electronics

MDPI

Article

Enhancing Crowd-Sourced Video Sharing through P2P-Assisted HTTP Video Streaming

Jieran Geng and Satoshi Fujita *

Department of Information Science, Graduate School of Advanced Science and Engineering, Hiroshima University, Kagamiyama 1-4-1, Higashi-Hiroshima 739-8527, Japan
* Correspondence: fujita@hiroshima-u.ac.jp

Abstract: This paper introduces a decentralized architecture designed for the sharing and distribution of user-generated video streams. The proposed system employs HTTP Live Streaming (HLS) as the delivery method for these video streams. In the architecture, a creator who captures a video stream using a smartphone camera subsequently transcodes it into a sequence of video chunks called HLS segments. These chunks are then stored in a distributed manner across the worker network, forming the core of the proposed architecture. Despite the presence of a coordinator for bootstrapping within the worker network, the selection of worker nodes for storing generated video chunks and autonomous load balancing among worker nodes are conducted in a decentralized fashion, eliminating the need for central servers. The worker network is implemented using the Golang-based IPFS (InterPlanetary File System) client, called kubo, leveraging essential IPFS functionalities such as node identification through Kademlia-DHT and message exchange using Bitswap. Beyond merely delivering stored video streams, the worker network can also amalgamate multiple streams to create a new composite stream. This bundling of multiple video streams into a unified video stream is executed on the worker nodes, making effective use of the FFmpeg library. To enhance download efficiency, parallel downloading with multiple threads is employed for retrieving the video stream from the worker network to the requester, thereby reducing download time. The result of the experiments conducted on the prototype system indicates that those concerned with the transmission time of the requested video streams compared with a server-based system using AWS exhibit a significant advantage, particularly evident in the case of low-resolution video streams, and this advantage becomes more pronounced as the stream length increases. Furthermore, it demonstrates a clear advantage in scenarios characterized by a substantial volume of viewing requests.

Keywords: HTTP video streaming; P2P-assisted video delivery; collaborative cache; InterPlanetary File System; bundling of multiple video streams

Citation: Geng, J.; Fujita, S. Enhancing Crowd-Sourced Video Sharing through P2P-Assisted HTTP Video Streaming. *Electronics* **2024**, *13*, 1270. https://doi.org/10.3390/electronics13071270

Academic Editor: Stefanos Kollias

Received: 18 February 2024
Revised: 26 March 2024
Accepted: 27 March 2024
Published: 29 March 2024

1. Introduction

The pervasive reach of the internet, projected to connect two-thirds of the global population by 2023 (Cisco, 2018–2023), has fueled the meteoric rise of video-sharing platforms like YouTube, TikTok, and Netflix. This surge in internet users translates to a proportional increase in video traffic, solidifying the dominance of HTTP video streaming as a technology for delivering diverse high-resolution content with minimal latency. Major video delivery platforms such as YouTube support HTTP video streaming as one of the major video delivery methods in use today. This trend is further underscored by recent research citing substantial advancements in video codecs and the proliferation of video-enabled devices [1–3].

At the heart of HTTP streaming lies HTTP Live Streaming (HLS), introduced by Apple in 2009. HLS leverages segmented MPEG2-TS containers to facilitate the delivery of fragmented video and audio files via HTTP/1.1. Originally designed for HDTV broadcasting, MPEG2-TS chops video data into small transport packets, enabling error correction,

packet reordering, and concurrent playback/download. While MP4 or WMV conversion is required at the client side for playback, HLS relies on a manifest file residing on the web server, specifying the location of segmented media files and their playback order for a particular video. Similar to HLS, MPEG-DASH (Dynamic Adaptive Streaming over HTTP) [4], standardized by ISO MPEG in 2012, uses an XML-formatted MPD (Media Presentation Description) as its manifest file. Other notable HTTP/1.1 streaming technologies include Smooth Streaming [5] and CMAF (Common Media Application Format) [6].

A defining feature of HTTP streaming is the ability to divide video content into segments with varying bitrates, allowing for adaptive bitrate (ABR) switching based on network conditions. Clients experiencing bandwidth fluctuations can request lower-resolution (lower bitrate) chunks from the media server, ensuring uninterrupted playback. However, as the number of users and video content grows, so does the load on media servers. This can lead to unintended bitrate reductions due to server-side congestion, compromising user experience.

These observations underscore the difficulty of achieving scalable delivery of high-quality HTTP streaming within traditional client–server architectures that rely on dedicated servers. A promising solution to this challenge lies in leveraging peer-to-peer (P2P) technology, where clients can contribute their upload bandwidth to share video content directly with each other. This decentralized approach offers several potential advantages, including the following:

- Reduced Server Load: By offloading video delivery from central servers, P2P distribution helps mitigate congestion and bandwidth bottlenecks, especially during peak demand periods.
- Improved Scalability: As additional clients join the network, their combined upload capacity contributes to a larger pool of available resources, enabling seamless scaling to accommodate a growing user base.
- Enhanced Resilience: P2P systems exhibit inherent fault tolerance and redundancy because they can avoid single points of failure. If a particular client becomes unavailable, other peers can continue sharing the content, minimizing disruptions and ensuring overall system stability.

The potential of P2P technology for enhancing video streaming has been extensively researched over the past two decades, as evidenced by numerous studies exploring various P2P streaming architectures and protocols [7–12]. These prior works provide a strong foundation for developing robust and efficient P2P-based solutions for high-quality HTTP streaming, addressing the limitations of traditional server-centric approaches.

This paper introduces a crowd-sourced video streaming system to explore novel possibilities for P2P-assisted HTTP video streaming. The system is designed to facilitate the sharing of multiple video streams captured by users participating in large events, such as Oktoberfests and parades, on a P2P network comprising nodes associated with participating users. User-generated video streams undergo conversion to the HLS format, ensuring system versatility, and the resulting HLS chunks are cached on the P2P network and available for download upon request. See Figure 1 for illustration of the flowchart. Activation of the system aligns with the event's timing, enabling users to join and leave at their discretion. The proposed system avoids centralized online cache management, reducing the risk of processing bottlenecks and single points of failure. Only the bootstrap node (coordinator), serving as the entry point for participating peers, is installed, managing limited tasks: bootstrapping and restarting the maintenance of the P2P overlay.

The proposed system is implemented in a real network environment using the Go language, and the evaluation assesses the time required for storing the generated video stream in the P2P network and retrieving/playing the cached video content. Experimental results confirm the effectiveness of the proposed system, demonstrating its suitability for sharing user-generated video content in an ad hoc manner.

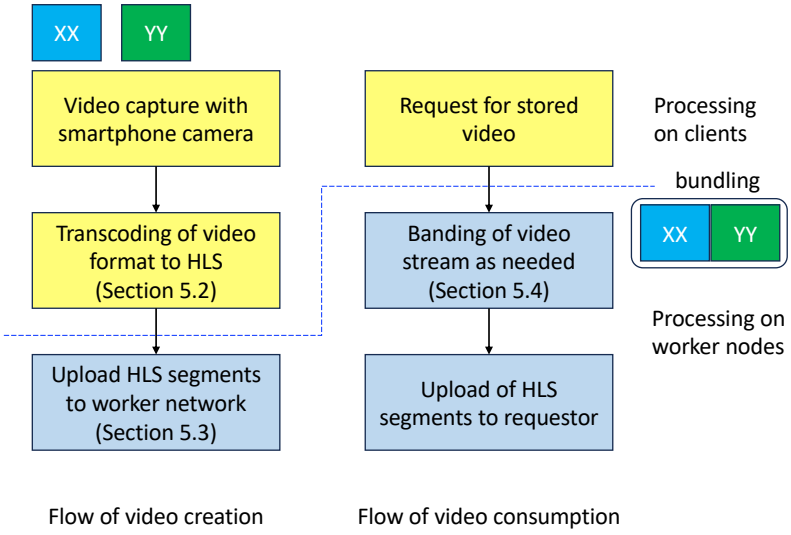

Figure 1. Flowchart of the proposed system.

The remainder of this paper is organized as follows. Section 2 outlines related work. After an overview of the proposed system in Section 3, the management of the P2P overlay, named worker network, and retrieval of user-generated video streams are described in Sections 4 and 5, respectively. Section 6 describes the results of the experiments conducted on our prototype system. Finally, Section 7 concludes the paper with future work.

2. Related Work

2.1. P2P-Assisted HTTP Video Streaming

SmoothCache [13] represents the pioneering effort in P2P live video streaming utilizing HTTP as the transport layer protocol. It accommodates both single-bitrate and multi-bitrate streaming modes, employing dynamic congestion control at the application layer to manage transfer priorities based on urgency. The paper delves into several optimization issues in HTTP Live Streaming, including neighbor management, uploader selection, and proactive caching. Network Function Virtualization (NFV) has gained recognition as an effective method for disseminating a video stream from a specific source to a large client base. Within the realm of NFV-based video streaming solutions, the hybridization of P2P and Content Delivery Network (CDN) stands out as a widely explored area. This hybrid typically encompasses three primary components: (i) media servers, including edge servers of CDN, responsible for delivering video content to peers; (ii) peers streaming the same video content at equivalent quality levels; and (iii) a tracker server maintaining a matching table to identify optimal peers viewing the same video content at matching quality levels, guided by predefined policies such as minimum latency. Noteworthy examples of such systems include the hybrid P2P-CDN service proposed by Nacakli et al. [14] and the machine-learning-based matchmaking scheme introduced by Ma et al. [15]. Additionally, Refs. [16,17] propose hybrid systems designed to curtail CDN bandwidth usage and transmission costs by customizing the HTTP Adaptive Streaming (HAS) player to incorporate a prefetch module.

Farahani et al. recently introduced ALIVE [18], a network-assisted video streaming framework tailored for HAS. ALIVE harnesses contemporary networking paradigms such as edge computing and NFV coupled with video solutions like HAS, Video Super-Resolution (SR), and Distributed Video Transcoding (TR) to effectively reduce latency and costs in live streaming scenarios. This paper presents the design of a substantial cloud-based testbed comprising 350 HAS players and conducts an experimental evaluation of ALIVE under diverse scenarios. The experimental findings demonstrate that, compared to the baseline approach, ALIVE (1) enhances user Quality of Experience (QoE) by a minimum of 22%, (2) diminishes streaming service providers' costs by at least 34%, (3) lowers

client delivery latency by a minimum of 40%, (4) exhibits improved edge server energy consumption by at least 31%, and (5) reduces backhaul bandwidth usage by at least 24%.

In a hybrid P2P and CDN architecture called RICHTER [19], proposed by the same research group as ALIVE, the challenge of serving peer requests with minimal latency and optimal quality is formulated as an optimization problem executed at the network edge. The authors introduce an online learning approach and leverage unsupervised self-organizing maps (SOM) to (1) address the time complexity issue of the optimization model and (2) enable decision-making on groups of requests rather than individual requests, rendering it suitable for large-scale scenarios.

As outlined above, although there are many existing works concerned with P2P-assisted HTTP video streaming, none of them treat crowd-sourced video streaming as in the proposed system.

2.2. Crowd-Sourced Video Streaming

In the realm of crowd-sourced live streaming, several commercial platforms are employed, primarily for live game play. Twitch, initiated in 2011, serves as a video platform predominantly dedicated to game play videos, e-sports, and live broadcasts by creators. Enabling users to live-stream their game play and daily activities, Twitch permits real-time content viewing, fostering interaction between viewers and broadcasters through its chat function. Similar live-streaming functionalities are also found on platforms such as Facebook Gaming (launched by Facebook in 2018), DLive (utilizing blockchain for donations), and Trovo (a Chinese live-streaming game platform owned by Tencent Games). In this subsection, we delve into the academic advancements in crowd-sourced live streaming research. Chen et al. proposed a comprehensive cloud-based framework for cost-effective crowd-sourced live streaming [20]. This framework empowers cloud servers to provision services in a fine-grained and adaptive manner, catering to geographically distributed video crowd sourcers. It also addresses service migration between cloud instances with diverse lease prices and considers the impact of location on streaming quality. To assess the real-world performance of the proposed strategy, a prototype system has been implemented and evaluated on PlanetLab, Amazon, and Microsoft clouds. Crowd-sourced video streaming is distinguished by the geographical dispersion and heterogeneity of sources globally, presenting challenges in large-scale transcoding and systematic optimization. These challenges include (1) the selection of video quality to maximize viewer satisfaction and (2) the allocation of computing resources to minimize operational costs. To tackle this, He et al. introduced a generic framework for crowd-sourced live streaming involving heterogeneous broadcasters and viewers, utilizing powerful and elastic cloud computing services [21]. While existing studies often focus on crowd-sourced video streaming within smaller regions, Bilal et al. propose a system, named Cloud-Based Multiview Crowd-sourced Streaming (CMVCS), that collects individual video streams (views) captured at the same event by multiple participants and combines them. This approach allows viewers to experience the event from various angles, optimizing the video representation based on available bandwidth for each viewer. The paper emphasizes resource allocation in CMVCS systems, aiming to maximize overall viewer satisfaction by optimizing resources allocated to the view transcoding process under computational and bandwidth constraints [22].

As mentioned above, there are various existing studies on crowd-sourced video streaming, but all of them use a centralized framework with cloud servers, and to the authors' knowledge, there is no attempt to realize crowd-sourced video streaming with a peer-to-peer architecture such as the one proposed in this paper.

3. Proposed System

3.1. Possible Usage Scenarios

Before describing the details of the proposed system, this subsection presents three scenarios in which the proposed system can be used effectively.

Suppose a sports team in your city wins a major tournament, and a parade or other social event is held to celebrate. The event would be attended by people from all over the city, and each participant would record a video of the event on his or her smartphone, possibly sharing the recorded video privately with family and friends. Our proposed system will allow such video streams to be shared among event participants on a larger scale and in a more timely manner. This will increase the satisfaction of event participants by allowing them to see the event from perspectives other than their own.

Since our system assumes that each mobile device is connected to the Internet through a cellular network, the venue of the target event is not limited by the geographical proximity between devices. Thus, for example, a user participating in a high school festival in one city and a user participating in a high school festival in another city can share each other's viewpoints over the network. This makes it possible, for example, for students at a high school with a small number of students to experience a similar variety of festivals as students at a high school with a large number of students.

Our system can be used not only to enjoy a single event from multiple perspectives or to seamlessly participate in multiple events but also to capture the movement of a particular subject with multiple cameras in fixed positions. While live coverage of Olympic marathon games is typically provided by mobile cameras mounted on cars and motorcycles, our system can be used to produce live video focused on specific runners from video streams captured by spectators along the route. While our current implementation supports only simple video composition with multiple streams displayed simultaneously, in principle it is possible to perform more advanced composition, such as serial-parallel composition of video streams. Combined with appropriate video authoring tools, more advanced use of user-generated video streams is expected.

3.2. Overview of the Proposed System

Figure 2 depicts the overall configuration of the proposed system. The system is composed of multiple worker nodes and a dedicated coordinator. At its core, the system relies on the worker network, a peer-to-peer (P2P) network of worker nodes responsible for storing video chunks as the primary source of video streaming. Detailed information about the worker network is provided in Section 4. The principal role of the coordinator is to oversee the Distributed Hash Table (DHT) utilized by the worker network and to facilitate the efficient transfer of video chunks within the worker network.

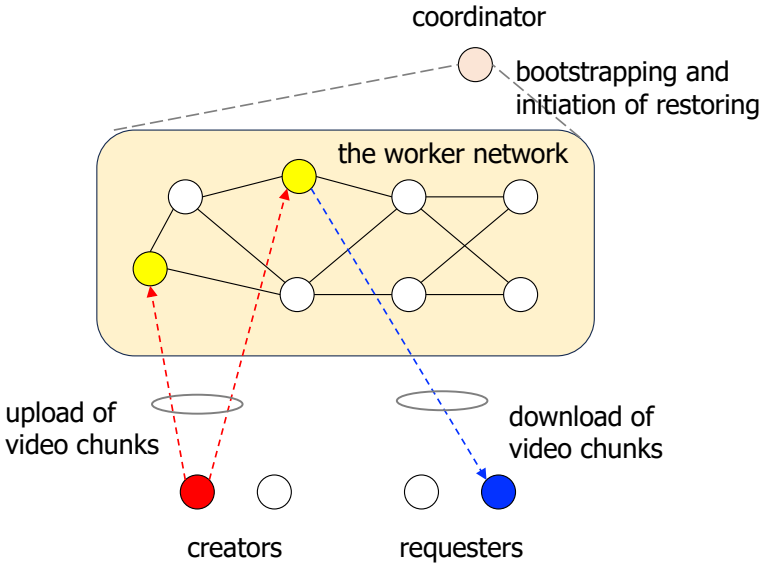

Figure 2. Overview of the proposed system. Creators (red circle) transcode the given mp4 file into a sequence of HLS chunks and upload it to worker nodes in the worker network. Video chunks stored in a worker node (yellow circle) are delivered to the requesters (blue circle) upon request.

All client devices involved in the system, including laptop PCs and tablets, function as worker nodes. These nodes encompass creators, who act as producers of video chunks, and requestors.

3.3. Implementation Details

The system is implemented using Golang version 1.12.3 and comprises several software modules. Among them, the Video Stream Processing (VSP) module is responsible for video streaming, while the peer-to-peer (P2P) module maintains and manages the worker network. The VSP module leverages the advanced features of FFmpeg (version 2021-12-23, git-60ead5cd68-essential_build), ensuring state-of-the-art video processing capabilities within the system. In the current implementation, we merely conduct the synthesis of multiple video streams and sound synthesis is not conducted by default. Further details about the VSP module are provided in Section 5.

The P2P module is developed using the Golang-based IPFS client kubo. Each node within the worker network is equipped with an integrated kubo client, establishing a private IPFS-based network. Additionally, the module facilitates direct information exchange between endpoints through the HTTP protocol, primarily utilized during specific operations, such as the initialization and termination of IPFS services.

Within the system, file resources, such as video chunks, are identified as Tasks, each assigned a unique Task Identifier (TID) generated using the SHA-256 algorithm. Worker nodes are effectively managed using a specialized address format named 'multiaddr,' encapsulating essential information such as the workerID (generated by the SHA-256 algorithm), the protocol in use, and the port number. The module employs a cryptographic algorithm to generate a private key for each workerID, ensuring the verification of authenticity when workers attempt to establish a P2P connection.

In the current implementation, certain functionalities enabling interaction with the public IPFS network are intentionally disabled to enhance security. Specifically, the IPFS Remote Procedure Call (RPC) and IPFS gateway features, common in typical IPFS deployments, are deactivated. This proactive measure strengthens the system's security framework, preserving the private network's insulation from potential vulnerabilities associated with public network interfaces.

4. Worker Network

4.1. Overview

In the proposed system, user-generated videos, which are converted into the HLS format by the creators, are distributed and cached on the worker network in a distributed manner. Video chunks cached on the network are requested by users who wish to view the video, and workers holding the video chunks transfer them to the requester. The main role of the worker network is to perform the above process efficiently without delay, and for this purpose, it effectively utilizes mechanisms provided by the IPFS, such as DHTs and Bitswap. In this section, we provide an overview of kademlia-DHT, a concrete DHT employed in the proposed method, followed by a description of the Pub/sub model used to distribute video chunks and a resource swap mechanism responsible for the discovery and transfer of cached chunks.

4.2. kad-DHT as an Underlying Overlay

The worker network incorporates a DHT known as Kademlia (kad-DHT) within its logical structure. Kademlia is a peer-to-peer network designed for storing key-value pairs, where the key is typically a hash value and the value is a record representing the network location of the resource corresponding to the hash value. The address space of kad-DHT spans from 0 to $2^{256} - 1$, with peers and records mapped to this range using SHA-256. Each record is stored on the nearest peer in the address space, and the connection between adjacent peers in the kad-DHT is established through an overlay that mimics the skip list,

enabling a quick lookup in $O(\log N)$ hops, where N represents the number of peers in the kad-DHT. The way of organizing the logical structure of kad-DHT is illustrated in Figure 3.

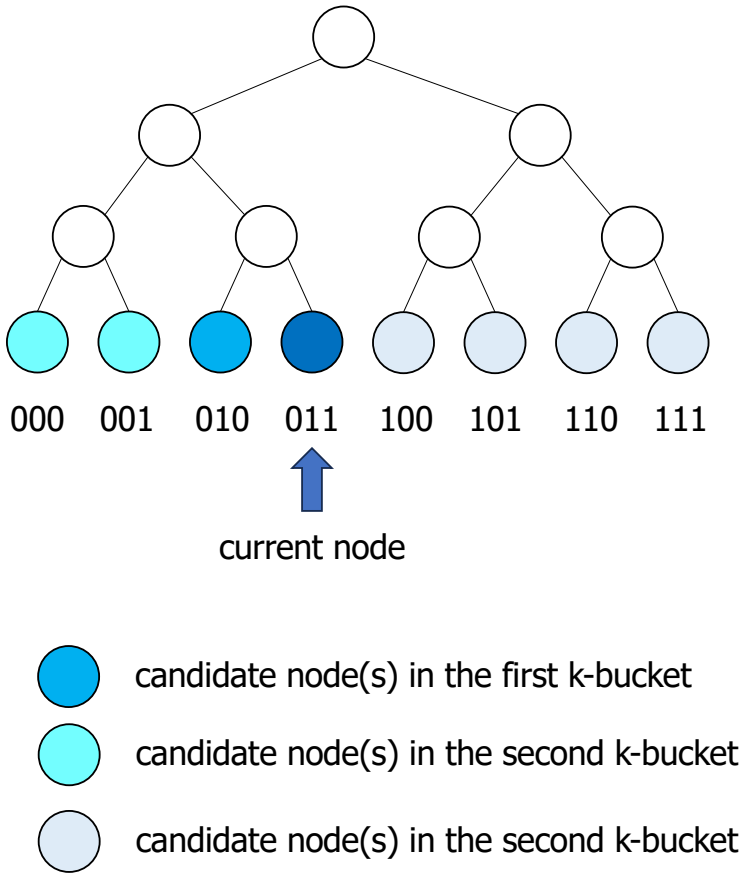

Figure 3. Kademlia binary tree, in which the current node 011 is painted dark blue. In this figure, node 011 keeps its contact (i.e., list of neighbors) in the form of three k-buckets, where the first k-bucket contains nodes painted blue, the second k-bucket contains nodes painted sky blue, and the third k-bucket contains nodes painted light blue, where the Hamming distance of a node to the current node determines the k-bucket that can contain the node.

kad-DHT provides support for the joining and leaving of peers. In the proposed system, a designated node called the coordinator acts as a bootstrap server, ensuring correct maintenance of the overall network structure as long as each worker appropriately executes the join/leave procedure of the kad-DHT. However, in real network environments, workers joining the kad-DHT may leave unexpectedly, temporarily turn off the device, or delete cached chunks or critical information in the routing table. Such a discrepancy can result in the coordinator's understanding of the network structure not aligning with the actual state, leading to malfunctioning in the join procedure and chunk retrievals.

To address this issue, the system delegates the coordinator with the authority to restore the entire network structure in a centralized manner. The restoration process aims to reconstruct the complete DHT utilizing only the current online workers in the system. The worker network is structured to enable the coordinator to receive reports on the success or miss of queries executed on the DHT. In this context, a query miss indicates that the resource requested by a worker cannot be found within a predefined timeout period, typically due to node departure and/or cache erasure.

The coordinator triggers the restore procedure when the percentage of query misses since the last restore surpasses 15%, signaling a potential degradation in system performance. Upon completion of the restoration, the coordinator relinquishes its centralized

management role, reverting to its normal role as a bootstrap server. This proactive approach ensures the continuous efficiency and robustness of the network structure.

4.3. Information Exchange Using Peer-to-Peer Module

In the presented system, the maintenance and management of the worker network are orchestrated in a decentralized manner through a dedicated P2P module. This module encompasses two information exchange mechanisms—DHT and the Publish/Subscribe (Pub/sub) model—and a resource transfer mechanism named resource swap. While this subsection delves into a detailed explanation of the information exchange mechanisms, the subsequent subsection provides an overview of the resource swap mechanism. All these mechanisms are implemented using libraries provided by the IPFS.

Within this system, resources, such as video chunks, are conceptualized as tasks, each assigned a unique Task ID (TID). The DHT functions as a means for workers to locate specific tasks. Precisely, a query to the DHT with the TID as a key retrieves a record containing the address of the worker node responsible for that task. The Pub/sub model, another mechanism facilitated by the P2P module, is employed for push-based information exchange. In this model, a message associated with a specific topic is disseminated to surrounding worker nodes in the overlay upon publication. The message is then forwarded and subscribed to exclusively by worker nodes that have subscribed to that particular topic. This Pub/sub model proves highly effective for distributing commands like the service restart directive. The coordinator leverages this mechanism to guide and direct other worker nodes, fostering rational and effective communication within the system.

4.4. Resource Swap Service for Transferring Segment Files

The system employs a resource swap service, developed based on the Bitswap protocol, a fundamental component of the IPFS, for distributing and retrieving video segments. The flow of the Bitswap protocol is shown in Figure 4. In the resource swap service, file transfer involves dividing each task into smaller units known as blocks. Each block is assigned a unique content identifier (CID) and organized into a Directed Acyclic Graph (DAG) structure. The TID is then derived from the string's hash value obtained by concatenating all the CIDs in the task.

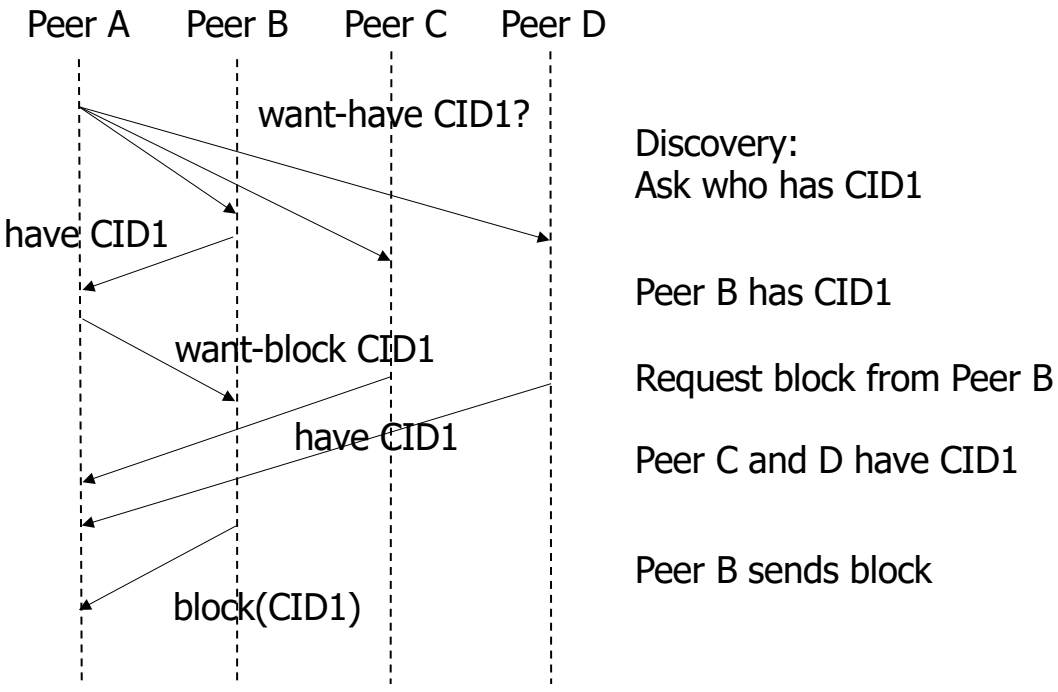

Figure 4. The flow of Bitswap protocol.

The primary functions of the resource swap service encompass task acquisition and distribution. During task acquisition, the service efficiently obtains video segments requested by clients from available worker nodes in the network. The discovery of a worker node owning a specific file begins with the issuance of a "want-have" request to all known workers. This request includes the CID of the root block of the DAG, and the recipient responds with a "have" message if it owns the root block, entering the subsequent transfer process, or a "don't-have" message if it does not possess the block. The Bitswap service aggregates responses, creating a map indicating which worker owns each block. If the file holder cannot be found through the "want-have" request, the DHT is queried to identify the worker holding the file.

Once the worker holding the desired block is identified, the requester issues a "want-block" request, and the identified worker transfers (the sequence of) requested blocks. Data transfer occurs through socket connections between worker nodes, with multi-addr format used for addressing communication partners. To maximize bandwidth utilization, each worker node can concurrently create up to three socket threads, and each block is transmitted and received as an independent data unit in a predefined socket order. In the event of a connection drop during file transfer, the system responds by issuing a "want list" of CIDs for the remaining blocks. This list is distributed throughout the system, prompting requests to retransmit outstanding blocks.

5. Video Stream Processing Module

5.1. Overview of HTTP Live Streaming (HLS)

The proposed system adopts HTTP Live Streaming (HLS) as the video format for optimal compatibility and efficient video delivery. HLS operates by transcoding an input MP4 file into an index file (.m3u8) and multiple segment files (.ts). Each .ts file contains video data, while the .m3u8 file specifies the order, playback time, and delivery format of each .ts file. Acting as a playlist for client-side playback, the .m3u8 file ensures seamless rendering of segment files with different resolutions and bit rates in consecutive order, facilitated by the requirement of consecutive time stamps.

5.2. Generation of Video Chunks

Figure 5 shows how a requester downloads video chunks obtained by bundling multiple chunks over the worker network. As is shown in the figure, in the envisioned system, user-generated video streams undergo a sequential process involving recording, transcoding, delivery, bundling, and playback. The transcoding step involves converting an MP4 file originating from a video camera or similar device into a sequence of segment files. In compressed video formats like MP4, a video stream is encoded into a series of picture groups (GOPs), each comprising multiple frames and commencing with a key frame referred to as an I-frame.

During transcoding, each segment file encapsulates an entire GOP, potentially causing variations in chunk length compared to the specified duration. Even slight discrepancies, measured in milliseconds, can lead to blank spaces when bundling multiple streams into a single stream—an undesirable outcome. To overcome this issue, the proposed system employs a strategy where the creator inserts keyframes at regular intervals before initiating transcoding, ensuring uniform chunk lengths. The keyframe insertion is achieved by resetting the IBP frame sequences using the libx264 library, maintaining the integrity of the video content while addressing the specified problem.

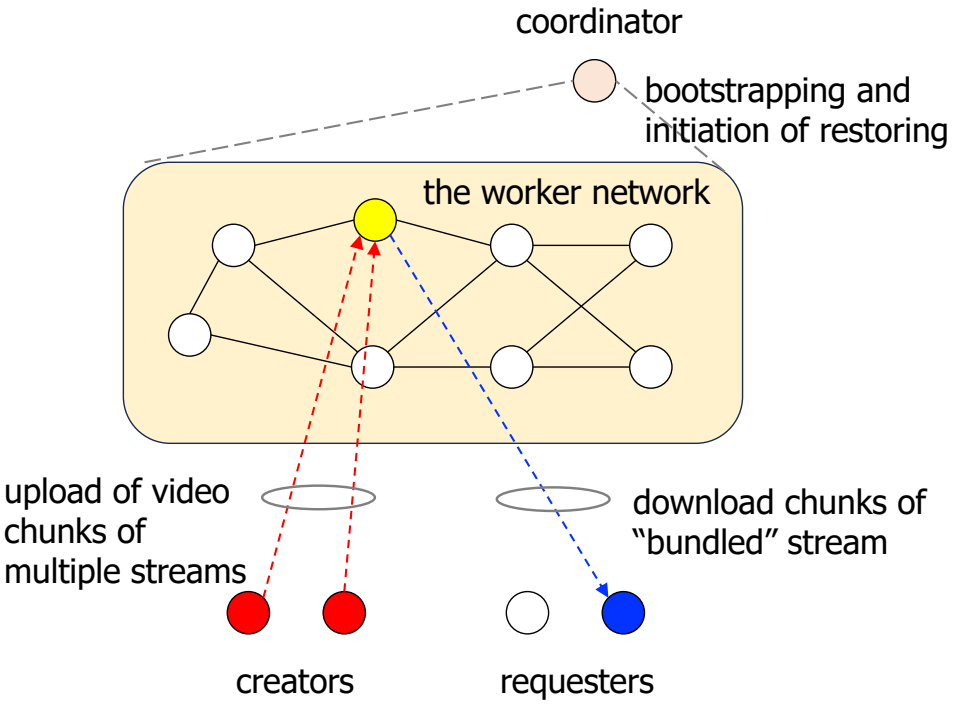

Figure 5. Sharing of bundled video streams. Creators (red circles) transcode the given mp4 file into a sequence of HLS chunks and upload it to the worker nodes in the worker network. The worker node storing two video streams (yellow circle) performs the bundling of corresponding video chunks and delivers it to the requester (blue circle) upon request.

5.3. Distribution of Generated Chunks

The video chunks generated through MP4 transcoding undergo distribution to workers and are automatically cached by the receiving workers. The destinations for chunk distribution are governed by the IDs assigned to the chunks. The implemented prototype system employs specific rules for ID assignment to enhance the efficiency of the subsequent bundling process:

1. Video chunks originating from the same MP4 stream typically share the same ID. However, when the MP4 stream exceeds a predefined length, such as 20 minutes, a new ID is generated and assigned to chunks each time it surpasses this specific duration. This approach prevents an excessive number of chunks from being cached on the same worker, promoting a balanced distribution of chunks from the same video camera across the worker network.
2. Video chunks that have the potential to be bundled together are assigned the same ID. In cases where the bundling potential cannot be predicted in advance, these chunks are permitted to be pre-cached on any worker. Upon receiving a request for a bundled stream, relevant chunks are then transferred between corresponding workers. It is crucial to note that in such scenarios, the download time for the bundled stream may increase due to the transfer time.

5.4. Bundling Multiple Video Streams

In this paper, bundling refers to the process of aligning multiple videos either horizontally or vertically to create a unified video. For instance, horizontally bundling two 24 s videos of 480 × 720 resolution results in a new 24 s video with 960 × 720 resolution. Figure 6 shows screenshots of bundled video streams (used in the experiments described in Section 6). As shown in the figure, the system supports various alignments such as 2 × 1 (1 × 2), 2 × 2, and 2 × 3 (3 × 2). While some deviations can be adjusted using FFmpeg's

pad filter, it is essential to be mindful that a significant difference in the bit rate or resolution of aligned videos may compromise the naturalness of the resulting video.

In the proposed system, the bundling process is applied directly to segment files, resulting in the generation of new .ts and .m3u8 files, where generated .ts files inherit the timestamp of the input files. Consequently, when a requester seeks a bundled video stream, a fresh *.m3u8 file is created, enabling the requester to download and play solely the newly generated segment files, where the responsibility of maintaining the new .m3u8 file lies with the worker who conducted the bundling. The hash value of the newly generated .ts file is annotated as a comment in the .m3u8 file. This resulting .m3u8 file serves both as a playlist for the bundled video stream and as a torrent file in BitTorrent.

(**a**) 1×2 (**b**) 2×1

(**c**) 2×2 (**d**) 2×3

Figure 6. Screenshot of bundled video streams used in the experiments.

The bundling process leverages FFmpeg libraries effectively. Specifically, the hstack (or vstack) operation combines frames with minimal frame loss, and the outcome is then rendered onto a larger canvas using the pad filter. The amerge filter is employed to combine audio tracks into a multi-channel stream, which is subsequently converted to stereo.

6. Experiments

6.1. Setup

We conducted a series of experiments to evaluate the performance of the proposed system. In the experiments, we used six videos, each featuring three distinct resolution types (360p, 1080p, and 4K) representing diverse scenarios: 360p for low-end smartphones, 1080p as a prevalent resolution in contemporary video applications, and 4K as a high resolution that preserves the original quality of camera-captured video streams. The specific parameters and characteristics of these six videos, including the number of chunks generated from them, are summarized in Table 1.

Table 1. Six different video streams used in the experiment and their specifications.

	left.mp4	**right.mp4**	**1080left.mp4**	**1080right.mp4**	**4k30left.mp4**	**4k30right.mp4**
Resolution	640 × 360	640 × 360	1920 × 1080	1920 × 1080	3840 × 2160	3840 ×2160
Duration (s)	31	30	36	35	30	30
FPS	30	30	30	30	30	30
Number of chunks	7	7	8	8	7	7
Decoder	MPEG-4 AAC,H.264	H.264	H.264	H.264	MPEG-4 AAC,HEVC	MPEG-4 AAC,HEVC
DataSize (KB)	3042	1990	48,737	59,035	93,842	112,563

The performance evaluation was performed by simulating the target P2P network within a virtual environment hosted on a machine, which was created by deploying clients within Docker containers. The Docker host, running on Windows 11 23H2, is a desktop machine equipped with an AMD 5600X CPU and 16 GB DDR4 RAM. Each Docker container was restricted to a maximum of 3 CPU cores and 6 GB RAM. The Docker image used in the experiment is based on the official Golang image and includes installed FFmpeg and kubo clients. Docker containers were configured to emulate individual worker nodes in the network, with network throughput assigned to each node reflecting the average throughput of a 4G network, approximately 8 Gbps.

For control experiments, we prepared a separate benchmark environment of the client/server (C/S) model described as follows. This model utilized an Azure cloud server in Tokyo as the delivery server, configured with Standard B1s specifications (1vCPU, 1GiB RAM, 30GiB SSD) and a bandwidth of approximately 328 Mbps. The video processing server, deployed on a desktop machine with specifications identical to the Docker host (AMD 5600X CPU, 16 GB DDR4 RAM), processed the video streams. Multiple devices running MacOS or Windows were employed for client-side operations, interacting with the server. Importantly, under this configuration, latency between the video processing server and the content delivery server was negligibly small.

In the subsequent experiments, the entire video creation–processing–storing–delivery process was iterated, and the time consumed by each module was recorded in both the simulation and benchmark environments in Figures 7 and 8

6.2. Time Required for Video Processing

Initially, two video streams, left and right, were generated by two creators, and a sequence of video generation, storage, and bundling operations was conducted by varying the size of the worker network to 3, 8, 12, and 50. In each configuration, all worker nodes participated in storing and bundling video chunks in a distributed manner, with each worker executing a two-times-one bundling process for each pair of received chunks.

As depicted in Table 1, the number of chunks generated from each test video is consistently seven or eight, highlighting that an eight-worker setup proves to be the most efficient, with each worker tasked with storing and bundling a pair of chunks. In contrast, a configuration with 3 workers represents the minimum number required, while setups with 12 and 50 workers are deemed redundant.

The results are presented in Figure 9. The values in this table denote the maximum time needed to transcode an MP4 file to an HLS stream at the creator level and the maximum time needed to generate a new video stream by bundling multiple chunks at a worker node, with time units expressed in seconds. Transcoding time remains constant regardless of the number of workers, while bundling time decreases as the number of workers increases from three to eight.

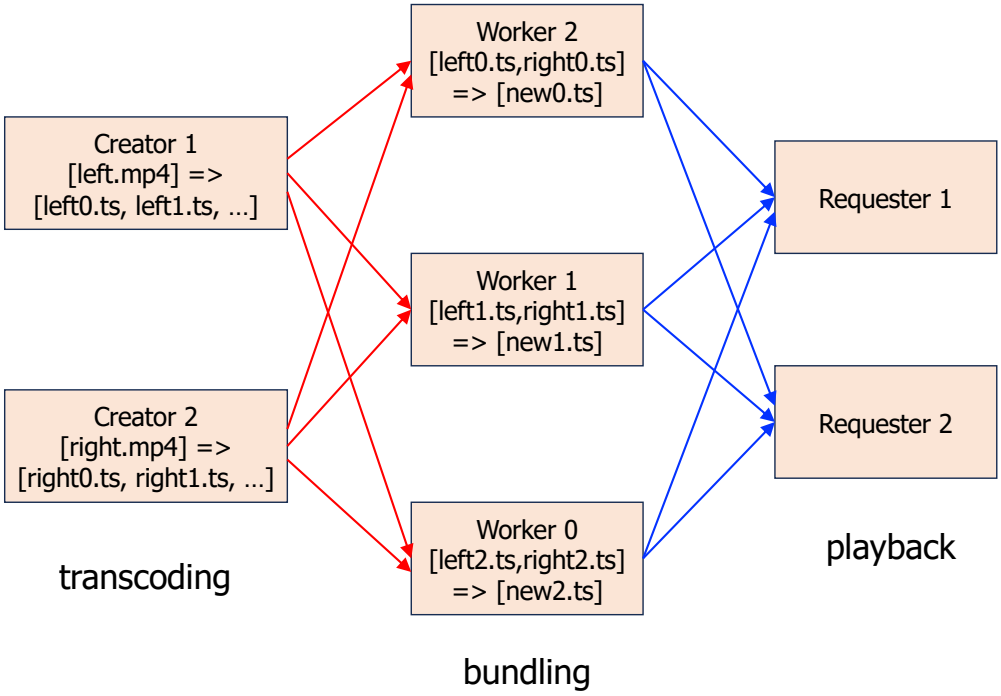

Figure 7. Flow of data processed in the simulation environment. Each creator transcodes the captured video stream into a sequence of HLS chunks and distributes them to workers, who then bundle the received chunk pairs and forward them to the requester.

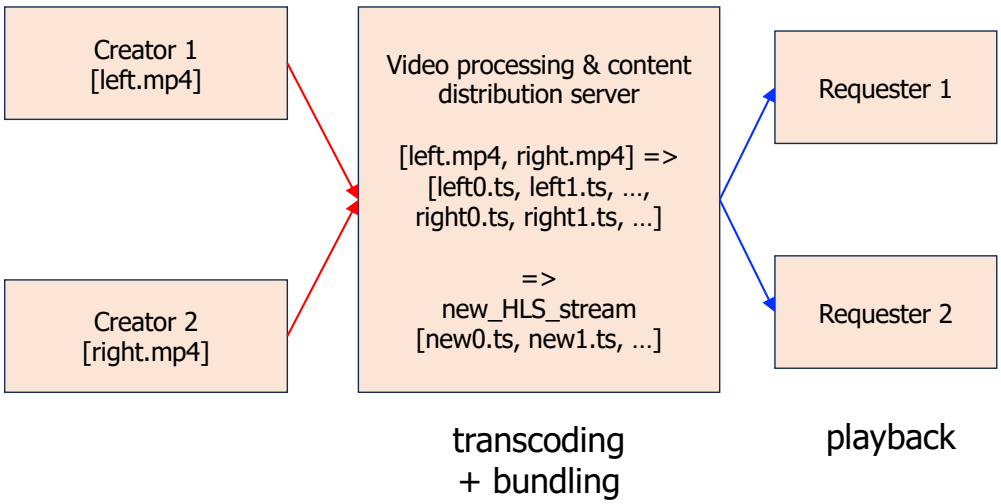

Figure 8. Data flow in the benchmark environment. Each creator transmits a captured video stream to the server. The server executes the transcoding and bundling of received streams and then forwards the resulting stream to the requesters.

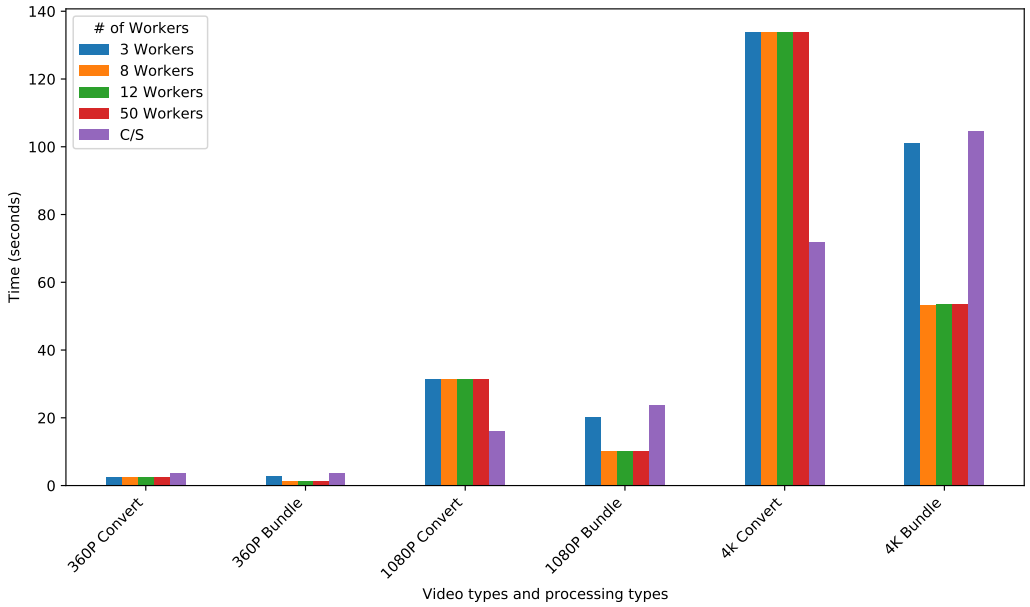

Figure 9. Time required for video processing (seconds).

The following is a comparison with the transcoding times on the C/S model shown in the rightmost column of the table. When the video resolution is low (360p), the time is halved because the transcoding of two videos on the server is equally distributed between two creators. At higher resolutions, the difference in server specifications becomes more pronounced, and the time required for each creator to transcode one video is twice the time required to transcode two videos on the server, reversing the time ratio.

In summary, it was observed that with a sufficient number of worker nodes, the time required for video processing decreases significantly. However, this trend stalls when the number of workers reaches or exceeds the total number of video chunks. The system also shows a marked advantage over the conventional server model, especially at low resolutions. This increase in performance is due to the ease with which worker nodes can process low-resolution video and the parallel processing advantages offered by the distributed nature of the system. As the resolution increased, the percentage of time spent on bundles in the system decreased, from 34% at 360P resolution to 28.5% at 4K resolution. This suggests that as video resolution increases, the impact of bundling on overall processing time decreases.

6.3. Time Required for Video Transmissions

In C/S-based video streaming systems, a significant portion of the total processing time is dedicated to the upload of video streams recorded by creators to the server and the subsequent download of the transcoded video stream. Similarly, in our proposed system, a substantial portion of processing time is allocated to video chunk transmission, which is further divided into the upload of HLS chunks from creators to worker nodes and the time taken by requesters to download stored chunks from the worker nodes. Consequently, the experiments presented below focus on measuring chunk transmission time, with variations in both the number of worker nodes involved in video chunk storage and bundling and the number of requesters seeking bundled video streams.

The corresponding results are detailed in Figures 10–12. Analysis of the table reveals that the performance of the video transmission phase is influenced by the ratio of worker nodes to requesters. Specifically, when the number of worker nodes exceeds the number of requesters, the transmission time is constrained by the download bandwidth of the requester. Conversely, when the number of worker nodes is less than the number of requesters, the bottleneck shifts to the upload bandwidth of the worker nodes.

Similarly, in C/S-based systems, the transmission bottleneck is contingent on the ratio of the number of requesters to the server's bandwidth. If the number of requesters surpasses the server's bandwidth, the server becomes the bottleneck; conversely, if the server's bandwidth is not fully utilized, the limitation shifts to the client side. Crucially, in scenarios where the server bandwidth is fully utilized (e.g., with 50 requesters), our proposed system, with an appropriate number of workers, exhibits a considerable advantage over the traditional C/S model. This advantage is particularly pronounced in configurations where the requesters also serve as worker nodes, highlighting the efficiency of the system in high-demand scenarios.

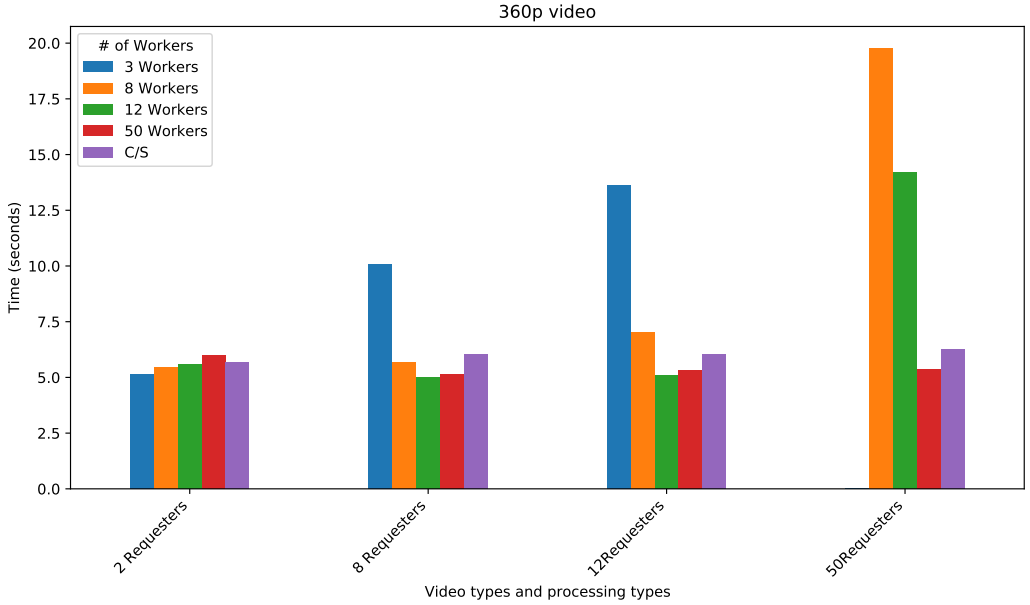

Figure 10. Time required for transmission of 360p video (seconds). The transmission time of 3 workers for 50 requesters is hidden because it exceeds 45 s.

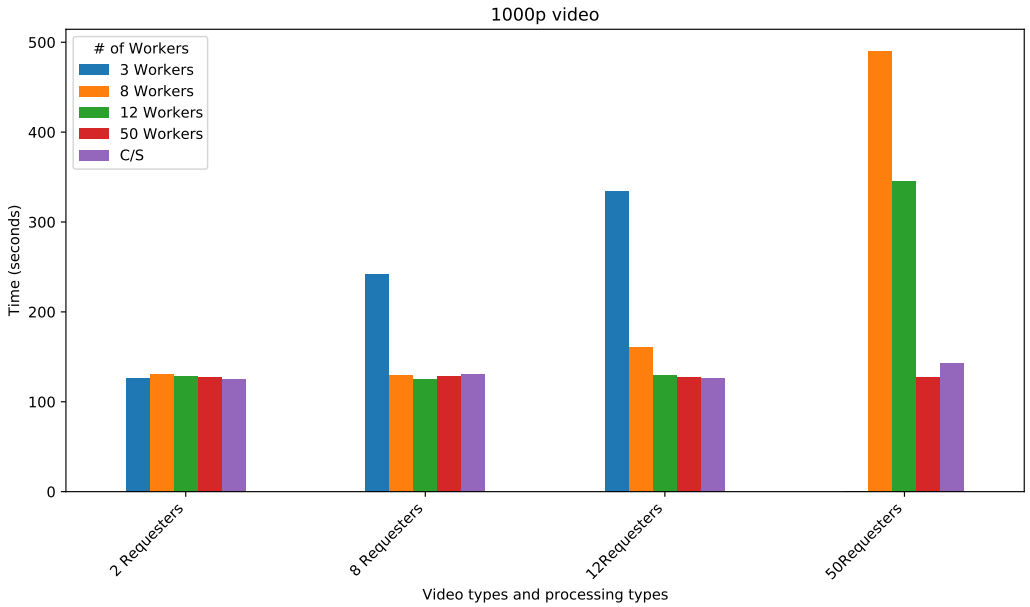

Figure 11. Time required for transmission of 1000p video (seconds). The transmission time of 3 workers for 50 requesters is hidden because it exceeds 200 s.

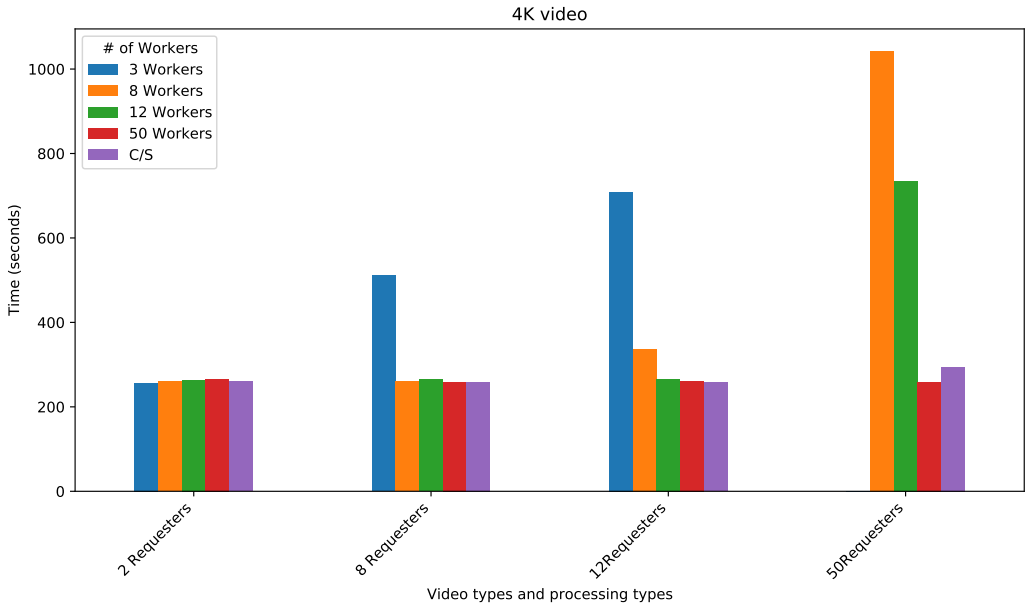

Figure 12. Time required for transmission of 4K video (seconds). The transmission time of 3 workers for 50 requesters is hidden because it exceeds 2500 s.

6.4. Summary

Our empirical findings strongly underscore the superiority of the proposed system compared to C/S-based systems. Notably, it exhibits a significant advantage, particularly evident in the case of low-resolution video streams such as 360P and 1080P, and this advantage becomes more pronounced as the stream length increases. Furthermore, it demonstrates a clear advantage in scenarios characterized by a substantial volume of viewing requests. Additionally, it achieves performance levels comparable to high-performance servers, even in scenarios lacking high-resolution videos or featuring a limited number of requests. Remarkably, this performance is attained using devices with lower processing capabilities and limited bandwidth. This outcome highlights the robustness and effectiveness of the proposed system, particularly in open environments with diverse users employing resource-constrained devices. It also underscores the sophisticated design principles embedded in the proposed methodology.

Concerned with the security and reliability, the system offers enhanced security compared to existing open P2P systems. Specifically, it prevents malicious or suspicious nodes from joining by implementing appropriate policies on the boot server. Regarding file security, the entire system relies on the SHA-256 hash function. Consequently, all files containing HLS segments are hashed before transmission, facilitating easy detection of file tampering. In terms of transport security, the system supports TLS for secure communication between nodes, safeguarding data in transit against eavesdropping and man-in-the-middle attacks. Regarding reliability and fault tolerance, most files are protected from loss thanks to the backup mechanism. Additionally, the network reboot conducted by the coordinator ensures high system reliability and effectively avoids data transfers over unreliable links.

7. Concluding Remarks

In this paper, we introduce a decentralized architecture for crowd-sourced video streaming that eliminates the need for servers. We conduct a performance evaluation using a prototype system implemented in the Go language. The proposed system employs HTTP video streaming for content delivery and effectively utilizes information-sharing and exchange mechanisms, such as Kademlia DHT and BitSwap provided by IPFS, to store HLS segments obtained through transcoding on worker nodes. Our experimental findings reveal a strong correlation between the proposed system's performance and the size of the worker

network. Notably, when the number of workers is sufficiently large compared to the number of video streams to be stored, our system outperforms a client/server (C/S)-type system implemented for comparison using AWS. Future work involves porting the system to mobile devices, such as smartphones and Raspberry Pi, and empirically assessing its effectiveness in real-world events with more than 50 worker nodes. Additionally, we address the crucial issue of implementing decentralized multi-view stream editing functionality.

Author Contributions: Conceptualization, S.F.; methodology, J.G. and S.F.; software, J.G.; validation, S.F.; resources, S.F.; data curation, J.G.; writing—original draft preparation, J.G.; visualization, S.F.; supervision, S.F.; project administration, S.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The code for the prototype system described in this paper, the code for the experiments conducted on it, and the raw data of the experimental results can be obtained from the following URL: https://github.com/gitgjr/Cherry (accessed on 1 February 2024).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Li, Z.; Zhu, X.; Gahm, J.; Pan, R.; Hu, H.; Begen, A.C.; Oran, D. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE J. Sel. Areas Commun.* **2014**, *32*, 719–733. [CrossRef]
2. Mok, R.K.P.; Chan, E.W.W.; Chang, R.K.C. Measuring the quality of experience of HTTP video streaming. In Proceedings of the 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops, Dublin, Ireland, 23–27 May 2011.
3. Mok, R.K.P.; Chan, E.W.W.; Luo, X.; Chang, R.K.C. Inferring the QoE of HTTP video streaming from user-viewing activities. In Proceedings of the First ACM SIGCOMM workshop on Measurements, Toronto, ON, Canada, 19 August 2011; pp. 31–36.
4. Sodagar, I. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimed.* **2011**, *18*, 62–67. [CrossRef]
5. Zambelli, A. IIS smooth streaming technical overview. *Microsoft Corp.* **2009**, *3*, 40.
6. CMAF. Available online: https://developer.apple.com/documentation/http_live_streaming/about_the_common_media_application_format_with_http_live_streaming_hls (accessed on 1 February 2024).
7. Hsu, T.; Tung, Y. A Social-Aware P2P Video Transmission Strategy for Multimedia IoT Devices. *IEEE Access* **2020**, *8*, 95574–95584. [CrossRef]
8. Jurca, D.; Chakareski, J.; Wagner, J.; Frossard, P. Enabling adaptive video streaming in P2P systems. *IEEE Commun. Mag.* **2007**, *45*, 108–114. [CrossRef]
9. Liu, Y.; Guo, Y.; Liang, C. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Netw. Appl.* **2008**, *1*, 18–28. [CrossRef]
10. Pal, K.; Govil, M.C.; Ahmed, M. Priority-based scheduling scheme for live video streaming in peer-to-peer network. *Multimed. Tools Appl.* **2018**, *77*, 24427–24457. [CrossRef]
11. Ramzan, N.; Park, H.; Izquierdo, E. Video streaming over P2P networks: Challenges and opportunities. *Signal Process. Image Commun.* **2012**, *27*, 401–411. [CrossRef]
12. Thampi, S.M. A Review on P2P Video Streaming. *arXiv* **2013**, arXiv:1304.1235.
13. Roverso, R.; El-Ansary, S.; Haridi, S. Smoothcache: Http-live streaming goes peer-to-peer. In Proceedings of the 11th International IFIP TC 6 Networking Conference (NETWORKING 2012), Part II 11, Prague, Czech Republic, 21–25 May 2012; Springer: Berlin/Heidelberg, Germany, 2012; pp. 29–43.
14. Nacakli, S.; Tekalp, A.M. Controlling P2P-CDN live streaming services at SDN-enabled multi-access edge datacenters. *IEEE Trans. Multimed.* **2020**, *23*, 3805–3816. [CrossRef]
15. Ma, Z.; Roubia, S.; Giroire, F.; Urvoy-Keller, G. When Locality is not enough: Boosting Peer Selection of Hybrid CDN-P2P Live Streaming Systems using Machine Learning. In Proceedings of the Network Traffic Measurement and Analysis Conference (IFIP TMA), Virtual, 13–15 September 2021.
16. Ha, T.T.T.; Kim, J.; Nam, J. Design and Deployment of Low-Delay Hybrid CDN–P2P Architecture for Live Video Streaming Over the Web. *Wirel. Pers. Commun.* **2017**, *94*, 513–525. [CrossRef]
17. Yousef, H.; Feuvre, J.L.; Ageneau, P.-L.; Storelli, A. Enabling adaptive bitrate algorithms in hybrid CDN/P2P networks. In Proceedings of the 11th ACM Multimedia Systems Conference, Istanbul, Turkey, 8–11 June 2020; pp. 54–65.
18. Farahani, R.; Çetinkaya, E.; Timmerer, C.; Shojafar, M.; Ghanbari, M.; Hellwagner, H. ALIVE: A Latency-and Cost-Aware Hybrid P2P-CDN Framework for Live Video Streaming. *IEEE Trans. Netw. Serv. Manag.* 2023, *early access*.
19. Farahani, R.; Bentaleb, A.; Çetinkaya, E.; Timmerer, C.; Zimmermann, R.; Hellwagner, H. Hybrid P2P-CDN architecture for live video streaming: An online learning approach. In Proceedings of the IEEE Global Communications Conference (GLOBECOM 2022), Rio de Janeiro, Brazil, 4–8 December 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 1911–1917.

20. Chen, F.; Zhang, C.; Wang, F.; Liu, J. Crowdsourced live streaming over the cloud. In Proceedings of the Conference on Computer Communications (INFOCOM 2015), Kowloon, Hong Kong, 26 April–1 May 2015; IEEE: Piscataway, NJ, USA,2015; pp. 2524–2532.
21. He, Q.; Liu, J.; Wang, C.; Li, B. Coping with heterogeneous video contributors and viewers in crowdsourced live streaming: A cloud-based approach. *IEEE Trans. Multimed.* **2016**, *18*, 916–928. [CrossRef]
22. Bilal, K.; Erbad, A.; Hefeeda, M. Crowdsourced multi-view live video streaming using cloud computing. *IEEE Access* **2017**, *5*, 12635–12647. [CrossRef]