Pscheduler: QoE-Enhanced MultiPath Scheduler for Video Services in Large-scale Peer-to-Peer CDNs

Dehui Wei[1 4], Jiao Zhang [1 2 *], Haozhe Li[3], Zhichen Xue[3], Yajie Peng[3],
Xiaofei Pang[3], Yuanjie Liu[1], Rui Han[3], and Jialin Li[4]

[1]State Key Laboratory of Networking and Switching Technology, BUPT, China
[2]Purple Mountain Laboratories, Nanjing, China
[3]ByteDance Ltd. Beijing
[4]School of Computing, National University of Singapore

IEEE INFOCOM 2024 - IEEE Conference on Computer Communications | 979-8-3503-8350-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/INFOCOM52122.2024.10621211

***Abstract*—Video content providers such as Douyin implement Peer-to-Peer Content Delivery Networks (PCDNs) to reduce the costs associated with Content Delivery Networks (CDNs) while still maintaining optimal user-perceived quality of experience (QoE). PCDNs rely on the remaining resources of edge devices, such as edge access devices and hosts, to store and distribute data with a Multiple-Server-to-One-Client (MS2OC) communication pattern. MS2OC parallel transmission pattern suffers from severe data out-of-order issues. However, direct applying existing schedulers designed for MPTCP to PCDN fails to meet the two goals of high aggregate bandwidth and low end-to-end delivery latency.**

To address this, we present the comprehensive detail of the Douyin self-developed PCDN video transmission system and propose the first QoE-enhanced packet-level scheduler for PCDN systems, called Pscheduler. Pscheduler estimates path quality using a congestion-control-decoupled algorithm and distributes data by the proposed *path-pick-packet* method to ensure smooth video playback. Additionally, a redundant transmission algorithm is proposed to improve the task download speed for segmented video transmission. Our large-scale online A/B tests, comprising 100,000 Douyin users that generate tens of millions of videos data, show that Pscheduler achieves an average improvement of 60% in goodput, 20% reduction in data delivery waiting time, and 30% reduction in rebuffering rate.

***Index Terms*—Peer-to-Peer CDNs, Video Transmission, Scheduler, QoE-enhanced, Large-scale A/B Test**

I. Introduction

Video applications have become a vital means of global information exchange, offering a wide range of content and effective communication capabilities. Their numerous advantages have contributed to their significance in today's digital landscape [1]. As one of the world's leading video providers, Douyin [2] serves hundreds of millions of daily users, and the annual content delivery traffic cost exceeds 600 million yuan. Therefore, saving CDN traffic costs while maintaining user QoE has become a vital issue for short video platforms [3].

PCDN is an attractive solution to reduce content delivery cost. Unlike traditional CDNs that rely on large-scale deployment of dedicated servers, PCDNs leverage underutilized resources on edge devices, such as edge access devices and hosts, to store and distribute data, resulting in a lower unit traffic price. However, these devices are more resource constrained and offer less stable performance than their CDN counterparts. To provide similar user experiences, PCDN uses multiple edge devices to serve a single user through parallel transmission. Such an architecture leads to a *Multiple-Server-to-One-Client (MS2OC) pattern*, a significant deviation from traditional multipath transmission schemes, such as MPTCP and MPQUIC, which follow a One-Server-to-One-Client (OS2OC) pattern.

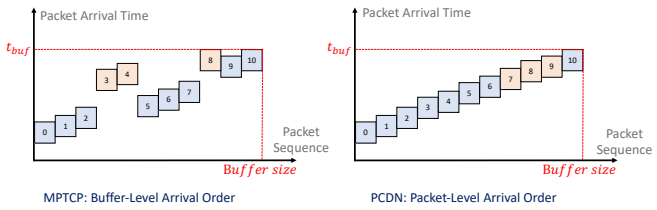


Fig. 1: Difference in order goals between MPTCP and PCDN.

The PCDN architecture brings new challenges in video content serving. Firstly, a majority (80%) of paths in PCDNs do not individually meet users' video streaming demands with satisfactory quality. PCDNs therefore require *aggregating* the bandwidth of multiple paths during a single transmission for fast video download rate. Secondly, video data received out-of-order cannot be delivered for playback, leading to rebufferring and poor video quality. PCDN needs to ensure ordered data arrival even in the presence of path diversity and a multi-source transmission scheme (MS2OC).

A naive solution is to apply existing schedulers designed for MPTCP, such as BLEST and ECF, to PCDNs. We, however, observe that they fail to meet the above PCDNs objectives due to the fundamental differences in design goals:

- **Low aggregate data download rate:** MPTCP is designed to achieve an aggregate transmission bandwidth no less than that of the fastest path (in terms of bandwidth). In the presence of path speed diversity, MPTCP suffers from Head-of-Line (HOL) blocking where a slow path prevents packet acknowledgment on the fast path. HOL can lead to buffer filling and stalling of the fast path [4]. The aggregate bandwidth is therefore limited by the

*Corresponding author: Jiao Zhang

receive rate of the slowest path R_{min}. MPTCP schedulers address this issue by applying an aggressive penalty to the slower paths when HOL occurs, suspending it from sending data and improving the rate of the faster paths. As a result, the aggregate receive rate in MPTCP typically approaches the bandwidth of the fastest path, i.e., $R_{agg} \approx R_{max}$. This, however, does not meet the goal of a PCDN, where $R_{agg} \approx \sum R$.

- **High end-to-end data delivery latency:** MPTCP schedulers optimize for buffer-level ordering, i.e., the latency of receiving all packets in the buffer from both slow and fast paths. As shown in Fig.1, such designs tolerate the unordered arrival of individual packets within the buffer. However, out-of-order packets arrival delays the average data delivery latency, which in the worst case approaches the longest Round-Trip Time (RTT) among all the paths. This is in direct conflict with PCDN's goal of low end-to-end data delivery latency.

We believe that PCDN's new deployment model and performance goals fundamentally necessitate a new transmission scheduling mechanism. However, proposing an efficient scheduler for PCDN faces many challenges. 1) MS2OC results in a *pull-based transmission pattern* where the client schedules individual video data `requests` to the set of source servers. Each data response therefore only opens a *single* flow control window in *one* source server connection. Such a flow control scheme differs significantly from MPTCP (or OS2OC in general), where one ACK can release multiple windows due to accumulated acknowledgments. It severely restricts the possible flow selection decisions the client can make. Without pausing the transmission after a data response, the scheduler will converge to a round-robin policy. 2) Existing multi-path schedulers heavily rely on path information provided by the Congestion Control (CC) algorithms. However, CCs can lead to *large cumulative errors in path condition prediction* due to phase fluctuations, particularly in the PCDN targeted unstable network environment. Real-world deployment experiences reveal that a single user requires up to 10 upload servers to fulfill its viewing demand, which further amplifies error accumulation. 3) The need for a larger number of paths per client increases the impact of *network heterogeneity*.

To address the above challenges, **we propose Pscheduler, a packet-level scheduler for PCDN systems**. Pscheduler consists of three components: a CC-decoupled path quality estimator, a Queue-Direct-Access data distribution algorithm, and a redundancy algorithm. Concretely, 1) Pscheduler takes a *"zero-dependency"* path status estimation approach, decoupling the scheduler from the congestion control algorithms. Consequently, path information can be predicted more accurately without the impact of CC iterations. 2) Pscheduler leverages the estimated path information to schedule the packet sequence numbers for each path. This *path-aware* packet scheduling maximizes the chance of in-order reception at the receiver. In particular, Pscheduler adopts a *Queue-Direct-Access* approach to retrieve packets from the global request queue, effectively addressing the dynamic network issue in PCDNs. 3) Pscheduler features a redundancy algorithm to enhance transmission reliability without wasting bandwidth. The algorithm determines if there is remaining space at the end of each short flow to accelerate the download process.

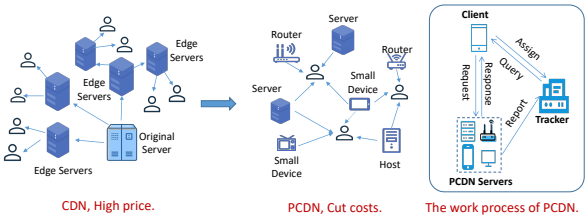


Fig. 2: The high traffic cost is driving the transition from CDN to PCDN.

We implement Pscheduler in Douyin's PCDN system and conduct large-scale online A/B testing with approximately 100,000 Douyin users whose APP updated to Pscheduler to participate in the test, generating tens of millions of video data. We observed that Pscheduler achieved an average improvement of 60% in goodput, a 20% reduction in data delivery waiting time, and a 30% reduction in rebuffering rate. Furthermore, controlled experiments show that Pscheduler outperforms other state-of-the-art schedulers, making it a better scheduling candidate for PCDN systems.

Contributions: In summary, the key contributions of this work are:

- We provide the comprehensive detail of **a new video delivery network – PCDN** and analyze the challenges it poses in multipath data packet scheduling.
- We propose the first **MS2OC scheduling mechanism for PCDN, Pscheduler**, to ensure smooth video playback.
- We **implement Pscheduler in the PCDN video system of Douyin**.
- We **conduct large-scale online tests of Pscheduler through the Douyin App**. The evaluation results with tens of millions of video data samples provide strong evidence that Pscheduler is effective in improving video QoE metrics.

In what follows, we first detail Douyin's PCDN including the architecture, the protocol, and advantages in §II. We present the key aspects of motivation in §III. We propose Pscheduler – the first MS2OC scheduling mechanism in PCDN in §IV. In §V, we implement PScheduler in Douyin's PCDN system and present the evaluation results through both simulations and practical large-scale online tests. §VI describes the related work. Finally, we draw the main conclusions in §VII.

II. Background

A. PCDN-Multipath Video Distribution Network

This section focuses on introducing the network architecture and transmission protocol of the Douyin self-developed PCDN, which has been operating on Douyin for four years.

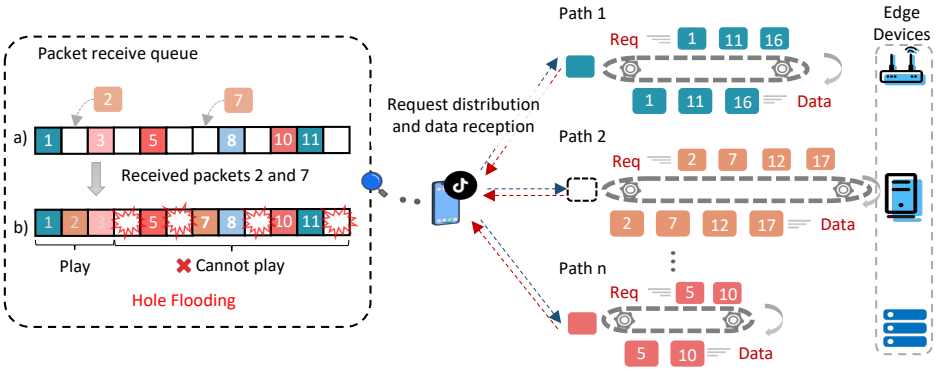


Fig. 3: Data likes rotating on conveyor belts of different lengths. Out-of-order data can cause hole flooding.

PCDN Network Architecture: To combat the high costs associated with traditional CDNs, video content providers like Douyin have turned to PCDNs in their underlying networks, as illustrated in Fig.2. CDNs rely on a tree-like interconnected server network to provide high-speed and stable bandwidth services [5]. The core idea of CDN is to load balance the original content to the user's nearest edge server to reduce data transmission delay. However, deploying CDN requires significant investment in infrastructure such as Internet data centers (IDCs) and dedicated edge servers, leading to high traffic rent fees [6]. In contrast, PCDNs utilize massive amounts of fragmented under-utilized computing storage and network resources available on the edge devices to distribute data. This approach enables the building of a low-cost, high-quality content distribution network that helps Douyin effectively manage its content delivery and maintain a competitive edge in the market.

The Douyin self-developed PCDN consists of three key components: 1) A distributed network of edge devices with under-utilized resources. 2) A client-side library, integrated in users' mobile devices. 3) A centralized tracker cluster operating on public network. Each device within PCDN serves as an uploader, caching a range of videos, typically storing entire video segments for efficiency. The system of devices is designed for simplicity: the tracker makes decisions regarding file distribution, while the client library manages data transmission. Trackers serves as a control plane in PCDN, collecting data on video popularity from client requests and operational states from devices to execute fine-grained video pre-allocation across devices. Consequently, the tracker maintains a detailed mapping table, linking each video ID to a list of devices that store the corresponding video. When the client requests a video, it first consults the tracker to locate the devices containing the video file. Following this, the client establishes connections with these devices to commence data transmission. Due to the limited and fluctuating available bandwidth of individual devices in PCDN, it adopts a multipath parallel transmission approach from multiple dispersed devices to ensure video service quality. The protocol's specifics are detailed below.

Pull-based Multipath Transmission Protocol: In PCDN,

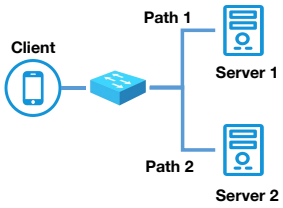


Fig. 4: Topology.

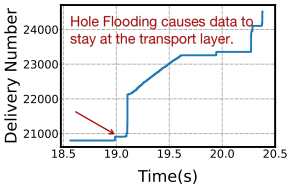


Fig. 5: The data shows a stepped delivery.

devices are invisible from each other, necessitating the client to manages the control across flows from these devices by a pull-based transmission protocol. This process involves the client establishing individual connections with a set of devices and sends `request` packets to each. Each request packet asks for a corresponding data packet, as shown on the right of the Fig. 3. In the context of video transmission, each device opens a flow control window. The client library strategically decides which request to send through which connection, based on the availability of free windows for effective flow control and scheduling. Upon receiving a request, a device responds with the requested data packet. If these packets arrive in the correct sequence, they are promptly forwarded to the player for decoding and playback. As a result, the client side needs to schedule data packets across different connections to ensure that the data arrives in the correct order, preserving the integrity and continuity of the video stream. Based on real deployment experience, a user's viewing demand typically requires support from 2 to 10 devices, as 80% of the devices have limited individual remaining resources, but the combined amount is substantial. This inevitably increases the heterogeneity among paths, posing a greater difficulty in designing the scheduling mechanism [7]. To facilitate updates, the entire transmission protocol is implemented in user space based on UDP.

B. Typical Multipath Scheduling Mechanisms

Douyin deployed the default scheduler in the Linux system, minRTT, that distributes packets to the path with the smallest RTT, which cannot solve the above problems. Some advanced multipath scheduling mechanisms have been proposed in recent years, such as BLEST [8], ECF [9], XLINK [10]. BLEST

and ECF will pause the slow path packet transmission when they predict that a HOL blocking will occur. XLINK uses the QoE feedback to calculate the remaining playback time and combines the data packet transmission time for re-injection operation, which reduces the request completion time and video rebuffer rate. They both sacrifice aggregate bandwidth to achieve improvements in other metrics.

III. Motivation

In this section, we discuss the limitations of existing multipath schedulers when applied to a PCDN deployment.

A. Aggregate Bandwidth and End-to-End Latency Trade-Offs

MPTCP schedulers fail to guarantee high throughput while minimizing end-to-end latency required by PCDN systems. MPTCP, similar to TCP, provides in-order and reliable data delivery to applications, which means that the receive buffer must temporarily store out-of-order data packets. Let's consider a scenario with one fast path and one slow path, where their respective bandwidths and RTTs are denoted as B_f and B_s, RTT_f and RTT_s. To avoid sacrificing aggregate throughput, the default minimum scheduler buffer size is [11]:

$$Buf_{agg} = (B_f + B_s) \times RTT_s \tag{1}$$

At this point, even though the fast path is not blocked for sending, it needs to wait for the arrival of packets from the slow path before delivery. During this period, the number of window rounds N that the fast path has to wait for is as shown in Eq.(2). The end-to-end delivery latency of packets in each round decreases by RTT_f compared to the previous round. So the sum of end-to-end delivery latency is computed as Eq.(3). Therefore, the average end-to-end delivery latency at this buffer size is given by Eq.(4). It can be observed that when RTT_f, B_f and B_s remain constant, the larger RTT_s, the higher the average end-to-end delivery latency.

$$N_{round} = RTT_s / RTT_f \tag{2}$$

$$\begin{aligned} L_{sum} = {} & RTT_s \times CWND_s + \\ & \sum_{i=0}^{N-1} (RTT_s - i \times RTT_f)) \times CWND_f \end{aligned} \tag{3}$$

$$\begin{aligned} L_{avg} = {} & L_{sum} / Buf_{agg} \\ = {} & RTT_s \times (1 - \frac{1}{2 \times RTT_f \times (B_f + B_s)}) \\ & + \frac{1}{2 \times (B_f + B_s)} \end{aligned} \tag{4}$$

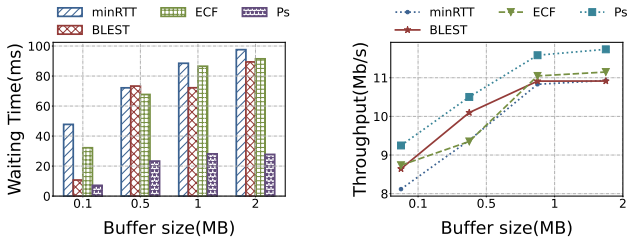
Then how about decreasing the buffer size to less than Buf_{agg}? If so, MPTCP schedulers cannot fully utilize the aggregate bandwidth. They will tend to block the slow path to fully utilize the fast path. At this point, the minimum achievable end-to-end delivery latency is RTT_f, but the aggregate bandwidth is also approximately equal to B_f [12]. Therefore, *MPTCP schedulers can only achieve a trade-off between high aggregate bandwidth and low end-to-end latency, but cannot simultaneously achieve both.*

We conducted experiments to demonstrate this phenomenon. Specifically, we replicated the representative BLEST and ECF mechanisms in mininet [13] and integrated the PCDN transmission protocol into the mininet. The experimental topology, as shown in Fig.4, consists of a client connected to two servers via a router, with a bottleneck having a bandwidth of 30Mbps and a loss rate of 0.1%. The RTTs of Path 1 and Path 2 are 30ms and 90ms, respectively. In Fig.6, we present the results showing the average waiting time between a packet received and delivered per packet and throughput for different mechanisms with varying receiving buffer sizes. Notably, we observed a direct proportionality between the waiting time and the buffer size, while the throughput displayed an inverse proportionality to it. Indeed, Pscheduler (Ps) can perform well in both aspects. It effectively manages waiting time and optimizes throughput, achieving a good balance between the two objectives.

B. Challenges in Aggregating Bandwidth Using Complete CWND

To avoid wasting any bandwidth, it is imperative to leverage the complete CWND without impeding the progress of slower paths. However, achieving this objective poses a notable challenge—maintaining the sequential arrival of packets when utilizing the entire CWND. In PCDN, the client adopts a request-reply pull-based transmission mode that each data response only opens a single flow control window in one source server connection. When a user downloads data from different devices, data is like rotating on conveyor belts (paths) of different lengths as shown in Fig.3. The longer the conveyor belt, the longer the path RTT. When the transmission is in a steady state, the data rotation speed is equal to the path bandwidth. For the path i, a data packet is received at each intervals of $\frac{packet\_size}{bandwidth_i}$. As the packet size is fixed, the arrival times of data packets will vary for each path if the path bandwidths are different. With the PCDN transmission feature, the client observes only one available transmission window across all paths each time it receives a data packet. Schedulers are triggered after receiving the packet, determining the path for sending subsequent packets. Without blocking the transmission, the next packet is sent to the path with an available window. Consequently, the client will distribute requests to different paths almost randomly, causing data request packets to be sent in order and data packets to be received out-of-order, can't be delivered.

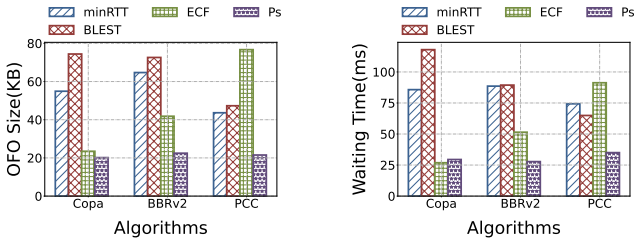
For example, taking Fig.3 left, at a certain moment, the receive queue is queue a). At this time, packets 2 and 7 arrive in the queue, which can only deliver ordered data packets 1∼3 to the player. Packets after 3 are unordered and incomplete, and cannot be played even if packet 7 arrives because there are holes between packets 3 and 7. With the increase in the number of paths, the heterogeneity among paths grows, and the correlation between packets arriving becomes smaller. The presence of holes between packets also becomes more frequent, which is referred to as "hole flooding". Out-of-order data in receive buffer of transport layer cannot be delivered to the upper layer, increasing end-to-end latency and jitter, and video playback may rebuffer (§V). Fig. 5 records the



(a) Waiting time between a packet received and delivered.

(b) Aggregate bandwidth.

Fig. 6: Trade-off between aggregate bandwidth and latency of different schedulers under different receive buffer sizes.



(a) The size of out-of-order data.

(b) Waiting time between a packet received and delivered.

Fig. 7: The performances of different schedulers under different congestion control algorithms.

delivery of data over a period of time without Pscheduler in PCDN, highlighting the ladder-like delivery pattern caused by the hole flooding problem. In summary, to ensure smooth video playback under PCDN, the scheduler must effectively manage strict packet-level arrival ordering.

C. Coupled with CC Algorithms Cause Imprecise Path Information.

To achieve strict packet arrival ordering, schedulers need accurate prediction of the packet arrival time on different paths. However, existing multipath schedulers rely heavily on CC algorithms to estimate the transmission time of each path and often perform very differently under different CC algorithms. To investigate this issue, we conducted experiments in the same environment as before, with a receiving buffer size of 0.5KB. And we choose to use some representative CC algorithms like Copa [14], BBRv2 [15], and PCC [16], to transmit files. The results are shown in Fig.7. We take BLEST using the BBRv2 [17] algorithm as an example. During probing, BBRv2 deliberately increases the congestion window (CWND), which can mislead the scheduling algorithm into believing that the path is improving, resulting in more packets being sent on the slow path. Conversely, during the emptying phase, BBRv2 intentionally reduces the CWND, causing the scheduling algorithm to send fewer packets on the slow path. Different CC algorithms have varying behaviors in network bandwidth detection and emptying, causing the scheduler performance to fluctuate. Such a scheduling mechanism is not CC-agnostic. When the transport protocol migrates to the user state, the CC algorithm can evolve quickly. Continually adapting the scheduling algorithm with different CCs would be labor-intensive and affect the advantages of the scheduling mechanism [18], [19].

IV. Design

The design of Pscheduler is centered around ensuring smooth video playback and cost-saving under PCDN. To meet these requirements, Pscheduler needs to achieve strict packet-level ordering while efficiently utilizing the throughput of all available paths.

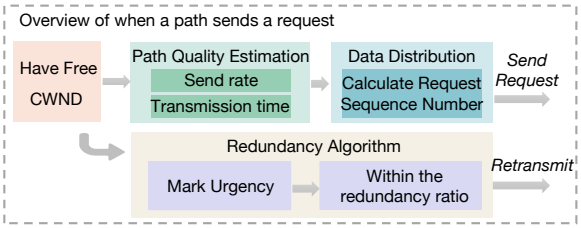


Fig. 8: The Overview of Pscheduler.

A. Framework

As illustrated in Fig. 8, Pscheduler comprises three key components:

1) **Path Quality Estimation (§IV-B)** that is independent of CC algorithms to assess the quality of paths.

2) **Data Distribution (§IV-C)** to guarantee sequential data reception and adapt to dynamic networks.

3) **Redundancy Transmission (§IV-D)** algorithm for smooth video playback, which will be activated in exceptional situations to enhance user QoE.

Pscheduler calculates the packet sequence number that should be sent for the current path based on accurate path estimation information. Then, it adopts the "Queue-Direct-Access" approach to retrieve packets from the overall request queue and send them out. Video transmission is always segmented to avoid waste, and each segment is downloaded as a range. The redundancy algorithm determines if there is remaining space at the end of each range to accelerate the download process.

B. CC-Decoupled Path Quality Estimation

To achieve packet-level arrival ordering in packets distribution of Pscheduler, two variables of the network transmission paths are required to be obtained: the transmission rate and the end-to-end transmission time of sending a data packet along different paths.

Transmission Rate Estimation. Previous mechanisms calculate the transmission rate by dividing CWND by RTT, which is obtained from CC algorithm decisions. However, the CC algorithms are often in an exploration state and may not accurately represent the actual transmission rate (§III-C). To decouple from CC algorithms, Pscheduler employs a transmis-

sion rate calculation method that is independent of the CWND, as shown in Eq.(5):

$$R = \frac{datasize}{endtime - starttime} \tag{5}$$

Where $datasize$ stands for the size of the received data, $starttime$ represents the time when the first packet is received in each interval, and $endtime$ is the time when the last packet is received in each interval. We calculate the rate at intervals of every received CWND packet, although CWND is not a parameter in the rate calculation equation. This approach avoids the impact of transient behavior in CC algorithms, ensuring a more precise estimation of the connection's transmission rate.

Transmission Time Estimation. Pscheduler avoids directly using RTT to estimate the transmission time of packets. Instead, it calculates the average transmission time required by a path to successfully complete a data packet transmission as Eq.(6), including the parameter of packet loss ratio ($loss$). This approach prevents the unnecessary sending of packets on the paths with small RTT but high packet loss. The probability that a packet is successfully transmitted after being lost $n-1$ times is $loss^{n-1} \times (1 - loss)$, and the transmission time is $((n-1) \times rto + rtt)$, $n \in N+$. Thus, the average transmission time of the data packet is:

$$\begin{aligned} E &= \sum_{n=1}^{\infty} (loss^{n-1} \times (1 - loss) \times ((n-1) \times rto + rtt))) \\ &= rtt + rto \times \frac{loss}{1 - loss} \end{aligned} \tag{6}$$

Where rto is the timeout retransmission time and $loss$ stands for the current packet loss rate. We use TFRC [20] to calculate the *loss* rate, store flight data packets in a queue, and use this queue to calculate the current packet loss rate.

In addition, due to the pull-based transmission mode of PCDN, both the sending of data request packets and receiving of data packets are on the client side. The transmission time of a data packet will change from OWD (one-way delay, the server sends data to the client) to RTT (round-trip delay, client requests data and the server replies), which means that the clock synchronization problem between the sender and receiver will be eliminated in the path quality estimation of Pscheduler. Therefore, using a transmission time-based scheduling algorithm in PCDN will be more accurate.

C. Queue-Direct-Access Data Distribution

To achieve packet-level data scheduling instead of connection-level scheduling, Pscheduler requires greater flexibility. We consider two data distribution methods: "packets pick paths" and "paths pick packets". Ultimately, to enable the mechanism to flexibly combat network fluctuations and achieve strict arrival ordering, we choose the "paths pick packets" method.

Packets distribution based on sub-buffers (Packets pick Paths): To efficiently schedule request packets on demand, one approach is to employ individual pending task queue buffers for each path, as depicted in Fig.9a). Request packets are pre-allocated to the sub-buffer to rearrange their sending order and achieve the desired arrival order of data packets. However, this

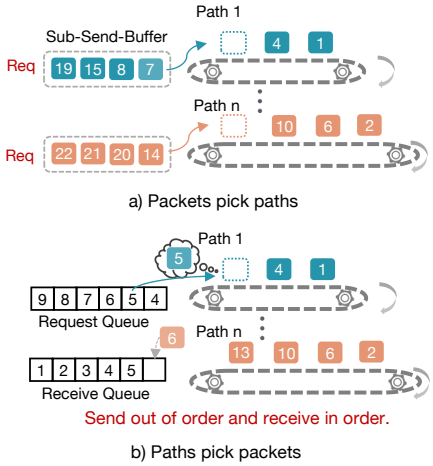


Fig. 9: Two packet distribution modes.

method lacks flexibility when the network experiences fluctuations, as the request packets in the sub-buffer may be out of order based on their pre-allocated sequence. Consequently, the pre-allocated packets cannot achieve strict packet-level arrival ordering, leading to a loss of flexibility.

Queue-Direct-Access packets distribution (Paths pick Packets): Pscheduler introduces a scheduling mode, *path-pick-packet* (Fig.9b), to enhance the flexibility of request packet transmission order. Unlike pre-allocation, this method dynamically selects request packets based on network quality, directly choosing packets from the overall queue for available path space. This process is akin to a rotating conveyor belt, where the queue learns which request packet to pick after each delivery, avoiding the need for a worker to pre-move packets in a fixed order. Despite only deciding one path's window per slot, Pscheduler accurately selects the specific packet, achieving strict packet-level arrival order.

Algorithm1 is the detailed design of the Queue-Direct-Access Packets Distribution algorithm. Whenever a path i has an available window, Pscheduler calculates the average transmission time E_i of the path according to Eq.(6) (line 3). Then traverse all connections and calculate their average transmission time E_j (lines 4-5). The smaller E means that the connection can complete the transmission of a data packet faster. Suppose the E_f of the first path is T, and the throughput is D; the E_s of the second path is $2T$, and the size of the data packet is S, then the arrival time of the data packet is $sendtime + E$. In $(E_s - E_f)$ time, the first path can send $\frac{T*D}{S}$ packets. The second path directly sends the packet whose position is $\frac{T*D}{S} + 1$ in the task queue which can ensure that the data arrives in an orderly manner. At this time, the arrival time of the packets at positions $\frac{T*D}{S}$ and $\frac{T*D}{S} + 1$ in the task queue is both $curtime + 2*T$. Therefore, Pscheduler calculates offset according to Eq.(7) (lines 7-8). Then the second path sends the packet whose pos is $Offset$ in the total task queue. It can be observed that throughout the entire calculation process, the buffer size variable is not utilized. Therefore, Pscheduler is buffer-size agnostic and remains unaffected by buffer inflation.

$$Offset = \sum_{j \neq i}^{n} \frac{(E_i - E_j) \times R_j}{packetsize} (E_i > E_j) \tag{7}$$

Algorithm 1: Packet Distribution Algorithm

Input: connection i, connection set C
Output: $offset$

```
begin
    offset = 0
    calculate average packet transmission time of
      connection i E_i via Eq.(6)
    for j ∈ C and j ≠ i do
        E_j = rtt_j + rto_j × loss_j / (1−loss_j)
        if E_j < E_i then
            calculate R_j via Eq.(5)
            Offset+ = (E_i−E_j)×R_j / packetsize
    return Offset
```

D. Redundancy Transmission

Pscheduler features a distinct redundancy algorithm, strategically triggered to enhance transmission reliability that doesn't waste the bandwidth. It considers data packet urgency, redundancy percentage, and task tail redundancy. In particular, tail redundancy is important as sporadic packets at the end of each range can limit the transmission of the next task segment.

Mark Packets Urgency. If the packet cannot arrive in time for its playback deadline, it will be marked as urgent, calculated as Eq.(8). Where $playtime_{remain}$ is the remaining time until the certain packet is played, and $sendtime$ is the time required to transmit it estimated as Eq.(6). The $threshold$ is the dangerous water level of the playback buffer. Once the remaining playback time is lower than the $threshold$, it is very likely to cause rebuffering. Considering that packets may be lost, Pscheduler adds a maximum path RTT above this threshold, so that even if packets are lost, there is a chance for retransmission.

$$playtime_{remain} - sendtime < threshold + RTT_{max} \quad (8)$$

Percentage of Redundancy. The second step is to ensure that redundant packets will not reduce user QoE by limiting the percentage between the number of redundant packets and the task download speed. We set a cap on the percentage of redundancy used. This approach guarantees that redundancy will not be activated when the bandwidth is insufficient to sustain the client's throughput requirements. Additionally, it ensures that the download speed of non-redundant data remains equal to or greater than the playback speed. Assuming the client's playback bitrate is F, the average download speed is $V_{arg} = \sum_i^n R_i$, $redundant\_data\_size$ the size of redundant data currently being transferred, $flight\_data\_size$ is the total data size being transmitted in the current network. Redundancy can be triggered only when $F < V_{arg}$, and the redundant data size should meet the following conditions:

$$\frac{redundant\_data\_size}{flight\_data\_size} \leq \frac{V_{arg} - F}{V_{arg}} \quad (9)$$

By satisfying these two criteria, a packet will only activate the redundancy algorithm if it is lost multiple times and the buffer data is depleted, striking a balance between redundancy and performance. Each redundant packet is vital in ensuring the minimum requirement, significantly decreasing the risk of stuttering.

Task Tail Redundancy. For segmented task transmission, Pscheduler designed a tail redundancy method. At the end of each range, there will be a situation where the number of remaining data packets is less than the total CWND. If the tail data packet is lost or does not reach the receiving end as expected due to network fluctuations, the link will appear in an idle waiting phase. In addition, if the previous task is not completed, the next range cannot transfer. But it is surprising that no existing works have addressed this critical problem. Therefore, once a connection i is detected to $offset > task\_size$, and $offset < 2 * task\_size$, which to used for redundant transmission. Each tail packet will only be redundant once.

V. Implementation and Evaluation

A. Implementation

We implement Pscheduler in the PCDN of Douyin. The scheduler will be triggered when the client needs to send a new "data request packet" by DataTimer(), Addnewtask(), or receiving a packet. DataTimer() is triggered every 100ms and will query the tracker for available nodes. Addnewtask() is called every time a new task is created. When the scheduling algorithm is triggered, the multipath scheduler will allocate request packets on the connections that have idle CWND and are still in the connection state.

B. Evaluation Setup

We verify the performance of Pscheduler by large-scale A/B tests from Douyin real users as well as simulations.

Online Large-scale A/B test: We select approximately 100,000 Douyin users to participate in the Pscheduler experiment. Our analysis cover over 10 million video plays per day. To conduct this large-scale online test, we employ the A/B testing methodology, wherein two user groups simultaneously used Pscheduler and basic minRTT for one months. Our A/B tests show the benefits of Pscheduler in terms of both continuous and discontinuous speed improvements, as well as the reduction of data packet out-of-order, which directly impacts QoE. From a QoE perspective, we focus on reducing rebuffering rates and improving the upward delivery speed of data. Additionally, we also pay attention to the wastage rate of the mechanism. To test the impact of redundancy algorithms on the results, we evaluated two variants of Pscheduler: Pscheduler (the full version of Pscheduler) and Pscheduler-NR (Pscheduler without redundancy).

Simulation Evaluation: We integrate PCDN's transmission protocol into Mininet. Additionally, we incorporate other MPTCP schedulers, such as ECF and BLEST, into the simulator. These schedulers are implemented in PCDN based on their algorithm descriptions outlined in papers. The utilization of Mininet allows us to establish diverse network scenarios, varying path numbers, and introducing heterogeneous RTT. This setup facilitates a comparative analysis of different schedulers

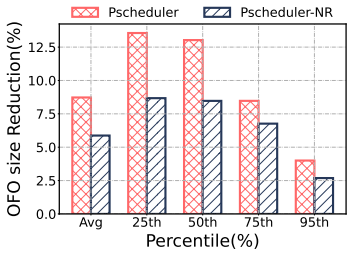


Fig. 10: [A/B test] The reduction in out-of-order data average size.

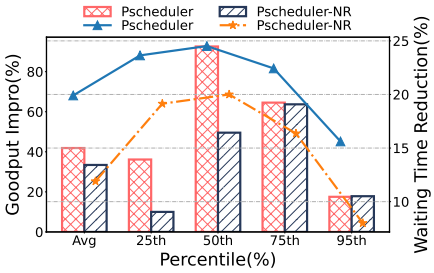


Fig. 11: [A/B test] Goodput vs. Data waiting time.

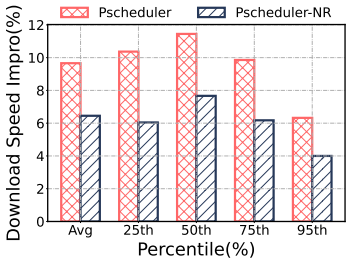


Fig. 12: [A/B test] The improvement in video download speed.

under varying conditions. Unless otherwise specified, the CC algorithm defaults to BBR.

C. Large-scale A/B Test

Out-of-order Data Size: The metric of out-of-order data size is the most straightforward indicator of the impact of scheduling mechanisms. After receiving each packet, we measured the current size of out-of-order data and calculated the average. Fig.10 demonstrates that Pscheduler effectively reduces the average amount of out-of-order data by approximately 8% compared to the baseline mechanism.

Goodput and Data Waiting Time: Fig.11 presents the goodput improvement and data waiting time reduction achieved by Pscheduler at different percentiles compared to the minRTT mechanism. Goodput is defined as the speed of data delivered for playback. In other words, it is calculated as the data in order divided by the duration between the beginning of range downloading and receiving half the size of the range. The data waiting time metric measures the time between a packet being received and its delivery upward for playback. Goodput and data waiting time are interrelated. If data remains in the buffer for a prolonged period but cannot be played due to out-of-order issues, the goodput will decrease, increasing the likelihood of buffering and stuttering occurrences. Pscheduler efficiently arranges data in order, resulting in a 20%∼80% increase in goodput and 15%∼25% reduction in waiting time, which effectively prevents rebuffering.

Video Download Speed: We measure video download speeds for each range being downloaded. Fig.12 shows the speed improvement of Pscheduler and Pscheduler-NR compared to the original algorithm (minRTT) at different percentiles. Pscheduler-NR shows an average improvement of 6% in video download speed, while Pscheduler achieves an average improvement of 10%. At the 95th percentile, Pscheduler still exhibits a 6% enhancement. As video downloads are often segmented based on ranges, with short flows being predominant, the selection of data packet transmission paths becomes crucial, making Pscheduler effective in boosting video download speeds. However, compared to continuous video download speeds (Goodput), the improvement in download speeds may not appear as significant. This is because the primary role of the scheduler is to enhance data ordering, which has already been demonstrated through the increase in

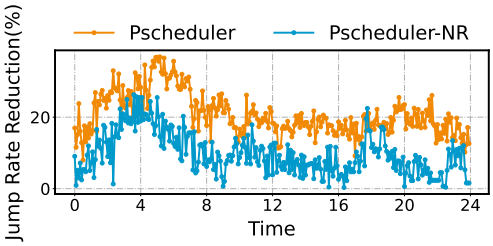
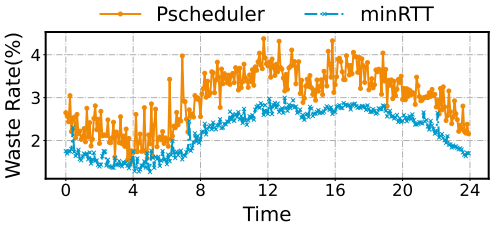


Fig. 13: [A/B test] The reduction in the rate of videos that jump out to the CDN network due to rebuffering.



Waste(%) Hour / Alg	[0, 4)	[4, 8)	[8, 12)	[12, 16)	[16, 20)	[20, 24)
Ps	2.17	2.27	3.39	3.44	3.46	2.79
$minRTT$	1.54	1.61	2.43	2.65	2.66	2.14

Fig. 14: [A/B test] The waste rate of Pscheduler and minRTT within a day.

continuous download speed and the reduction in rebuffering rates, validating the effectiveness of the algorithm.

Video Jump Rate: The rebuffer rate is a crucial metric influencing video Quality of Experience (QoE) and video smoothness. In Douyin's PCDN, rebuffering triggers a switch from PCDN to CDN as a backup. We use the online jump rate as a measure of PCDN transmission failures leading to CDN transition, calculated by dividing jump duration by total playback time. Fig.13 displays the average jump rate throughout the day, comparing Pscheduler to the minRTT mechanism. Pscheduler effectively reduces the jump rate by up to 30%.

Data Waste Rate: In Fig.14, the data waste rate for a specific day is presented. This rate is calculated by taking the difference between the number of requests sent and the total number of packets in a task, divided by the total number of packets in the task. The waste rate accounts for both packet

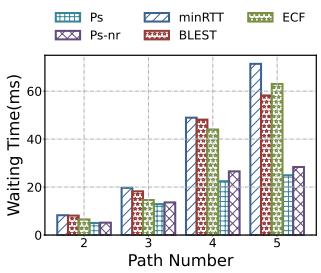


Fig. 15: [Simu] Waiting time before a packet delivered.

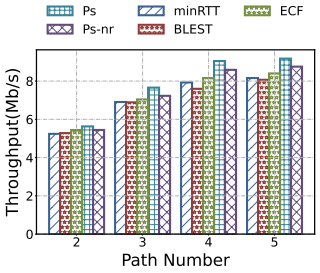


Fig. 16: [Simu] The speed of data delivered for playback.

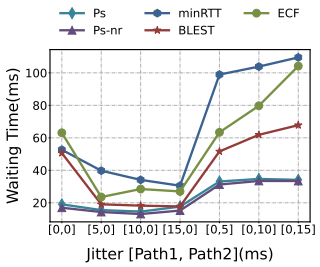


Fig. 17: [Simu] Waiting time before a packet delivered.

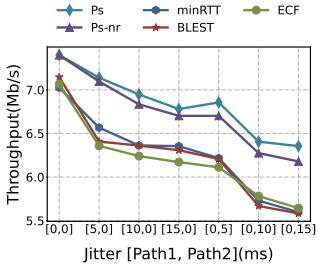


Fig. 18: [Simu] The speed of data delivered for playback.

loss and redundancy. Pscheduler exhibits a slightly higher waste rate than minRTT due to its redundancy algorithm. However, the increase is only 0.74%, while achieving a 10% improvement in video download speed. This suggests that Pscheduler's redundancy strategy employs a minimal amount of redundant data but plays a crucial role.

D. Simulation Evaluation

To compare Pscheduler with other schedulers, we conduct controlled experiments in a mininet simulation environment.

Different Path Number Scenario: A client downloads video content from a varying number of devices. The minimum RTT of paths is set to 30ms, and with each additional path, the RTT increases by 10ms. Each path is considered a bottleneck with a bandwidth of 3Mbps, and the buffer size is sufficiently large. In each scenario, every scheduler is tested through 10 experiments, and the average results are calculated. Fig.15 and Fig.16 display the outcomes in terms of throughput and waiting time. Remarkably, these figures show similar trends to the results obtained from online experiments. Pscheduler demonstrates a superior ability to enhance aggregate throughput and reduce waiting time compared to other schedulers.

Jittered RTT Scenario: Fig.17 and 18 present the simulation results. The x-axis represents the upper bound of RTT jitter for two paths, while the bandwidth for both paths is set at 5Mbps. It is evident that even in scenarios with significant jitter, Pscheduler outperforms other mechanisms in terms of performance.

VI. Related Work

Multipath Transmission Protocols: Multipath transmission protocols have evolved as the number of device network interfaces increases [21]–[23]. MPTCP [21] and MPQUIC [24] respectively expand the single-path transmission protocol based on TCP [25], UDP to an end-to-end multipath transmission protocol to increase the transmission rate and robustness. PCDN is a multiple-server-to-one-client transmission protocol that makes up for the lack of single-path transmission performance.

Multipath Scheduling Algorithms: Several MPTCP scheduling algorithms exist, including RoundRobin [26], which traverses all subflows and sends data as long as the subflow has a free CWND and the default minRTT algorithm in the Linux kernel. Other algorithms like Otias [27], ECF [9], and BLEST [8] prioritize fast paths, potentially causing the slow path to idle and reducing overall throughput. Redundancy-based approaches like TWC [28] and XLINK [10] have been explored to ensure orderly data transmission, which also compromise the aggregate throughput. These mechanisms are mostly designed for MPTCP and MPQUIC scenarios and their focus is primarily on solving the HOL issue.

Network Architecture: CDN, content distribution network, caches video data on servers closer to users. With the increase in bandwidth demand, the construction cost of servers, switches, etc. has also risen sharply. P2P [29], [30] is a peer-to-peer transmission network architecture, which can be divided into tree-based, grid-based, and hybrid tree-grid-based systems. Although P2P networks have higher scalability and lower deployment costs, dynamic nodes also cause instability and low performance due to insufficient participating nodes. PCDN combines the two to build a high-quality and low-cost video distribution network.

VII. Conclusion

With the increasing costs of traffic, the demand for more cost-efficient video distribution has led to the transition from traditional CCDNs to PCDNs. In this paper, we present Pscheduler, a novel MS2OC scheduling mechanism tailored specifically for PCDNs. Pscheduler adopts a fine-grained packet-level scheduling approach, with the primary goal of optimizing the aggregate bandwidth while simultaneously minimizing end-to-end latency, thus enhancing the overall video QoE. Through extensive large-scale online A/B tests conducted on the popular platform Douyin, our results demonstrate that Pscheduler effectively reduces data disorder in the MS2OC scenario and significantly improves video goodput.

Acknowledgments

This work is partly supported by the National Key RD Program of China under Grant No. 2022YFB2901700, the National Natural Science Foundation of China (NSFC) under Grant No. 62132022 and 62372053, Fok Ying Tung Education Foundation under Grant No. 171059, and BUPT Excellent Ph.D. Students Foundation (No. CX2021102).

REFERENCES

[1] U. Cisco, "Cisco annual internet report (2018–2023) white paper," *Cisco: San Jose, CA, USA*, 2020.

[2] "Douyin web version." https://www.douyin.com/.

[3] S. Nacakli and A. M. Tekalp, "Controlling p2p-cdn live streaming services at sdn-enabled multi-access edge datacenters," *IEEE Transactions on Multimedia*, 2020.

[4] M. Scharf and S. Kiesel, "Nxg03-5: Head-of-line blocking in tcp and sctp: Analysis and measurements," 2006.

[5] T.-M. Pham, M. Minoux, S. Fdida, and M. Pilarski, "Optimization of content caching in content-centric network," 2017.

[6] F. Chen, K. Guo, J. Lin, and T. La Porta, "Intra-cloud lightning: Building cdns in the cloud," in *Proceedings IEEE INFOCOM*, 2012.

[7] G. Zhang, W. Liu, X. Hei, and W. Cheng, "Unreeling xunlei kankan: Understanding hybrid cdn-p2p video-on-demand streaming," *IEEE Transactions on Multimedia*, 2014.

[8] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks," in *IFIP networking conference and workshops*, 2016.

[9] Y.-s. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "Ecf: An mptcp path scheduler to manage heterogeneous paths," in *Proceedings of the international conference on emerging networking experiments and technologies*, 2017.

[10] Z. Zheng, Y. Ma, Y. Liu, F. Yang, Z. Li, Y. Zhang, J. Zhang, W. Shi, W. Chen, D. Li *et al.*, "Xlink: Qoe-driven multi-path quic transport in large-scale video services," in *Proceedings of the SIGCOMM*, 2021.

[11] 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 *NSDI*, 2012.

[12] H. Shi, Y. Cui, X. Wang, Y. Hu, M. Dai, F. Wang, and K. Zheng, "STMS: Improving MPTCP throughput under heterogeneous networks," in *ATC*, 2018.

[13] "mininet." http://mininet.org/.

[14] V. Arun and H. Balakrishnan, "Copa: Practical delay-based congestion control for the internet," in *NSDI*, 2018.

[15] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: congestion-based congestion control," *Communications of the ACM*, 2017.

[16] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "Pcc: Re-architecting congestion control for consistent high performance," in *NSDI*, 2015.

[17] N. Cardwell, Y. Cheng, S. H. Yeganeh, P. Jha, Y. Seung, K. Yang, I. Swett, V. Vasiliev, B. Wu, L. Hsiao *et al.*, "Bbrv2: A model-based congestion control performance optimization," in *Proceedings of IETF Meeting*, 2019.

[18] A. Nikravesh, Y. Guo, X. Zhu, F. Qian, and Z. M. Mao, "Mp-h2: a client-only multipath solution for http/2," in *Annual International Conference on Mobile Computing and Networking*, 2019.

[19] Q. De Coninck and O. Bonaventure, "Observing network handovers with multipath tcp," in *Proceedings of the SIGCOMM Conference on Posters and Demos*, 2018.

[20] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "Tcp friendly rate control (tfrc): Protocol specification," Tech. Rep., 2003.

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

[22] Q. Coninck and O. Bonaventure, "Multipath extensions for quic (mp-quic)," *IETF, Individual Submission, Internet Draft draftdeconinck-quic-multipath-04*, 2020.

[23] Y. E. Guo, A. Nikravesh, Z. M. Mao, F. Qian, and S. Sen, "Accelerating multipath transport through balanced subflow completion," in *Proceedings of the Annual International Conference on Mobile Computing and Networking*, 2017.

[24] Q. De Coninck and O. Bonaventure, "Multipath quic: Design and evaluation," in *Proceedings of the international conference on emerging networking experiments and technologies*, 2017.

[25] B. A. Forouzan, *TCP/IP protocol suite*, 2002.

[26] M. F. Imaduddin, A. G. Putrada, and S. A. Karimah, "Multipath tcp scheduling performance analysis and congestion control on video streaming on the mptcp network," in *International Conference on Software Engineering & Computer Systems and International Conference on Computational Science and Information Management*, 2021.

[27] F. Yang, Q. Wang, and P. D. Amer, "Out-of-order transmission for in-order arrival scheduling for multipath tcp," in *International conference on advanced information networking and applications workshops*, 2014.

[28] Y. Xing, K. Xue, Y. Zhang, J. Han, J. Li, J. Liu, and R. Li, "A low-latency mptcp scheduler for live video streaming in mobile networks," *IEEE Transactions on Wireless Communications*, 2021.

[29] Y. Huang, T. Z. Fu, D.-M. Chiu, J. C. Lui, and C. Huang, "Challenges, design and analysis of a large-scale p2p-vod system," *SIGCOMM computer communication review*, 2008.

[30] J. Lei, L. Shi, and X. Fu, "An experimental analysis of joost peer-to-peer vod service," *Peer-to-peer networking and applications*, 2010.