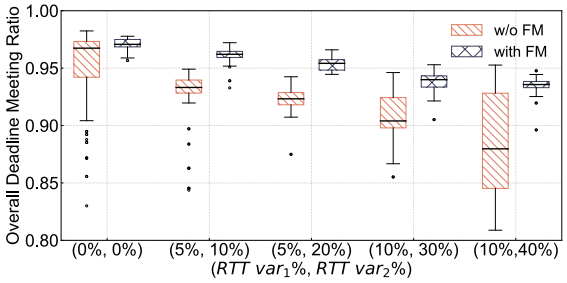
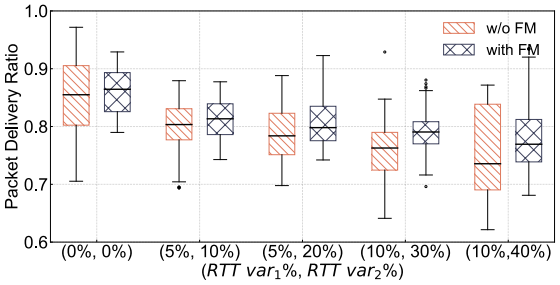


Fig. 5: Deadline adherence performance under different RTT combinations.



(a) Deadline adherence performance



(b) Packet delivery ratio

Fig. 6: Performance of DaMPS with and without (w/o) the *Fluctuation Monitor* (FM) module under different RTT variation rate combinations.

C. Impact of Fluctuation Monitor Module in DaMPS

In the *Fluctuation Monitor* module, we use four strictness factors, denoted as $\mathcal{A} = \{0.9, 1.0, 1.1, 1.2\}$. For the discount reward function, we set the discount factor as $\gamma = 0.8$, and the fixed length of ACKs as $L = 5$. To demonstrate the effectiveness of the *Fluctuation Monitor* module, we evaluate the deadline adherence performance of DaMPS both with and without the *Fluctuation Monitor* module under various combinations of RTT variations rates. The path parameters and the RTT variations rates are consistent with those described in Section IV-B.

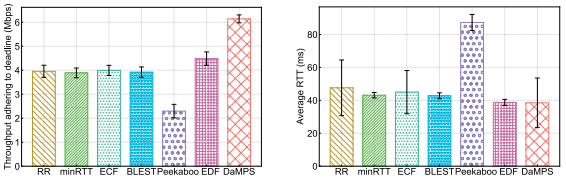
Fig. 6(a) illustrates that the incorporation of the *Fluctuation Monitor* module significantly improves the performance of DaMPS. With the *Fluctuation Monitor* (FM) module enabled, DaMPS achieves a higher percentage of packets being delivered on time. It indicates that the *Fluctuation Monitor* module effectively detects the network dynamics and gives accurate delay estimates for each path to adapt to network variations. Consequently, DaMPS can employ more aggressive policies in stable network conditions and adopt more conservative strategies when the network dynamics become more volatile.

To further investigate the impact of the *Fluctuation Monitor* module on packet delivery, we evaluate another deadline adherence performance metric, the *packet delivery ratio*, which can be calculated as $\frac{nPktsMD}{TotalPkts}$, where $nPktsMD$ is the number of packets delivered before their deadlines, and $TotalPkts$ represents the sum of the received and dropped packets. Fig. 6(b) shows that, regardless of the RTT variation rate combination, the packet delivery ratio with the *Fluctuation Monitor* module is higher that without the module. This indicates that the *Fluctuation Monitor* module facilitates the timely delivery of more packets, and has a negligible impact on the packets that can be delivered before their deadlines. On the contrary, it drops packets that are almost impossible to meet their deadlines, reallocating bandwidth to more viable packets.

In summary, the integration of the *Fluctuation Monitor* module enables DaMPS to effectively detect the network dynamics and generate appropriate policies, leading to improved deadline adherence performance in dynamic environments.

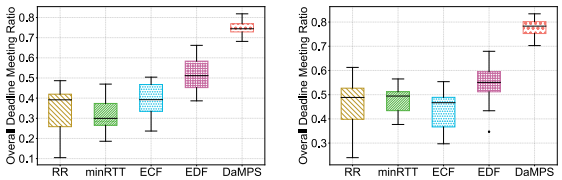
D. Impact of Packet Schedulers on Throughput and RTT

We study the impact of different packet schedulers on the throughput and RTT of the network connection between the server and the client. The setup for Path 1 and Path 2 is detailed in Table I, with RTT variations set at 5% and 10% respectively. We run the experiments 30 times and calculate the average throughput and RTT. In particular, only the packets that can meet their deadlines are counted in calculating the throughput. The results are shown in Fig. 7. In Fig. 7(a), DaMPS outperforms other existing schedulers by achieving the highest throughput (6.14Mbps) with deadline requirements. This is largely attributed to DaMPS's deadline-aware design, which ensures that the most of packets adheres to their



(a) Average throughput (b) Average RTT

Fig. 7: Average of throughput and RTT of different schedulers.



(a) Under changing bandwidth (b) Under changing RTT

Fig. 8: Deadline adherence performance under abrupt changing network conditions.

deadlines. Besides, EDF also achieves a higher bandwidth compared to other schedulers that lack deadline awareness. However, Peekaboo achieves the worst performance, which might because it selects too many `wait` actions.

Fig. 7(b) shows that the average RTT of all schedulers is comparable, which indicates that DaMPS improves deadline adherence performance without increasing network congestion. Notably, Peekaboo achieves the RTT exceeding 80ms, which is higher than the RTT of the individual paths, indicating potential congestion caused by Peekaboo.

E. Impact of Abrupt Changing Network Conditions

In this subsection, we evaluate the performance of different schedulers under abrupt changing network conditions, a scenario commonly encountered in mobile networks. First, we introduce abrupt variations in the bandwidth of two paths as follows: Path 1 experiences a drop to 0.5Mbps, followed by a restoration to 2Mbps, whereas Path 2 encounters a decrease to 1Mbps before returning to 5Mbps. These changes occur in a two-second cycle, with each state lasting one second. Second, to examine the impact of abrupt changes of RTTs on the schedulers, we conduct additional experiments with varying RTT settings: Path 1's RTT fluctuates between 100ms and 60ms over a two-second cycle, with each state maintained for one second, while Path 2's RTT is fixed at 20ms.

Fig. 8 shows the deadline adherence performance of the schedulers across various dynamic network environments, including both abruptly changing bandwidth and abruptly changing RTT scenarios. As shown in Fig. 8(a), despite that periodic network changes introduce inaccuracies in bandwidth and one-way delay measurements, DaMPS outperforms other schedulers due to the effective cooperation among its three modules, enabling resilience to changing network conditions. Fig. 8(b) demonstrates that even in networks with abruptly changing RTT, DaMPS still demonstrates superior performance compared to other schedulers.

V. Related Work

In recent years, many multipath transmission schemes have been developed, spanning multiple layers of network architecture, such as link, network, transport, application and cross layers [27]. To enhance the performance of multipath transmission, various packet schedulers have been proposed. For example, ECF [13] focuses on estimating packet arrival times to improve the utilization of the fastest path. BLEST [14] estimates the receiver buffer blocking and skips the slower path to prevent buffer blocking. ETC [15] allocates different data partitions across multiple paths to minimize the maximum completion time.

However, these schedulers do not account for the deadline requirements of packets, which significantly impact the user's experience. Therefore, there are several attempts to design deadline-aware schedulers. EDF [28] prioritizes tasks with the tightest deadlines. D3T [29] employs a deadline-aware scheduler alongside an adaptive DRL-based agent for FEC redundancy ratio and congestion control. DAMS [26] utilizes a variation of EDF for determining the sending order and detecting deadline satisfaction, effectively reducing bandwidth wastage. Although DAMS shares relevance with our DaMPS, the two schedulers are quite different and not directly comparable. Specifically, DAMS focus on managing stream blocks with hard-wired heuristic schemes, while DaMPS is designed for packet-level granularity, adapting various network conditions through optimization-based strategies.

Instead of designing a deadline-aware scheduler for multipath transmission, Cui et al. [30], [31] propose a single-path protocol, DTP, that supports block-based data delivery with deadlines and priorities. Moreover, they adopt the ADS scheduler and an adaptive redundancy mechanism in the DTP to provide better deliver-before-deadline service [32]. In addition, Liu et al. [33] formulate the task scheduling problem with deadline and throughput constraints as a constrained online learning problem. Beyond single-path protocol designs, John et al. [34] design the protocol, DMTP, for real-time applications with strict deadline requirements on the SCION [35] internet architecture. Tsanikidis et al. [36] introduce a novel framework for online scheduling and routing of deadline-constraint packets in wireless multi-hop networks.

Recently, many machine learning-powered packet scheduler have been proposed. In contrast to the hard-wired schedulers, learning-based schedulers exhibit adaptability and consistently perform well in various network scenarios. ReLes [8] utilizes asynchronous deep reinforcement learning techniques to derive packet scheduling policies. Peekaboo [9] and OLAPS [10] are both online learning-based schedulers. Peekaboo employs the LinUCB [25] algorithm to determine whether to transmit over an available path or wait a more advantageous one, while OLAPS uses the UCB [19] algorithm to decide whether to redundantly send packets.

Our DaMPS scheduler is different from previous studies as it delivers packets before their deadlines with optimized scheduling decisions and incorporates a specially designed bandit algorithm to adjust delay estimates for adapting to network variations.

VI. Conclusion and Future Work

In this paper, we propose DaMPS, a novel deadline-aware multipath packet scheduler aiming to deliver packets before their deadlines. DaMPS includes three main modules: the *Packet Scheduling Solver* module, the *Selective Preservation* module, and the *Fluctuation Monitor* module. The three modules operate together to provide proper packet scheduling decisions and adapt to different levels of network dynamics, achieving robust deadline adherence performance. Through extensive Mininet experiments, we show that DaMPS outperforms existing schedulers. For future work, we should consider the theoretical guarantees for our bandit algorithm and the heterogeneous monetary cost of multiple paths in mobile networks using real-world trace. Another interesting direction for future work is to reformulate the multipath packet scheduling problem as a constrained combinatorial multi-armed bandit problem.

Acknowledgements

This work was supported in part by National Key R&D Program of China under Grant 2022YFB2902700, NSF China (Grant No. 62202508, 62071501), and Shenzhen Science and Technology Program (Grant 20220817094427001, JCYJ20220818102011023, ZDSYS20210623091807023).

References

[1] M. Baldi and Y. Ofek, "End-to-end delay analysis of videoconferencing over packet-switched networks," *IEEE/ACM Transactions On Networking*, vol. 8, no. 4, pp. 479–492, 2000.

[2] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," in *Proc. of MobiCom 2017*, 2017, pp. 409–421.

[3] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti, "The ppp multilink protocol (mp)," Tech. Rep., 1996.

[4] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," Tech. Rep., 2013.

[5] Q. De Coninck and O. Bonaventure, "Multipath quic: Design and evaluation," in *Proc. of CoNEXT 2017*, 2017, pp. 160–166.

[6] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, "How hard can it be? designing and implementing a deployable multipath {TCP}," in *Proc. of NSDI 2012*, 2012, pp. 399–412.

[7] A. Frommgen, T. Erbshäußer, A. Buchmann, T. Zimmermann, and K. Wehrle, "Remp tcp: Low latency multipath tcp," in *Proc. of IEEE ICC 2016*. IEEE, 2016, pp. 1–7.

[8] H. Zhang, W. Li, S. Gao, X. Wang, and B. Ye, "Reles: A neural adaptive multipath scheduler based on deep reinforcement learning," in *Proc. of IEEE INFOCOM 2019*. IEEE, 2019, pp. 1648–1656.

[9] H. Wu, Ö. Alay, A. Brunstrom, S. Ferlin, and G. Caso, "Peekaboo: Learning-based multipath scheduling for dynamic heterogeneous environments," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 10, pp. 2295–2310, 2020.

[10] Y. Xing, K. Xue, Y. Zhang, J. Han, J. Li, and D. S. Wei, "An online learning assisted packet scheduler for mptcp in mobile networks," *IEEE/ACM Transactions on Networking*, 2023.

[11] J. Han, K. Xue, J. Li, R. Zhuang, R. Li, R. Yu, G. Xue, and Q. Sun, "Edar: An experience-driven multipath scheduler for seamless handoff in mobile networks," *IEEE Transactions on Wireless Communications*, 2023.

[12] K. Yedugundla, S. Ferlin, T. Dreibholz, Ö. Alay, N. Kuhn, P. Hurtig, and A. Brunstrom, "Is multi-path transport suitable for latency sensitive traffic?" *Computer Networks*, vol. 105, pp. 1–21, 2016.

[13] Y.-s. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "Ecf: An mptcp path scheduler to manage heterogeneous paths," in *Proc. of CoNEXT 2017*, 2017, pp. 147–159.

[14] P. Hurtig, K.-J. Grinnemo, A. Brunstrom, S. Ferlin, Ö. Alay, and N. Kuhn, "Low-latency scheduling in mptcp," *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 302–315, 2018.

[15] H. Zeng, L. Cui, F. P. Tso, and Z. Zhang, "Optimizing multipath quic transmission over heterogeneous paths," *Computer Networks*, vol. 215, p. 109198, 2022.

[16] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan, "Wifi, lte, or both? measuring multi-homed wireless internet performance," in *Proc. of IEEE INFOCOM 2014*, 2014, pp. 181–194.

[17] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath tcp schedulers," in *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, 2014, pp. 27–32.

[18] P. B. Diakhate, T.-T. Chu, M. A. Labiod, B. Augustin, H.-A. Tran, and A. Mellouk, "Experimental evaluation of multiple multipath schedulers over various urban mobile environments," in *Proceedings of the 11th International Symposium on Information and Communication Technology*, 2022, pp. 201–207.

[19] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, pp. 235–256, 2002.

[20] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[21] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "Opportunistic linked-increases congestion control algorithm for mptcp," *draft-khalili-mptcp-congestioncontrol-02*, 2014.

[22] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proc. of CoNEXT 2012*, 2012, pp. 253–264.

[23] B. Hubert *et al.*, "Linux advanced routing & traffic control howto," *Netherlabs BV*, vol. 1, pp. 99–107, 2002.

[24] S. Hemminger *et al.*, "Network emulation with netem," in *Linux conf au*, vol. 5, 2005, p. 2005.

[25] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. of WWW 2010*, 2010, pp. 661–670.

[26] X. Zuo, Y. Cui, X. Wang, and J. Yang, "Deadline-aware multipath transmission for streaming blocks," in *Proc. of IEEE INFOCOM 2022*. IEEE, 2022, pp. 2178–2187.

[27] M. Li, A. Lukyanenko, Z. Ou, A. Ylä-Jääski, S. Tarkoma, M. Coudron, and S. Secci, "Multipath transmission for the internet: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2887–2925, 2016.

[28] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[29] L. Zhang, Y. Cui, J. Pan, and Y. Jiang, "Deadline-aware transmission control for real-time video streaming," in *Proc. of IEEE ICNP 2021*. IEEE, 2021, pp. 1–6.

[30] H. Shi, Y. Cui, F. Qian, and Y. Hu, "Dtp: Deadline-aware transport protocol," in *Proc. of APNet 2019*, 2019, pp. 1–7.

[31] Y. Cui, C. Ma, H. Shi, K. Zheng, and W. Wang, "Deadline-aware Transport Protocol," Internet Engineering Task Force, Internet-Draft draft-shi-quic-dtp-08, 2023, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-shi-quic-dtp/08/

[32] J. Zhang, H. Shi, Y. Cui, F. Qian, W. Wang, K. Zheng, and J. Wu, "To punctuality and beyond: Meeting application deadlines with dtp," in *Proc. of IEEE ICNP 2022*. IEEE, 2022, pp. 1–11.

[33] Q. Liu and Z. Fang, "Learning to schedule tasks with deadline and throughput constraints," in *Proc. of IEEE INFOCOM 2023*. IEEE, 2023, pp. 1–10.

[34] T. John, A. Perrig, and D. Hausheer, "Dmtp: Deadline-aware multipath transport protocol," in *2023 IFIP Networking Conference*. IEEE, 2023, pp. 1–9.

[35] L. Chuat, M. Legner, D. Basin, D. Hausheer, S. Hitz, P. Müller, and A. Perrig, *The Complete Guide to SCION*. Springer, 2022.

[36] C. Tsanikidis and J. Ghaderi, "Online scheduling and routing with end-to-end deadline constraints in multihop wireless networks," in *Proc. of MobiHoc 2022*, 2022, pp. 11–20.