Multipath Parallel Reverse Segment Download (MPRD) for Peer-to-Peer Content Delivery

Suresh Srinivasan and Ehsan Aryafar
Department of Computer Science, Portland State University, Portland, OR, USA

***Abstract*—Internet content providers (e.g., video streaming, web services, file downloads) are increasingly deploying Peer-to-Peer Content Delivery Networks (PCDNs) to cache and distribute content by leveraging spare compute and storage resources at the network edge. This shift is primarily driven by the high costs associated with building and maintaining large-scale, centralized Content Delivery Network (CDN) architectures. A PCDN architecture can use multiple edge devices (e.g., access points, smart home gadgets) to distribute data in parallel to each client to increase the communication bandwidth and robustness to link/server failures. However, existing distributed multipath transport schedulers face significant challenges, such as Head-of-Line (HoL) blocking and Out-Of-Order (OOO) packet delivery, which can degrade multipath throughput to levels even lower than using a single path alone. In this paper, we introduce MPRD (Multipath Parallel Reverse Segment Download), a novel receiver-driven distributed transport protocol designed to overcome these limitations in PCDN architectures. MPRD is built on top of TCP and rethinks traditional multipath transport by using single-path TCP at each edge server. In MPRD, the client optimizes the starting point and direction of file segments to be downloaded by each single-path TCP connection in order to eliminate the inefficiencies of existing multipath schedulers. We implemented a prototype of MPRD in the Linux userspace and demonstrate that it significantly improves client throughput and reduces the CDN load (when the CDN is augmented by PCDN edge servers). We also evaluate MPRD's performance in improving video delivery quality and show that it can substantially reduce the stalls and increase the PSNR visual quality by up to 35 dB.**

***Index Terms*—Multipath Transport Protocol, Edge Computing, Distributed Parallel Data Download, Head-of-Line Blocking**

I. Introduction

Over the past few years, the rapid proliferation of Internet-connected devices and the surge in data traffic demand have posed significant challenges to traditional centralized content delivery architectures, including reduced reliability [1] and high costs [2]. To address these issues, Peer-to-Peer Content Delivery Networks (PCDNs) [3] based on decentralized architectures have emerged, enabling Internet-connected devices to share data directly with one another.

Decentralized data sharing has roots in early technologies such as USENET [4], which was used for news dissemination, and later evolved to support applications like music sharing (e.g., Napster [5]) and file distribution (e.g., BitTorrent [6]). In recent years, the growing demand for video-based applications—particularly short-lived videos—and the widespread adoption of mobile devices have prompted video content providers, such as ByteDance [7], to adopt decentralized PCDN architectures [3], [8], [9] to enhance scalability and reduce cost of content delivery.

Traditional Content Delivery Networks (CDNs) rely on large-scale deployment of dedicated servers, which costs hundreds of millions of dollars annually to deploy and maintain [3], [8]. These costs are also expected to significantly increase with the emergence of new content (e.g., mixed reality) due to the need to provision more capacity. PCDNs, on the other hand, leverage under-utilized resources at the network edge to store and distribute data, resulting in much lower cost to serve each user's data. PCDNs can be either deployed as a fully decentralized replacement of CDNs or deployed alongside traditional CDNs to create a hybrid architecture [10]–[12].

In a PCDN architecture, edge devices, including set-top boxes, smart home gadgets, and access points function as peer-to-peer nodes. These devices offer resource elasticity, allowing edge devices to join or leave the network dynamically and enabling content providers to scale resources up or down based on demand. While decentralized architectures address certain limitations of centralized systems, they also introduce new challenges. These include less compute and storage resources and less stable communication links compared to CDN architectures. To provide an equivalent user experience, PCDN architecture can use multiple edge devices to distribute data in parallel to each client. This parallel data transmission can increase the communication bandwidth, reduce latency, and increase robustness to link or server failures.

Over the last two decades, multipath transport protocols such as Multipath TCP (MPTCP [13]) and Multipath QUIC (MPQUIC [14]) have been developed to enable communication between a single client and a single server using multiple network paths simultaneously. These transport protocols can be adapted and deployed in PCDN architectures, enabling communication between multiple servers and a single client. However, the heterogeneity in path *data rates* and *packet drop rates* introduces challenges to existing multipath transport protocols (e.g., MPTCP/MPQUIC), such as Out-of-Order (OOO) packet delivery and Head-of-Line (HoL) blocking. Further, the change in architecture from a single server communicating with a single client (over multiple paths) to multiple servers communicating with a single client (over multiple paths) exacerbates these challenges. In fact, as we demonstrate later, existing multipath transport protocols can get total throughput values even lower than those achieved with a single path.

To illustrate the HoL blocking and OOO packet delivery issues, consider the scenario illustrated in Fig. 1, where six data segments, numbered sequentially from 1 to 6, are transferred

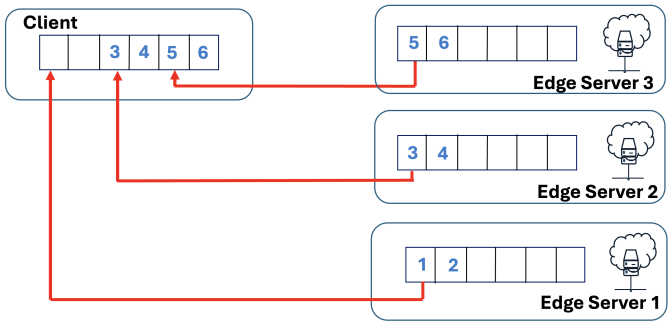

Fig. 1. **A PCDN where different pieces of content come from different servers and paths. The heterogeneity in paths' data rates and packet drop rate characteristics, can cause some parts of data (e.g., segments 3-6 in this figure) arrive earlier than other parts (e.g., segments 1 and 2). Since application requires packets to be delivered in order, this can cause HoL blocking, degrading overall throughput and client QoE.**

from three edge servers to a single client. Each edge server is assigned two segments: edge1 transmits segments 1 and 2, edge2 handles segments 3 and 4, and edge3 transmits segments 5 and 6. If edge1 experiences delays or packet losses, while edge2 and edge3 deliver their data successfully and on time, the client receives segments 3 to 6 before segments 1 and 2. Despite the timely arrival of segments 3 to 6, the application cannot process them due to the missing data from edge1. Many applications such as music players and video players require data to be received in sequential order for proper playback. This scenario—where later data segments arrive but cannot be processed due to missing earlier segments—results in OOO data arrival and leads to HoL blocking. Consequently, this phenomenon degrades throughput and reduces the user's Quality of Experience (QoE).

Such HoL blocking underscores critical challenges in decentralized multipath architectures, particularly in ensuring seamless data delivery from heterogeneous edge devices. Overcoming these issues is essential to enhance the efficiency and reliability of decentralized systems, especially as demand grows for high-performance content delivery in applications like video streaming and immersive experiences.

We argue that HoL blocking and OOO packet delivery limitations stem from fundamental issues in existing multipath protocols like MPTCP, where components such as schedulers, congestion control, and retransmission queues at the multipath level conflict with similar mechanisms at the single-path level. To address these challenges, we introduce MPRD (Multipath Parallel Reverse Segment Download), a novel distributed transport protocol specifically designed for PCDN architectures. MPRD builds on single-path TCP (SPTCP) and serves as a replacement for MPTCP in PCDN networks. Specifically, this paper makes the following contributions:

- **MPRD Design:** We develop a new multipath transport protocol, MPRD, designed specifically for PCDNs. MPRD is a receiver-driven solution in which the client establishes multiple SPTCP connections and coordinates the starting point and transmission window offset at each server to parallelize segment downloads. The selection of edge servers is guided by information from a tracker entity and the path characteristics to each edge server. We demonstrate that MPRD eliminates HoL blocking and OOO packet delivery observed in existing multipath schedulers, incurs minimal communication overhead, and achieves a total throughput close to the sum of the throughput of each individual path.
- **Evaluation:** We conduct extensive experiments to evaluate MPRD across a variety of applications, path characteristics (data rate, loss), and numbers of clients and servers. We show that compared to the state-of-the-art, MPRD increases the aggregate throughput by a factor of 4-30x. We also show that MPRD reduces the load on the CDN up to 90%, when the CDN is augmented by a PCDN. For video applications, we show that MPRD significantly reduces the percentage of pauses (stalls) and improves the PSNR, which is a key metric for visual quality, by up to 35 dB.

The paper is organized as follows: Sections II and III cover the background and motivation. Section IV details the MPRD design. Implementation and evaluation are discussed in Sections V and VI, respectively. Related work and conclusions are presented in Sections VII and VIII.

II. Background

CDN. A CDN (depicted in Fig. 2(a)) operates on a hierarchical client-server architecture, with clusters of servers distributed across various geographical regions. These clusters deliver content, such as videos and files, to edge servers located closer to users, reducing latency and improving performance. CDNs utilize the Domain Name System (DNS) to minimize communication delays by directing users to the nearest edge server, ensuring load balancing and efficient resource utilization. Providers often scale server clusters and integrate dedicated ISP networks to maintain high-speed, low-latency communication. Prominent CDN providers, such as Akamai and Limelight, either deploy edge servers globally or rely on centralized data centers interfacing with ISPs. However, the high infrastructure costs required to scale CDNs and meet the growing bandwidth demands of Internet users remain a significant challenge.

PCDN. A PCDN (depicted in Fig. 2(b)) is a decentralized system architecture where peers (e.g., edge devices) function as both content providers and consumers. In a PCDN, each peer can contribute to content storage, forwarding, and/or consumption. Unlike the DNS-based redirection employed in CDNs, PCDNs utilize a *tracker* entity to connect clients with peers that store the requested content. A PCDN can either be deployed in isolation as a replacement of a CDN or alongside a CDN to create a hybrid CDN-PCDN architecture. One key benefit of a hybrid architecture is to reduce the traffic load on the original CDN server.

Tracker. A tracker serves as a key component that facilitates efficient content delivery by managing peer discovery and connection setup. Unlike traditional CDNs that rely on DNS-based redirection, the tracker operates as a coordinating entity,

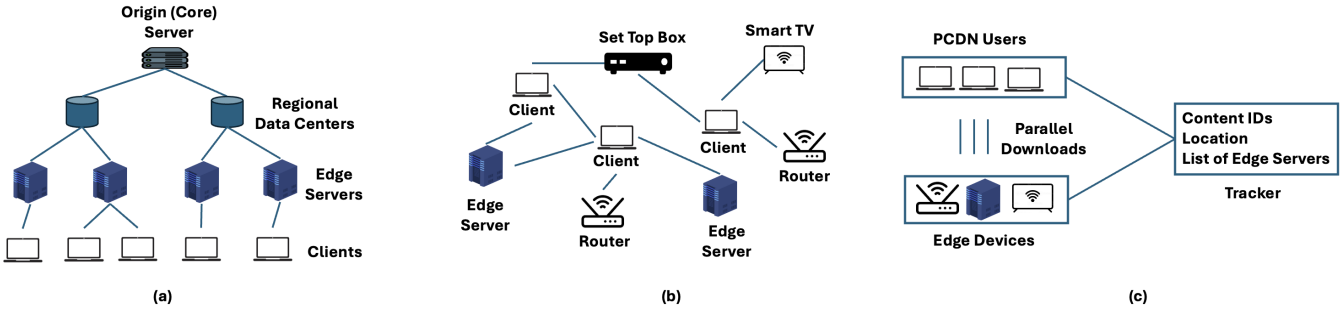

Fig. 2. **(a): CDNs use a hierarchical structure with origin servers at the root, regional data centers in the middle, and edge servers close to end users. (b): PCDNs employ a decentralized architecture, operating standalone or in hybrid mode to reduce CDN load. (c): A tracker lists edge servers and their stored content, guiding clients to the appropriate servers for downloads.**

maintaining a directory of peers and the content they store. When a client requests specific content, the tracker identifies and provides a list of suitable peers that can serve the request, optimizing selection based on factors such as proximity, bandwidth, and network load. By helping establish connections between clients and peers, the tracker ensures seamless data retrieval while supporting load balancing and efficient resource utilization. This functionality makes the tracker a cornerstone of decentralized architectures, enabling scalable and cost-effective content delivery.

User Interactions in PCDNs. The workflow of a PCDN revolves around the collaborative sharing of content among peers, such as edge devices. When a client requests specific content, it first communicates with a tracker entity, which maintains a directory of available peers and their stored content. The tracker identifies suitable peers based on criteria like proximity, bandwidth, and availability, then provides the client with a list of peers capable of serving the content. The client establishes direct connections with these peers and downloads the content from one or multiple sources. Throughout this process, peers may also cache the content they download, making it available to other clients and reducing the load on the original content source.

Multipath Transport Schedulers. Multipath transport for PCDNs enables clients to leverage multiple network paths simultaneously to improve data throughput, reduce latency, and enhance reliability. In this context, multipath transport protocols such as MPTCP can facilitate communication between clients and multiple peers or servers. To achieve optimal performance, these protocols rely on schedulers to determine how traffic is distributed across paths. Among the widely adopted MPTCP schedulers, minRTT [15] focuses on optimizing performance by prioritizing data transmission over subflows with the lowest Round-Trip Time (RTT). It retransmits blocked segments via the fastest available path, penalizing subflows responsible for delays. While this approach ensures lower latency, it can result in under-utilization of bursty network paths, limiting its ability to achieve optimal capacity aggregation. To address these issues, the BLEST scheduler was introduced to enhance MPTCP performance over heterogeneous paths [16]. BLEST actively monitors the send window and minimizes delays caused by insufficient buffer space, ensuring faster subflows are not unnecessarily idle. In [17], the FBDT protocol is introduced to eliminate the HoL blocking issue, but the design is specific to a single server communicating with a single client over multiple paths. MPRD extends this design to PCDN architectures with multiple servers.

III. Motivation

We conduct preliminary experiments to highlight the poor performance of existing MPTCP schedulers in PCDN architectures. The experimental topology consists of two servers: a CDN server and an edge server, both hosted on a single Linux desktop machine. A single client, instantiated on a separate Linux machine, is connected to both servers. Details of our system implementation are provided in Section V.

Fig. 3(a) illustrates the throughput of each server to the client, both in isolation and when serving the client jointly. Without packet loss or delay, the CDN server achieves a data rate of approximately 10 Mbps, and the edge server achieves 4 Mbps (leftmost bar). When using MPTCP's minRTT scheduler for multipath download, the aggregate throughput is about 5% lower than the sum of the individual server throughputs. However, introducing packet loss or delay to the edge server using Linux TC reveals a significant impact on multipath download with minRTT scheduler. While packet loss or delay has visible effect on the edge server's standalone data rate, the impact on aggregate throughput with minRTT is substantial: a 10% packet loss rate reduces throughput by 79%, and a 200 ms delay reduces throughput by 60%. Repeating the experiments with the BLEST scheduler shows only minor improvements in throughput. This is because existing MPTCP schedulers are highly susceptible to OOO packet delivery and HoL blocking, which degrade throughput.

The reduction in transport layer throughput directly impacts client QoE. For instance, Fig. 3(b) illustrates the relationship between PSNR (a widely used visual quality metric) and bitrate for the video Big Buck Bunny [18]. A drop in video bitrate from 4 Mbps to 1 Mbps reduces the PSNR from approximately 46 dB to 36 dB, indicating a significant decline in visual quality. This highlights the need for robust multipath transport solutions capable of maintaining high throughput despite packet loss or delay, conditions commonly encountered in both wireless networks (e.g., due to mobility or interference) and wired networks (e.g., due to router congestion).

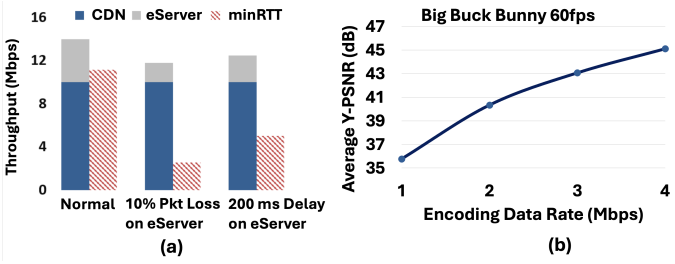

Fig. 3. **(a): Throughput of each server in isolation and when multipath download with minRTT scheduler is used. Introduction of packet loss or delay can significantly degrade MPTCP throughput. (b): PSNR vs. bitrate for Big Buck Bunny video. Drop in transport layer data rate can cause large drops in client visual quality.**

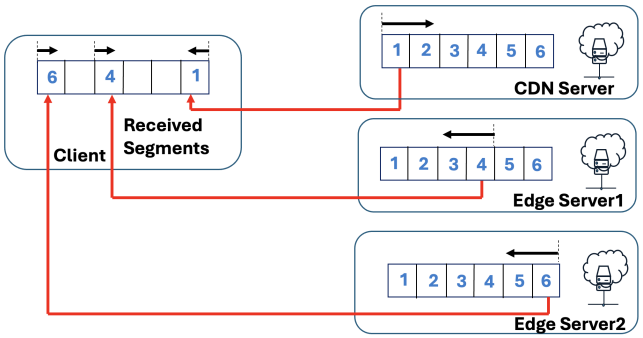

Fig. 4. **All three servers have segments 1 to 6 in their transmission window. The most reliable server is selected by the client to serve segments from the forward direction. Other servers serve segments from the reverse direction. The starting points and directions are determined by the client.**

IV. Design of MPRD

In this section, we introduce the design of MPRD, starting with an overview of its architecture, followed by a detailed explanation of the protocol.

A. Overall Architecture

Downloading content concurrently from multiple servers is an effective way to enhance throughput. However, ensuring sequential data delivery in the face of varying link data rates and packet loss presents a significant challenge. For example, consider two servers, Edge Server 1 and Edge Server 2, tasked with delivering six data segments labeled 1 through 6. A naive strategy might assign Edge Server 1 to send segments 1–3 and Edge Server 2 to send segments 4–6. In this case, any delay or packet loss from Edge Server 1 would make segments from Edge Server 2 unusable until segments 1–3 arrive. This problem, known as the HoL blocking issue, reduces effective throughput to the slower server's rate, rather than achieving the combined throughput of both servers.

To address the HoL blocking limitations, we propose a bidirectional downloading mechanism. In this approach, Edge Server 1 leverages SPTCP to transmit data sequentially in ascending order (1 to 6), while Edge Server 2 leverages another SPTCP to transmit in descending order (6 to 1). This dual-direction strategy allows the streams to converge, enabling sequential assembly of data at the client and effectively combining the throughput of both servers. For instance, if Edge Server 1 encounters delays transmitting segment 2, Edge Server 2 can finish segments 6 to 4 and then transmit segments 3 and 2, ensuring seamless delivery.

This approach maximizes throughput by leveraging the full capacity of both servers, despite individual uncertainties. Additionally, it enhances reliability and optimizes resource utilization, making it a robust solution for content delivery in PCDNs. Building on this principle, we extend the mechanism to multiple download sources and introduce the MPRD scheduler, which operates over single-path TCP.

The MPRD scheduler facilitates simultaneous download of data segments in both forward and reverse directions, using specific *segment offsets* to preserve order. The client first contacts the tracker to retrieve a list of servers hosting the requested content. MPRD then assigns the most reliable server (e.g., based on historical data) to transmit segments in the forward direction, while other servers handle reverse direction transmissions. To ensure orderly data delivery, the client assigns distinct segment offset to each edge server.

Through this coordinated approach, MPRD eliminates the delays caused by OOO packets, improving throughput and minimizing the buffering overhead at the client. By aligning segment downloads in multiple directions, MPRD optimizes data flow and balances network load across multiple servers, making it a powerful solution to meet the demands of real-time applications such as streaming and live content delivery.

MPRD emphasizes using the most reliable server to transmit data in the forward direction, while other servers handle reverse-direction transmissions. When segments from both directions converge, the client communicates updated segment offsets to all servers. The order of reverse-transmitted segments and the sequence of participating servers are determined based on their historical performance metrics.

For example, consider the scenario illustrated in Fig. 4 with one CDN server and two edge servers, all of which have access to data segments 1 through 6. The CDN server begins transmitting in the forward direction, starting with segment 1, while Edge Server 1 and Edge Server 2 transmit in the reverse direction, starting with segments 4 and 6, respectively. This reverse transmission strategy allows each edge server to operate at its own data rate.

When forward and reverse transmissions overlap, the client automatically detects the convergence point and adjusts segment offsets to maintain seamless delivery. The CDN server continues to deliver segments sequentially at its data rate, supplemented by the reverse-transmitted segments from the edge servers. These reverse segments effectively accelerate the overall throughput by augmenting the forward data flow.

This multipath transmission strategy eliminates HoL blocking, reduces retransmission overhead, and significantly increases total throughput.

B. MPRD Detailed Design

For ease of discussion, consider a hybrid CDN+PCDN architecture composed of a CDN server and some edge servers.

In MPRD, the client would simultaneously request the data (e.g., files or videos) from the CDN server and queries the PCDN tracker for a list of eligible edge servers. The tracker responds with a list of proximate edge servers capable of serving the requested content.

Unlike conventional vertical buffering mechanisms, MPRD utilizes a horizontal buffer, known as the *transmission window* (TW), with a predefined capacity of *N* megabytes. All servers load the same data segments in their transmission window, but end up transmitting different sets of segments. The client opens a similar window on its side (referred to as reception window) and populates it as segments arrive. Each data segment within each transmit window buffer is assigned the same sequence numbers, ensuring orderly data management.

At the outset, the MPRD scheduler sends a synchronized message to the CDN server and selected edge servers. This message includes essential details such as the requested content, the start sequence number, the offset, and the transmission direction for each server. The start sequence number specifies the first segment in the requested data, while the offset indicates the position from which the server should begin transmitting segments. The direction parameter determines whether data transmission proceeds in ascending (forward) or descending (reverse) order.

The forward server, responsible for transmitting data in ascending order, is ideally designated as the CDN server. However, if the CDN server is unavailable, the edge server with the highest throughput is selected as the forward server. Edge servers are then sorted based on their estimated throughput and positioned within the transmission window. The initial throughput estimate can be determined by *W/RTT*, where W denotes the congestion window size and RTT is the round trip time between the server and client.

Each edge sever is assigned a transmission window offset, computed using Eqs. 1 and 2. Here, K is the total number of edge servers, R_k denotes the throughput of edge server k, and R_0 denotes the CDN throughput. In Eq. 1, the estimated number of segments to be sent by each server is first determined. These estimates are then aggregated to compute the transmission window offset for each edge server (Eq. 2).

$$\widetilde{SegSent}_j = N(\frac{R_j}{R_0 + \sum_{k=1}^{K} R_k}), \quad \forall j \in \{0, ..., K\} \tag{1}$$

$$TW_{offset_k} = \sum_{j=0}^{k} \widetilde{SegSent}_j, \quad \forall k \in \{1, ..., K\} \tag{2}$$

The transmission window offset determines the starting sequence number for each edge server's transmission. While the forward server transmits data sequentially in ascending order and does not have an offset, all other edge devices transmit data in descending order, starting from their assigned transmission window offsets. Initially, MPRD distributes edge servers across the transmission window based on estimated throughput. However, this distribution may dynamically adjust

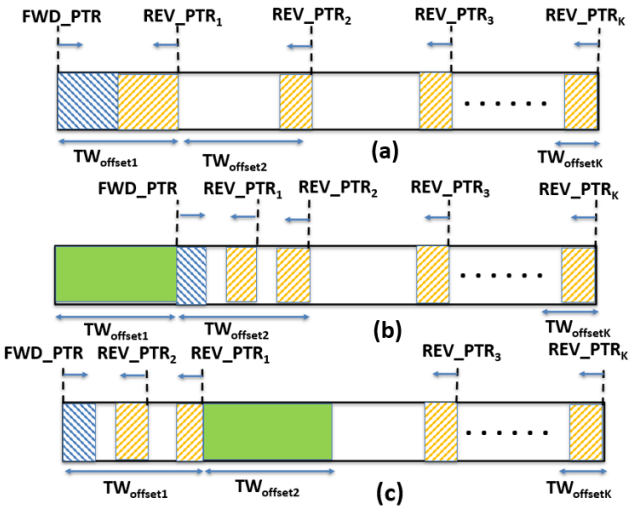

Fig. 5. **(a): Initial placement of forward and reverse pointers at the client reception window. The dashed parts represent the segments transmitted by the associated server. (b): When forward and reverse pointer 1 cross over (i.e., finish transmitting the segments between them), MPRD reassigns them to help with transmitting segments in $TW_{offset2}$. The green blocks shows the segments received by the client. (c): As reverse pointer 2 finishes transmitting its segments, MPRD reassigns it to be in between the forward pointer and reverse pointer 1.**

across different transmission windows to better align with the throughput and reliability of each individual server over time.

MPRD assigns TW_{Offset} (starting sequence numbers) and transmission directions (either forward or reverse) to all servers. Upon receiving these instructions, servers begin populating the client's reception window buffer according to their assigned starting sequence numbers and directions. The MPRD scheduler continuously monitors data flow within the client's reception window, employing pointers to track progress for each server. Specifically, the forward server is tracked using fwd_ptr, while reverse servers are monitored using rev_ptr_1, rev_ptr_2, ..., rev_ptr_K. Fig. 5(a) shows these pointers on the client's reception window.

Initially, these pointers are positioned at specified offsets, as communicated to the servers. As data transmission progresses, the pointers increment or decrement their positions in the client's reception window, based on received sequence numbers. Over time, some pointers would start to overlap and cross each other. The scheduler identifies crossover points—positions where two or more pointers intersect. Each pointer has defined crossover conditions: for instance, fwd_ptr encounters a crossover when it aligns with any rev_ptr. Similarly, for reverse pointers, rev_ptr_k experiences a crossover when it reaches the initial position of rev_ptr_{k-1}. The crossover points are based on the speed of segment transmissions from each server.

Managing crossover points is critical for achieving high throughput in PCDN networks with heterogeneous server data rates. In MPRD, once a crossover is detected, the corresponding servers halt their transmissions and enter an idle state. To optimize resource utilization, idle pointers are reassigned by the client to manage undelivered data chunks

with lower sequence numbers within the transmission window. The detailed mechanics of these crossovers are discussed next.

Forward and Reverse Pointer Crossover. When the forward server and reverse server 1 finish transmitting segments in $TW_{offset1}$, MPRD migrates the fwd_ptr and rev_ptr_1 to assist in retrieving unreceived segments originally allocated to rev_ptr_2. This migration prioritizes rev_ptr_2 over other reverse pointers due to the urgency of receiving lower-sequence-numbered data compared to higher-sequence-numbered data[1]. Initially, segments within $TW_{offset2}$ have been assigned to rev_ptr_2 for reverse transmission. However, when portions of $TW_{offset2}$ remain unreceived, the MPRD scheduler redistributes these remaining segments to fwd_ptr and rev_ptr_1, as depicted in Fig. 5(b). The new segment offset for rev_ptr_1 is determined based on Eq. 2. For smaller-sized unreceived segments, rev_ptr_1 is positioned near the midpoint of the remaining data to balance the transmission load efficiently.

Reverse Pointer Crossover. While fwd_ptr and rev_ptr_1 are generally expected to complete their transmissions earlier than reverse pointers assigned to other servers, link rate and loss variations across servers can lead to different outcomes. In some cases, one or more servers associated with reverse pointers may complete their tasks ahead of schedule, leaving them idle. To address this, the MPRD scheduler reallocates these idle servers to accelerate the retrieval of unreceived segments associated with fwd_ptr and rev_ptr_1. This reassignment prioritizes lower-sequence-numbered data to ensure efficient data transmission. Idle reverse pointers are positioned between fwd_ptr and rev_ptr_1, as determined by Eq. 2. For smaller-sized unreceived TW offsets, the new reverse pointer is placed closer to the midpoint between fwd_ptr and rev_ptr_1, as illustrated in Fig. 5(c). This approach optimizes the distribution of resources, ensuring that idle servers contribute effectively to minimizing delays in data retrieval.

MPRD Total Throughput is Close to the Summation of Individual Throughput Values. To maximize download efficiency from multiple servers, it is essential to utilize the full capacity of individual servers, while avoiding interdependencies that can cause HoL blocking. MPRD achieves this by serving the transmission window from both forward and reverse directions, and continuously monitoring and reassigning servers as servers finish their tasks. This can, however, sometimes cause redundant packet transmissions at the transport layer. For example, suppose there are two segments in the transmission window and two servers. Suppose forward server has sent segment 1 and reverse server has sent segment 2. Now suppose the client receives segment 1 but not segment 2. In this case, forward server will send segment 2 irrespective of the transmission from reverse server. Delayed arrival of segment 2 from reverse server will result in duplication. Duplicate packets will be detected and dropped at the client transport layer, however, this results in a total MPRD throughput that is slightly less than individual data rates as we will show through experiments in Section VI.

[1]Some applications such as live streaming and video playback may be able to process orderly received lower-sequence-numbered segments without having to wait for the entire client reception window to be filled.

Transmission/Reception Window Update. As the client reception window begins to fill, MPRD ensures that data is delivered to the application either as a complete set or incrementally, based on application requirements. Once the reception window reaches capacity, the entire client reception window is cleared (handed to the application layer). Similarly, each server refreshes its transmission window with the set of new segments that should be delivered to the client. Further, MPRD evaluates the throughput of each server and dynamically reassigns them as forward or reverse servers, as described in the design section. This periodic reassessment optimizes resource allocation for subsequent transmission windows, ensuring sustained performance and efficient data delivery.

Congestion Control. MPRD supports both coupled and uncoupled congestion control. Our current implementation of MPRD is based on uncoupled congestion control, where each SPTCP operates independent of other SPTCPs. Uncoupled congestion control can sometimes lead to unfairness and over the past several years, several works have been proposed to extend MPTCP to support coupled congestion control for improved fairness [19]. These approaches can be integrated into MPRD by regulating the amount of traffic served by each server and/or designing new pointer distribution mechanisms within the transmission window. We leave development and evaluation of such mechanisms as part of our future work.

V. Implementation

In this section, we detail our implementation, which comprises two servers connected via multiple Ethernet interfaces, along with software to emulate (virtualize) various configurations of servers, clients, and multipath characteristics (e.g., delay, packet loss rate, data rate). Specifically, we developed generic client and server software tailored for the PCDN architecture. We then implemented three multipath schedulers and transport protocols: MPTCP minRTT, P-Scheduler [3], and MPRD. Our implementation is based on Linux Ubuntu 24.04 LTS. Additionally, we created custom scripts and libraries to process and generate diverse types of data (e.g., file downloads, video processing files).

Hardware Setup. The experimental setup, depicted in Fig. 6(a), consisted of two Dell desktops equipped with Intel Core i7 processors (12 cores, 3.7 GHz), 32 GB of RAM, and running Ubuntu 22.04 LTS. The machines were interconnected through up to five dedicated Ethernet connections using USB-to-Ethernet adapters, each supporting speeds of up to 1 Gbps. These dedicated Ethernet interfaces provided precise control over bandwidth (data rate) and packet loss on each connection, enabling the emulation of real-world PCDN scenarios under varying resource constraints. While the entire setup could have been emulated on a single Linux machine using a loopback interface, this approach posed limitations: bandwidth would be shared among servers, and it would be difficult to control bandwidth or induce packet loss on individual connections. To

address these challenges, we extended the setup to two Linux machines with dedicated Ethernet interfaces.

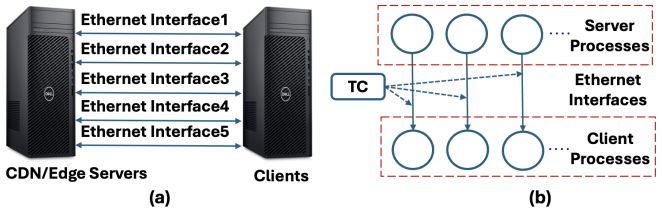

Fig. 6. **(a): Client and server processes are on two separate machines. (b): Linux TC controls bandwidth, delay, and loss for each path.**

Client and Server Software. We developed generic client and server software for PCDNs using C++ in Linux environments. The PCDN client software supports configurations for minRTT, P-Scheduler, and MPRD, offering the flexibility to scale to any number of clients and servers. In this architecture, each client can spawn a configurable number of processes, with each process independently receiving data from one of the PCDN servers (e.g., one CDN and one or more edge servers), as illustrated in Fig. 6(b). This design choice—dedicating a process to each server rather than using a single process to handle multiple servers—was made to ensure better resource allocation and higher throughput between clients and servers. On the server side, the architecture similarly supports a configurable number of processes, each functioning as an independent server (CDN or an edge server). Unique server port numbers and IP addresses are used to pair each client-server connection, enabling efficient, isolated communication with configurable bandwidth and packet loss for each pair.

Bandwidth and Loss Control. To induce packet loss and control bandwidth, we utilized the TC and Wondershaper tools in Linux. TC was used to introduce packet loss on specific interfaces between the client and server, while Wondershaper regulated bandwidth, allowing us to emulate PCDN edge or CDN servers under varying resource and bandwidth constraints. However, these tools did not perform as expected on virtual interfaces or loopback connections, making it difficult to achieve dedicated loss and bandwidth control. This limitation necessitated the use of two physical servers with dedicated Ethernet interfaces to ensure precise and reliable parameter configuration.

PCDN System Framework and Multipath Transport Implementation. In our implementation, each client process establishes a bidirectional socket connection with the server. One socket is dedicated to downloading the requested content from the server, as specified by the client-side scheduler, while the other socket handles metadata communication. This metadata includes information such as the content (e.g., video) ID, start sequence number offset, and download direction in the case of MPRD. In our performance evaluation, we consider up to five paths (client-server process pairs) between a server and a client, as depicted in Fig.6(a). We then implemented three[2] multipath transport protocols and schedulers:

- **minRTT:** The minRTT scheduler was integrated into the PCDN system software based on the minRTT approach from MPTCP [15]. In this approach, the client-side scheduler prioritizes requests to servers with the lowest RTT, ensuring that the congestion window of the PCDN connection is filled by these servers first before addressing those with higher RTTs. In our controlled environment, RTT values remain constant unless packet losses are deliberately introduced. The server is pre-configured with the RTT values for each client-server path and uses this information to prioritize filling the congestion window of the lowest RTT connections first, followed by higher RTT connections, in alignment with the minRTT strategy.
- **P-Scheduler:** We incorporated the P-Scheduler mechanism into our PCDN system design, following the methodology outlined in [3]. Each server is assigned an offset calculated using parameters such as average transmission time, data rate, packet size, and retransmission timeout. At the start of the process, P-Scheduler computes the offset for each client-server process pair based on its algorithm, and each server initiates its transmission according to the assigned offset. Once a server completes its transmission within the specified offset, P-Scheduler dynamically reassigns a new offset to the client-server process pair, ensuring efficient and orderly data transfers across connections.
- **MPRD:** The multipath transport protocol architecture and scheduling follow the design outlined in Section IV. Specifically, the scheduler assigns sequence numbers and forward/reverse directions to all client-server process pairs. Each client process reports successfully received data to the MPRD scheduler using sequence numbers. The scheduler continuously monitors the transmission window for crossovers, as described in Section IV. Upon detecting a crossover, the MPRD scheduler promptly reassigns sequence numbers to the completed client-server process pairs, ensuring efficient data flow and seamless coordination across the different paths.

VI. Results and Discussion

In this section, we present the results of our extensive performance evaluation. We begin by analyzing MPRD's performance in terms of throughput aggregation in both high and low data rate regions. Next, we examine the reduction in CDN load when a PCDN is used to augment a CDN server. We then evaluate MPRD's ability to enhance video performance using metrics such as PSNR and percentage of pauses (re-buffering). Finally, we assess the time required to download a file, emulating a multipath BitTorrent application. In all evaluations, we compare MPRD's performance against the default Linux scheduler (minRTT) and the state-of-the-art

[2]We also implemented BLEST [16] and observed that it consistently gets performance in between minRTT and P-Scheduler. Thus, we eliminated its evaluation results for ease of presentation.

(P-Scheduler). Results from BLEST are omitted, as its performance consistently fell between minRTT and P-Scheduler.

Scenario. We consider a hybrid CDN+PCDN architecture, where one or more edge PCDN servers augment the CDN server. The CDN server is characterized by a higher data rate to the client compared to the edge servers. This scenario also captures a pure PCDN architecture, where one server achieves a higher data rate to the client than the other servers.

A. Throughput Aggregation

Throughput Measurement Procedure. For all three multipath schemes examined in this study, each client process logs the received data, including sequence numbers and timestamps, into individual CSV files. These files are then post-processed. Sequence numbers are inserted into a priority queue for throughput estimation at 1-second intervals, based on the logged timestamps. The priority queue maintains an ascending order of sequence numbers, and the total in-sequence data in the queue represents the number of packets received within a given second. In this study, a data packet size of 1400 bytes was used in all experiments. Throughput was calculated by multiplying the number of in-sequence packets in the priority queue per second by the packet size.

Throughput Statistics. Video service providers, such as Bytedance, report that 80% of users in China stream videos at rates between 2–4 Mbps, with 1% utilizing up to 8 Mbps [3]. Additionally, the average CDN speed in China is approximately 24 Mbps. In the USA, the average ISP downlink and uplink throughputs are 200 Mbps and 24 Mbps, respectively. For mobile networks, 5G and 4G in the USA achieve theoretical maximum downlink speeds of up to 2 Gbps, with an average downlink speed of 50 Mbps [20]. Using these statistics, we designed two scenarios to evaluate throughput: one with low individual server data rates and another with high rates.

Specifically, we conducted tests using configurations with 2, 3, 4, and 5 total number of servers. In the low-throughput setting, server-to-client links were bandwidth-limited to 10 Mbps, 4 Mbps, 4 Mbps, 3 Mbps, and 1 Mbps in the five-server setup, with the CDN server providing the highest data rate (10 Mbps). This heterogeneous setup was designed to produce out-of-order packets, mimicking real-world network variability and enabling an assessment of each scheduler's effectiveness. For the high-throughput setting, bandwidths were set to 500 Mbps, 350 Mbps, 350 Mbps, 200 Mbps, and 200 Mbps for the respective server-to-client links. This configuration aimed to evaluate performance under conditions with higher bandwidth and less stringent link limitations.

All three schedulers—minRTT, P-Scheduler, and MPRD—were evaluated under two conditions: (1) without packet loss and (2) with a 10% packet loss introduced on edge server 1. For systems without packet loss, minRTT achieved about 80% of the total system throughput, P-Scheduler achieved about 85%, and MPRD demonstrated the best performance by achieving about 87-90% of the total system throughput across all configurations as shown in Fig. 7(a) (low throughput configuration) and 7(d) (high throughput configuration). Here, the first bar for a given x-axis value shows the throughput of each individual server in isolation. The second, third, and fourth bars show the throughput values of minRTT, P-Scheduler, and MPRD, respectively.

When a 10% packet loss was introduced to edge server one, MPRD consistently retained 75 to 85% of the total throughput across all configurations, while minRTT and P-Scheduler collapsed significantly during the 1, 2, 3, and 4 edge server configurations, achieving only 2–3% of the sum throughput as shown in Fig. 7(b) (low throughput configuration) and 7(e) (high throughput configuration). This collapse was attributed to the inefficiency of minRTT and P-scheduler to handle OOO packets and HoL blocking issues.

The results highlight that loss and jitter, common in real-world scenarios, can severely degrade the performance of minRTT and P-Scheduler. On the other hand, MPRD demonstrated robustness and superior loss tolerance, maintaining higher throughput even under adverse conditions. These findings underscore MPRD's effectiveness as a multipath transport solution for practical PCDN deployments, ensuring optimal QoE for users with varying number of servers.

B. CDN Load Reduction

One of the primary challenges for content service providers using CDNs is addressing the demands of an ever-increasing user base and the escalating bandwidth requirements. Factors such as higher video frame rates and the growing adoption of emerging high-volume content formats (e.g., 4K or 8K video) place considerable strain on CDN infrastructure, both in terms of cost and scalability. A PCDN system can mitigate this bottleneck by offloading some of the content delivery responsibilities to edge servers. This distributed approach not only eases the load on central CDN servers but also helps reduce operator costs, making it a critical solution for handling modern content delivery demands.

The effectiveness of a PCDN system in reducing CDN load depends on how well the edge servers offload traffic from the CDN, while maintaining the required user QoE. We next proceed to evaluate the load reduction capabilities of the three multipath systems studied in this paper.

CDN Load Measurement Setup. To quantify the reduction in load when a CDN server is augmented with edge PCDN servers, we conducted a series of measurements. The client downloaded data with sequential numbers from both the CDN and edge servers. During a 5-minute interval, we logged the amount of data (in bytes) transferred from the CDN and each edge server to the client. These measurements were taken at the interface between the transport and application layers because packets received OOO at the transport layer cannot be delivered to the application layer, and hence, do not contribute to application-layer throughput. The logged data was post-processed to calculate the CDN load. This load was determined as the ratio of the bytes received from the CDN server to the total bytes received from all servers.

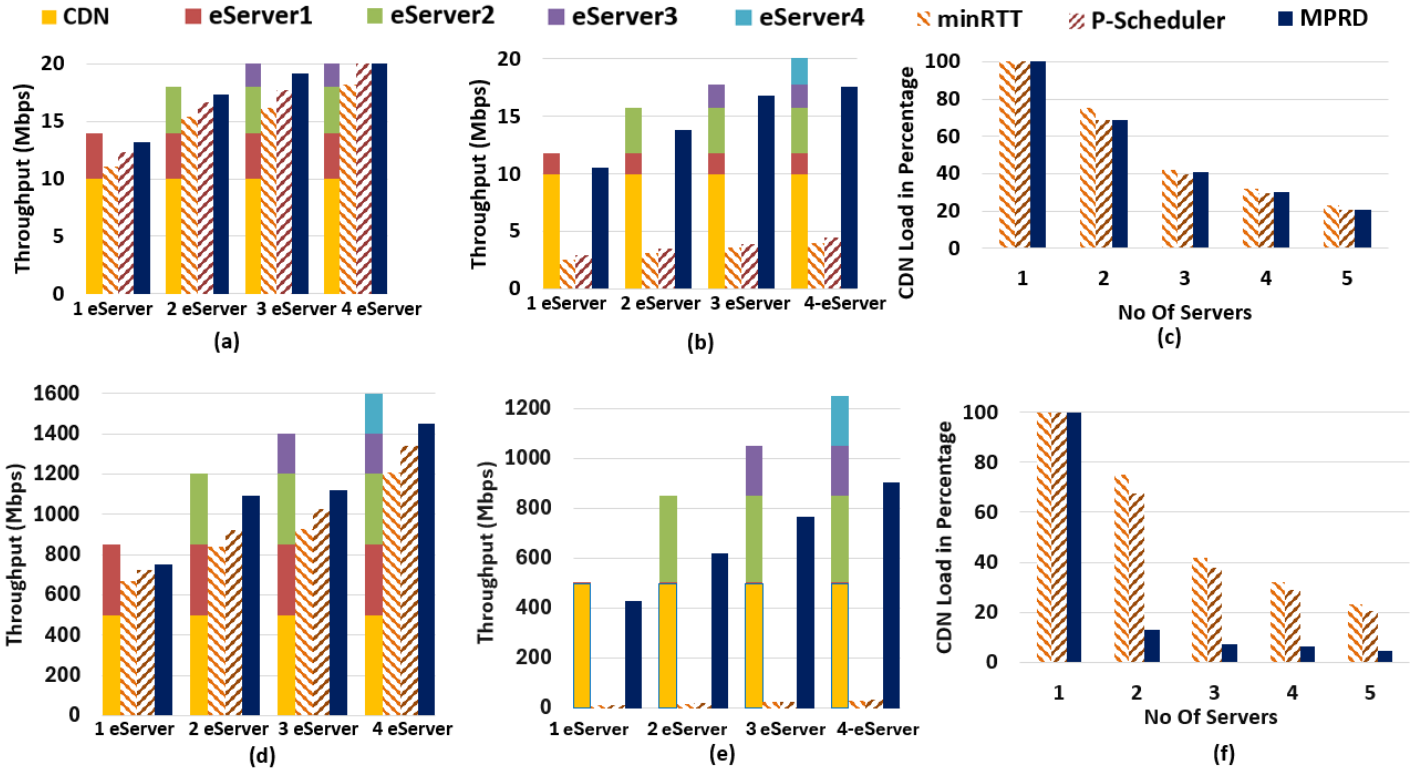

Fig. 7. **In the top row, server-to-client links were bandwidth-limited to mimic a low throughput configuration. In the bottom row, the rates were set to mimic a high throughput setting. (a): Throughput across different schemes as a function of number of edge servers. No server experiences packet loss. (b) Throughput results when edge server one experience 10% packet loss rate. (c) CDN load as a function of number of servers (CDN server + zero to four edge servers). No server experiences packet loss. (d) Throughput across different schemes in a high throughput configuration. No server experiences packet loss. (e) Throughput results when edge server one experience 10% packet loss rate. (f) CDN load as a function of number of servers (CDN server + zero to four edge servers). Edge server one experience 10% packet loss rate.**

CDN Load Reduction Characterization. Fig.7(c) illustrates the CDN load as a function of the total number of servers. When there is only one server, it is solely the CDN server. In these experiments, no server was subjected to packet loss. Further, the server data rates remain the same as the high throughput setup in Fig.7(d). We observe that all three mechanisms achieve close CDN loads, regardless of the number of servers. This is because, in the absence of packet loss, all three schemes achieve close throughput values, as discussed in Fig.7(d). We also observe that as the number of servers increases, the CDN load progressively reduces, demonstrating the efficacy of PCDNs to reduce CDN load.

Fig. 7(f) illustrates the CDN load as a function of the total number of servers, with edge server one experiencing a 10% packet loss rate. The results show that MPRD significantly outperforms both minRTT and P-Scheduler, achieving a significant reduction in CDN load. Additionally, as more servers are added to the PCDN system, MPRD consistently maintains a substantially lower load on the CDN, demonstrating its effectiveness in mitigating the impact of packet loss.

C. Video Performance

Video Data Set and Preparation. For the video streaming evaluations, we used a 5-minute clip from the widely used 3D animated short film Big Buck Bunny [18]. The original video was encoded in H.264 format at a resolution of 3840x2160 and a frame rate of 60 fps. To prepare it for streaming, we employed x264 [21] via ffmpeg [22] to encode the video into four representations with bitrates of 1 Mbps, 2 Mbps, 4 Mbps, and 8 Mbps, with resolutions of 360p, 480p, and 1080p (for the two highest bitrates). The frame rate was fixed at 60 fps across all representations, with a uniform GOP (Group of Pictures) size of 60 frames, achieved by inserting keyframes at one-second intervals. During streaming, the client logged bytes received at one-second intervals and recorded any pauses when the received data fell short of the required amount for a single GOP. We then calculated the percentage of pauses over the entire video duration. Additionally, we assessed video quality by computing PSNR values relative to the original video.

Video Performance in terms of PSNR and Stalled (Re-buffering) Times. We consider a scenario with one CDN with 10 Mbps data rate and one edge server with 4 Mbps data rate and 10% packet loss rate. Fig. 8(a) and (b) show percentage of client pauses and PSNR versus video bitrate, respectively.

From Fig. 8(a), we observe that minRTT and P-Scheduler experience significant re-buffering, with stall rates from 60% to 95%. Further, P-Scheduler only slightly improves video performance against min RTT in terms of pauses. We also observer that MPRD consistently avoided re-buffering even for higher bitrate videos, highlighting its ability to deliver

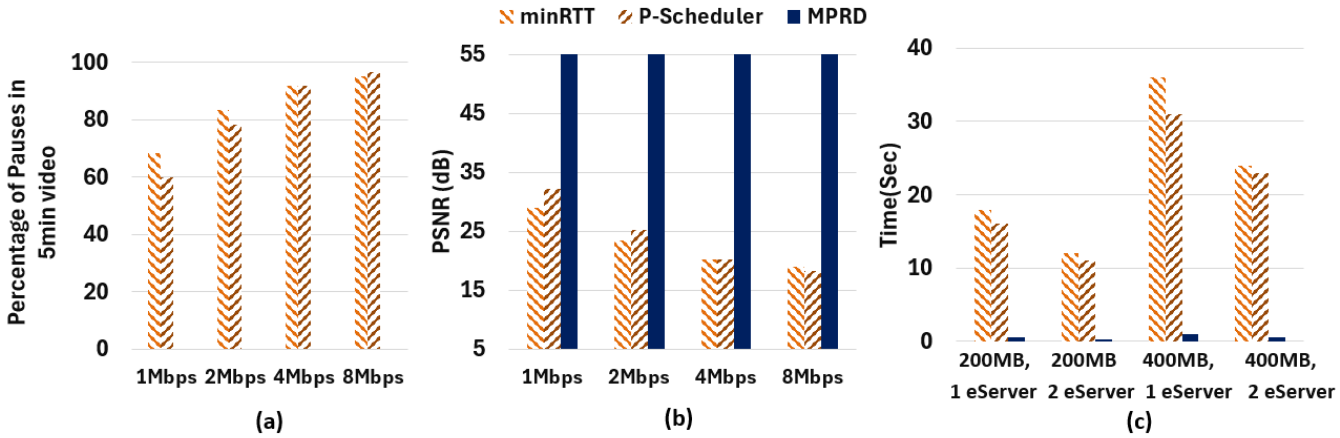

Fig. 8. **(a): Percentage of pauses for Big Buck Bunny video. (b): PSNR. (c) File download time.**

uninterrupted playback.

To further quantify the client QoE, we measured the PSNR for all three schedulers. A PSNR value of 55 dB is typically associated with good QoE at a given bitrate. For a 1 Mbps bitrate video, clients using minRTT and P-Scheduler experienced a significant degradation in QoE, with PSNR values dropping by 25 dB as shown in Fig. 8(b). In contrast, MPRD achieved the target PSNR of 55 dB, delivering a high-quality user experience. At higher bitrates of 2 Mbps, 4 Mbps, and 8 Mbps, minRTT and P-Scheduler further deteriorated, resulting in PSNR values of 25 dB, 20 dB, and 18 dB, respectively as shown in Fig. 8(b). These results demonstrate the efficacy of MPRD in handling video applications across varying bitrates and network throughput conditions. MPRD not only ensures seamless playback but also maintains high PSNR, making it well-suited for real-world video delivery scenarios.

D. File Download Time

Measurement Setup. For file download evaluations, files of predetermined sizes (200 and 400 MB) were transferred using each of the three multipath schedulers and protocols. The time taken to download files were recorded and analyzed to assess the performance of each method. Here, we consider a topology composed of one CDN and one or two edge servers. Further, edge server one has a 10% packet loss probability.

Download Time. Fig. 8(c) illustrates the download times as a function of file size and the number of edge servers. The results show that MPRD reduces download time by a factor of 20-30x, consistently outperforming both minRTT and P-Scheduler, regardless of file size or the number of edge servers. In other words, minRTT and P-Scheduler would require 20-30 times more servers to match MPRD's download times.

This performance improvement is primarily due to the HoL blocking issues faced by minRTT and P-Scheduler when one server experiences a significantly higher packet loss rate compared to others. MPRD circumvents these limitations by employing bidirectional communication, which mitigates HoL blocking, increases total system throughput, and significantly reduces file download times.

VII. Related Work

Multipath Transmission Evaluation. Several studies [17], [23]–[31] have analyzed MPTCP performance (e.g., in terms of throughput or energy efficiency), but their evaluations are largely confined to an architecture where a single server communicates with a single client over multiple paths. This setup differs from the PCDN architectures explored in this paper. Other research [3], [8], [9] has proposed new schedulers built on MPTCP for PCDN architectures. We evaluated the performance of MPRD against the state-of-the-art in this domain (P-Scheduler) and demonstrated that MPRD delivers superior performance across various metrics, including total throughput and PSNR.

Multipath Transmission Schedulers. The Linux kernel includes several MPTCP schedulers by default, such as RoundRobin, minRTT, and BLEST [16]. RoundRobin is well-suited for scenarios where all paths have similar characteristics (e.g., delay and data rate), while minRTT and BLEST are designed to perform effectively across more heterogeneous links. Additionally, the research community has proposed various schedulers, such as those addressing challenges posed by heterogeneous paths [32]–[34], schedulers leveraging differences in subflow RTTs [32], [34]–[37], and others aimed at improving MPTCP performance for specific use cases [38]–[41]. However, all these solutions have been designed and evaluated for single-server-to-single-client multipath architectures. We compared MPRD's performance against minRTT (a default Linux MPTCP scheduler) and P-Scheduler [3], which represents the state-of-the-art for PCDN architectures. While we also conducted experiments with the BLEST [16] scheduler, the results were omitted as BLEST consistently achieved performance levels between minRTT and P-Scheduler.

Content Delivery Architectures. CDNs such as Akamai [42] and Cloudflare [43] are systems of servers that deliver web media to users based on their geographic location, ensuring low latency and high availability. PCDNs [3], [8], [9] extend this concept by leveraging end-user devices as part of the delivery infrastructure, enabling cost-effective scalability

and better utilization of edge resources. Hybrid CDNs [10]–[12] combine the strengths of traditional CDNs and PCDNs, using centralized servers for reliable delivery and peer-to-peer networks for efficient resource utilization and scalability, particularly during traffic surges or in regions with limited server presence. MPRD is a transport protocol designed for multipath architectures, with a focus on increasing throughput in PCDNs or reducing CDN load in hybrid architectures.

VIII. Conclusion

In this paper, we presented MPRD, a multipath transport protocol designed to enhance content delivery in PCDN networks by leveraging multiple servers to concurrently transmit data to each client. MPRD, built on top of single-path TCP, addresses the problem of HoL blocking in PCDN systems by dynamically assigning transmission window pointers with optimized directions and placements for each server. We implemented MPRD alongside two other schedulers in the Linux userspace and demonstrated its significant performance benefits. Our results show that when even one edge server experience packet loss, MPRD can achieve a 4–30x increase in total throughput, reduce CDN load by up to 85-90%, and improve PSNR visual quality by up to 35 dB.

IX. Acknowledgements

This research was supported in part an NSF award (CNS-1942305). We would also like to thank the anonymous reviewers for their valuable feedback, which improved the paper.

References

[1] U. Krishnaswamy, R. Singh, N. Bjorner, and H. Raj, "Decentralized cloud wide-area network traffic engineering with blastshield," in *Proceedings of USENIX NSDI*, 2022.
[2] R. Xie, Q. Tang, S. Qiao, H. Zhu, F. Yu, and T. Huang, "When serverless computing meets edge computing: Architecture, challenges, and open issues," in *IEEE Wireless Communications*, 2021.
[3] D. Wei, J. Zhang, H. Li, Z. Xue, Y. Peng, X. Pang, Y. Liu, R. Han, and J. Li, "Pscheduler: Qoe-enhanced multipath scheduler for video services in large-scale peer-to-peer cdns," in *Proceedings of IEEE INFOCOM*, 2024.
[4] "Usenet," https://en.wikipedia.org/wiki/Usenet.
[5] "Napster," https://en.wikipedia.org/wiki/Napster.
[6] "Bittorrent," https://en.wikipedia.org/wiki/BitTorrent.
[7] "Bytedance," https://en.wikipedia.org/wiki/ByteDance.
[8] R. Zhang, H. Wang, S. Shi, X. Pang, Y. Peng, Z. Xue, and J. Liu, "Enhancing resource management of the world's largest pcdn system for on-demand video streaming," in *Proceedings of USENIX ATC*, 2024.
[9] D. Wei, J. Zhang, H. Li, Z. Xue, Y. Peng, X. Pang, R. Han, Y. Ma, and J. Li, "Swarm: Cost-efficient video content distribution with a peer-to-peer system," in *arXiv:2401.15839*, 2024.
[10] B. Zolfaghari, G. Srivastava, S. Roy, H. R. Nemati, F. Afghah, T. Koshiba, A. Razi, K. Bibak, P. Mitra, and B. Rai, "Content delivery networks: State of the art, trends, and future roadmap," in *ACM Computing Surveys*, 2020.
[11] R. Farahani, H. Amirpour, F. Tashtarian, A. Bentaleb, C. Timmerer, H. Hellwagner, and R. Zimmermann, "Richter: hybrid p2p-cdn architecture for low latency live video streaming," in *Proceedings of ACM MHV*, 2022.
[12] R. Farahani, A. Bentaleb, E. Centinkaya, C. Timmerer, R. Zimmermann, and H. Hellwagner, "Hybrid p2p-cdn architecture for live video streaming: An online learning approach," in *Proceedings of IEEE GLOBECOM*, 2022.
[13] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath tcp: Design, analsysis and implementation," in *IEEE/ACM Transactions on Networking*, 2016.
[14] Q. D. Coninck and O. Bonaventure, "Multipath quic: Design and evaluation," in *Proceedigs of ACM CONEXT*, 2017.
[15] 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 *Proceedings of USENIX NSDI*, 2012.
[16] S. Ferlin, Alay, O. Mehani, and R. Boreli, "Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks," in *Proceedings of IFIP Networking*, 2016.
[17] S. Srinivasan, S. Shippey, E. Aryafar, and J. Chakareski, "Fbdt: Forward and backward data transmission across rats for high quality mobile 360-degree video vr streaming," in *Proceedings of ACM MMSYS*, 2023.
[18] "Big buck bunny," https://en.wikipedia.org/wiki/Big_Buck_Bunny.
[19] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," Tech. Rep., 2011.
[20] "Cisco report," https://www.cisco.com/site/us/en/solutions/index.html.
[21] "x264 software library," https://www.videolan.org/developers/x264.html.
[22] "Ffmpeg," https://ffmpeg.org/.
[23] Y. Chen, R. J. G. Y. Lim, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath tcp performance over wireless networks," in *Proceedings of ACM IMC*, 2013.
[24] Q. Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "A first analysis of multipath tcp on smartphones," in *Proceedings of PAM*, 2016.
[25] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan, "Wifi, lte, or both? measuring multi-homed wireless internet performance," in *Proceedings of ACM IMC*, 2016.
[26] A. Nikravesh, Y. Guo, F. Qian, Z. Mao, and S. Sen, "An in-depth understanding of multipath tcp on mobile devices: Measurement and system design," in *Proceedings of ACM MOBICOM*, 2016.
[27] Y. Lim, Y. Chen, E. M. Nahum, D. Towsley, and K. Lee, "Cross-layer path management in multi-path transport protocol for mobile devices," in *Proceedings of IEEE INFOCOM*, 2014.
[28] K. Nguyen, M. Kibria, K. Ishiz, and F. Kojima, "Feasibility study of providing backward compatibility with mptcp to wigig/ieee 802.11ad," in *Proceedings of IEEE VTC*, 2017.
[29] S. Sur, I. Pefkianakis, X. Zhang, and K. Kim, "Wifi-assisted 60 ghz wireless networks," in *Proceedings of ACM MOBICOM*, 2017.
[30] J. Zhao, J. Liu, and H. Wang, "On energy-efficient congestion control for multipath tcp," in *Proceedings of IEEE ICDCS*, 2017.
[31] J. Wu, B. Cheng, and M. Wang, "Energy minimization for quality-constrained video with multipath tcp over heterogeneous wireless networks," in *Proceedings of IEEE ICDCS*, 2016.
[32] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "Daps: Intelligent delayaware packet scheduling for multipath transport," in *Proceedings of IEEE ICC*, 2014.
[33] T. Shreedhar, N. Mohan, S. K. Kaul, and J. Kangasharju, "Qaware: A cross-layer approach to mptcp scheduling," in *Proceedings of IFIP Networking*, 2018.
[34] Y. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "Ecf: An mptcp path scheduler to manage heterogeneous paths," in *Proceedings of ACM SIGMETRICS*, 2017.
[35] S. Baidya and R. Prakash, "Improving the performance of multipath tcp over heterogeneous paths using slow path adaptation," in *Proceedings of IEEE ICC*, 2014.
[36] J. Hwang and J. Yoo, "Packet scheduling for multipath tcp," in *Proceedings of IEEE ICUFN*, 2015.
[37] D. Ni, K. Xue, P. Hong, H. Zhang, and H. Lu, "Ocps: Offset compensation based packet scheduling mechanism for multipath tcp," in *Proceedings of IEEE ICC*, 2015.
[38] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon, "Cross-layer scheduler for video streaming over mptcp," in *Proceedings of ACM MMSYS*, 2017.
[39] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan, "Mpdash: Adaptive video streaming over preference-aware multipath," in *Proceedings of ACM CONEXT*, 2016.
[40] A. Nikravesh, Y. Guo, X. Zhu, F. Qian, and Z. Mao, "Mp-h2: a client-only multipath solution for http/2," in *Proceedings of ACM MOBICOM*, 2019.
[41] Y. Guo, A. Nikravesh, Z. Mao, F. Qian, and S. Sen, "Accelerating multipath transport through balanced subflow completion," in *Proceedings of ACM MOBICOM*, 2017.
[42] "Akamai technologies," https://en.wikipedia.org/wiki/Akamai_Technologies.
[43] "Cloudflare inc." https://en.wikipedia.org/wiki/Cloudflare.