Appendix

A Details of selected videos

Except for *video-bbb* and *video-sintel*, we download a set of public videos from a video website Bilibili[1] for live streaming, which are used both in Section 3 and Section 7. There are different types of content available in the video set, such as game, film, and live sport. Table 1 shows the details of these video slices. According to Section 3, the video's real-time bitrate fluctuates over time. In Section 7, we select the most classic videos and omit others since they do not offer any additional insights. All videos are chosen 60s for evaluations except *video-sintel* and *video-ss* which have 600s versions for adaptive and real-world experiments.

Table 1. Description of video set. (AvgBR., MaxBR., Rs., Fr. and Dur. are average bitrate, maximum bitrate, resolution, frame rate and video duration, respectively.)

Video set	AvgBR.	MaxBR.	Rs.	Fr.	Dur.
three	1.17 Mbps	2.61 Mbps	720p	24 fps	60s
wd	2.11 Mbps	5.53 Mbps	720p	30 fps	60s
sbd	2.68 Mbps	4.25 Mbps	720p	60 fps	60s
tears	6.13 Mbps	7.45 Mbps	1080p	24 fps	60s
sintel	9.42 Mbps	20.71 Mbps	1080p	24 fps	60s
bbb	2.78 Mbps	6.54 Mbps	1080p	30 fps	60s
ld	4.68 Mbps	8.72 Mbps	1080p	30 fps	60s
zd	3.97 Mbps	10.51 Mbps	1080p	60 fps	60s
ss	22.33 Mbps	34.41 Mbps	4K	24 fps	60s
qh	10.89 Mbps	22.37 Mbps	4K	30 fps	60s
hy	30.55 Mbps	73.34 Mbps	4K	60 fps	60s
sintel-10min	10.20 Mbps	26.72 Mbps	1080p	24 fps	600s
ss-10min	20.46 Mbps	56.04 Mbps	4K	24 fps	600s

B LinUCB algorithm

LinUCB can handle contextual multi-arm bandit (MAB) problems efficiently. In Section 4, we define our mechanism selection problem as a contextual MAB problem with *state*, *action*, and *reward*. LinUCB assumes that the expected reward $R_{t,a}$ of an action $a \in \mathcal{A}$ is linear in its d-dimensional state S_t with a coefficient vector $\hat{\boldsymbol{\theta}}_a$. The expectation of reward can be expressed as follows for each trial time t:

$$\mathbb{E}\left[R_{t,a} \mid \boldsymbol{S}_t\right] = \boldsymbol{S}_t^{\mathrm{T}} \hat{\boldsymbol{\theta}}_a. \tag{1}$$

With the utilization of ridge regression, the coefficient vector $\hat{\boldsymbol{\theta}}_a$ can be estimated as:

$$\hat{\boldsymbol{\theta}}_a = \left(\boldsymbol{D}_a^{\mathrm{T}} \boldsymbol{D}_a + \boldsymbol{I}_d\right)^{-1} \boldsymbol{D}_a^{\mathrm{T}} \boldsymbol{c}_a, \tag{2}$$

$\boldsymbol{D}_a$ here refers to the state history when acition a is selected, which is a $m \times d$ matrix, m means m contexts were observed previously for action a. $\boldsymbol{I}_d$ is the $d \times d$ identity matrix. While $\boldsymbol{c}_a \in \mathbb{R}_m$ is the corresponding reward vector.

Based on the UCB algorithm, which increases exploration chances, it computes an upper confidence bound by $\alpha\sqrt{\boldsymbol{S}_t^{\mathrm{T}} \boldsymbol{A}_a^{-1} \boldsymbol{S}_t}$, where $\boldsymbol{A}_a \overset{\text{def}}{=} \boldsymbol{D}_a^{\mathrm{T}} \boldsymbol{D}_a + \boldsymbol{I}_d$ and α here is a hyper-parameter. Instead of choosing the action that generates the highest expectation, it chooses the action that achieves the highest upper confidence bound as:

$$p_{t,a} \leftarrow \mathbb{E}\left[R_{t,a} \mid \boldsymbol{S}_t\right] + \alpha\sqrt{\boldsymbol{S}_t^{\mathrm{T}} \boldsymbol{A}_a^{-1} \boldsymbol{S}_t}, \tag{3}$$

$$a_t \leftarrow \arg\max_a \boldsymbol{p}_{t,a}. \tag{4}$$

The model training of LinUCB algorithm is shown in Algorithm 1. The input is the hyperparameter α, and the output is the coefficient $\hat{\theta}_a$ of all $a \in \mathcal{A}$.

Algorithm 1: LinUCB with disjoint linear models

Input: **Input:** α :hyperparameter
for ***all*** $a \in \mathcal{A}$ **do**
  $\boldsymbol{A}_a \leftarrow \boldsymbol{I}_d$ (d-dimensional identity matrix)
  $\boldsymbol{b}_a \leftarrow \boldsymbol{O}_{d\times 1}$ (d–dimensional zero vector)
endfor
for $t = 1, 2, 3, ..., T$ **do**
  for ***all*** $a \in \mathcal{A}$ **do**
    $\hat{\boldsymbol{\theta}}_a \leftarrow \boldsymbol{A}_a^{-1} \boldsymbol{b}_a$
    $\boldsymbol{p}_{t,a} \leftarrow \boldsymbol{S}_t^{\mathrm{T}} \hat{\boldsymbol{\theta}}_a + \alpha\sqrt{\boldsymbol{S}_t^{\mathrm{T}} \boldsymbol{A}_a^{-1} \boldsymbol{S}_t}$
  endfor
  $a_t \leftarrow \arg\max_a \boldsymbol{p}_{t,a}$
  $\boldsymbol{A}_{a_t} = \boldsymbol{A}_{a_t} + \boldsymbol{S}_t \boldsymbol{S}_t^{\mathrm{T}}$
  $\boldsymbol{b}_{a_t} = \boldsymbol{b}_{a_t} + R_{t,a_t} \boldsymbol{S}_t$
endfor

Peekaboo also uses LinUCB algorithm to decide whether to wait or transmit packets, which only solves the scheduling issue. Our work focuses on live streaming in which video characteristics are added as states and different action sets and rewards are developed.

C Optimization on start-up latency

The start-up latency is a crucial QoE metric for live streaming, which impacts whether the user continues to watch the video. At the beginning of the online learning , the framework needs time to adapt to the current network, which extends the valuable start-up latency. Consequently, we evaluate the average start-up latency of our four mechanisms and select the most suitable one to process the first video frame. As *Discard* is a

[1]www.bilibili.com, Dec.2022

frame-dropping mechanism, it is not suitable for transmitting the first video frame. As shown in Fig. 1, we find that the *Duplicate* mechanism outperforms better than others. Thus, the first video frame is loaded using the *Duplicate* mechanism.

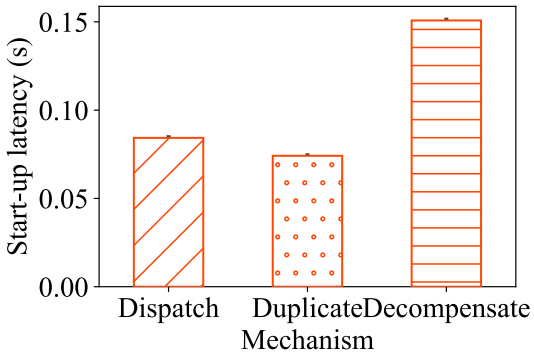

Fig.1. Optimization on start-up latency.

D Overhead analysis

Overhead from cross-layer sensing. Cross-layer sensing involves collecting information from both the network and application layers. For network layer information, three metrics, including RTT, congestion window, and inflight bytes, are maintained in the MPQUIC protocol stack, which are stored in the "sentPacketHandler" structure and provides interfaces for reading them. Bandwidth and packet loss rate estimation involve basic division operations, resulting in a time complexity of $\mathcal{O}(1)$. Regarding video characteristics, macro-information can be determined at the beginning of live streaming, which does not add any overhead. For micro-information, we have implemented an interface called "GetChunkInfo()" that obtains video frame size and type information from the RTMP message header when the message is written to the sending buffer. The real-time bitrate is computed as $\sum_{i=0}^{FR} FS_{i^*-i}$, where FR is the frame rate, i^* is the current video frame, and FS_{i^*-i} is the size of frame i^*-i. The time complexity of this computation is $\mathcal{O}(FR)$.

Overhead from online learning algorithm. We use LinUCB disjoint linear model as our online learning algorithm. The time complexity of LinUCB is discussed in [**?**]. Let d be the dimension of the feature vectors, and K be the number of arms. Then the computation complexity of LinUCB per time step is $\mathcal{O}(Kd^2 + Kd\log T)$, where T is the number of time steps. The first term Kd^2 comes from the matrix inversion operation, where LinUCB algorithm computes the inverse of the matrix $\boldsymbol{A}_{a_t}$. The second term $Kd\log T$ comes from the matrix multiplication, where the algorithm computes the product of $\boldsymbol{A}_{a_t}^{-1}$ and $\boldsymbol{S}_t$. Overall, the computation complexity of the LinUCB algorithm is dominated by the first term Kd^2, which makes it feasible to use for a large number of arms and features.

In addition, we conduct a comparison of the convergence speed of the online learning approach. We generate a dynamic network condition trace, and the achievable goodput is shown as a dashed line. Using this network trace, we measure the real-time goodput and compare whether our solution and Peekaboo, which is also an online-based multipath packet scheduler, could respond to network changes in time. The results are presented in Fig. 2, where we observe that our *4D-MAP* converges faster than Peekaboo.

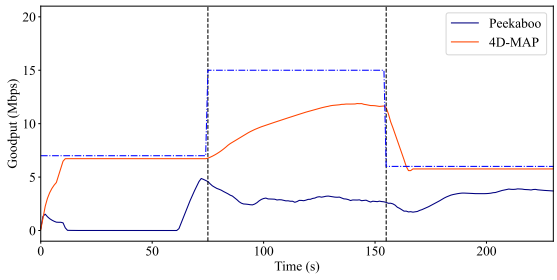

Fig.2. Convergence speed test.