Is IPFS Ready for Decentralized Video Streaming?

ChengHao Ryan Yang*
Northeastern University
USA
yang.chengha@northeastern.edu

Zhengyu Wu*
Stony Brook University
USA
zhenwu@cs.stonybrook.edu

Santiago Vargas
Stony Brook University
USA
savargas@cs.stonybrook.edu

Aruna Balasubramanian
Stony Brook University
USA
arunab@cs.stonybrook.edu

ABSTRACT

InterPlanetary File System (IPFS) is a peer-to-peer protocol for decentralized content storage and retrieval. The IPFS platform has the potential to help users evade censorship and avoid a central point of failure. IPFS is seeing increasing adoption for distributing various kinds of files, including video. However, the performance of video streaming on IPFS has not been well-studied. We conduct a measurement study with over 28,000 videos hosted on the IPFS network and find that video streaming experiences high stall rates due to relatively high Round Trip Times (RTT). Further, videos are encoded using a single static quality, because of which streaming cannot adapt to different network conditions.

A natural approach is to use adaptive bitrate (ABR) algorithms for streaming, which encode videos in multiple qualities and streams according to the throughput available. However, traditional ABR algorithms perform poorly on IPFS because the throughput cannot be estimated correctly. The main problem is that video segments can be retrieved from multiple sources, making it difficult to estimate the throughput. To overcome this issue, we have designed Telescope, an IPFS-aware ABR system. We conduct experiments on the IPFS network, where IPFS video providers are geographically distributed across the globe. Our results show that Telescope significantly improves the Quality of Experience (QoE) of videos, for a diverse set of network and cache conditions, compared to traditional ABR.

CCS CONCEPTS

• **Networks → Peer-to-peer networks**.

KEYWORDS

IPFS, ABR, Distributed Storage System, Video Streaming

ACM Reference Format:
ChengHao Ryan Yang, Zhengyu Wu, Santiago Vargas, and Aruna Balasubramanian. 2023. Is IPFS Ready for Decentralized Video Streaming?. In *Proceedings of the ACM Web Conference 2023 (WWW '23), May 1–5, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3543507.3583404

*Both authors contributed equally to this research.

WWW '23, May 1–5, 2023, Austin, TX, USA

ACM ISBN 978-1-4503-9416-1/23/04…$15.00
https://doi.org/10.1145/3543507.3583404

1 INTRODUCTION

InterPlanetary File System (IPFS) [12] is an emerging peer-to-peer hypermedia protocol for storing and retrieving content. As an alternative to the Hypertext Transfer Protocol (HTTP) protocol, IPFS builds on the principles of peer-to-peer networking and content-based addressing to create a decentralized, distributed, and trustless data storage and delivery network.

A recent large-scale study of the IPFS network [36] shows the significant popularity of the platform. IPFS has a presence in 152 countries and has over 3 million clients. Major web browsers, Brave and Opera, provide native support for IPFS as an alternative to HTTP [11, 13], and other browsers including Firefox, Chrome, and Edge have extensions for IPFS [24]. IPFS has been used to support decentralized Web applications, social network platforms, and content search [36]. Netflix uses IPFS in their infrastructure to transfer docker images [17].

In effect, IPFS provides an alternative platform to make the Web more decentralized, improve availability, and help circumvent censorship. P2P systems [10, 15, 22] have been designed with similar objectives but these systems have seen a decline in adoption. Instead, platforms such as IPFS have a revised incentive model, better integration with browsers, and an effective content discovery mechanism, making them a viable alternative.

Given these advantages, one natural use case for the IPFS architecture is decentralized video streaming. Video is the most popular type of traffic on the Internet [33]. In fact, there is a video streaming platform over IPFS called DTube [4] which sees over 2.1 million monthly unique users [3]. However, DTube is not truly decentralized. The nodes that store the videos in DTube belong to a private cluster under DTube's control. Our goal is to study the performance of video streaming over IPFS when the content is truly decentralized.

To this end, we conduct a measurement study of over 28,000 videos hosted in the IPFS network across the globe. We find that almost all videos are at most 3 IPFS hops away. One IPFS hop refers to a node contacting its closest set of peers for content; each of these peers in-turn contacts their closest peers at each hop until the content is retrieved. However, since IPFS nodes are overlays, the RTT to retrieve content is relatively high, with a median of 60ms.

The result is that video streaming over IPFS experiences high stall rates. When streaming video at 8 Mbps bandwidth, which is the median bandwidth seen in the IPFS nodes in US and EU [36], 90% of the videos experience a stall. Even when experimenting with an over-provisioned network with a bandwidth of 25 Mbps, 50% of the videos experience stall. Part of the issue is that IPFS videos are encoded at a single quality; the quality of the video cannot be adapted based on the network condition.

One approach to solving the single quality problem is to use the popular Adaptive Bitrate Streaming (ABR) algorithm [8]. ABR encodes videos under different qualities and streams the best quality segment depending on the estimated throughput. ABR is used by most popular video streaming services on the Internet including Youtube and Netflix. However, our analysis shows that traditional ABR algorithms perform poorly in the IPFS setup. The main problem is that ABR cannot estimate the throughput accurately because video segments can be fetched from multiple providers in the IPFS network, each with a different network condition.

In this work, we present Telescope, an IPFS-aware ABR system, to significantly improve video streaming QoE (Quality of Experience) over IPFS. Telescope works by accounting for the throughput discrepancies at the ABR client and reverse-engineers the bandwidth requirement. Traditional ABR uses a manifest file to track the bandwidth requirement of each video segment. Telescope modifies the bandwidth requirement such that it reflects the real bandwidth needed to fetch the segment from IPFS, based on the location of the provider. Telescope also takes into account how far off the client's estimated throughput is. This modification forces the ABR algorithm to choose the right video quality segment despite the incorrect throughput estimation and does not require changes to the client ABR or the IPFS infrastructure.

We evaluate Telescope by streaming videos from geographically distributed providers over the IPFS network. We set the client bandwidth based on bandwidths seen by IPFS clients across different global regions [36]. In all cases, Telescope outperforms traditional ABR algorithm and vanilla IPFS streaming.

In the median bandwidth case, Telescope outperforms ABR by 123% when the cache hit is 60% and by 54% when the cache hit is 80%. These cache hit numbers are obtained from the global IPFS cache performance [36]. Our results are consistent across different ABR algorithms and client locations. Even in cases where there is no cache, but cache builds up over time (what we call progressive cache), IPFS outperforms traditional ABR by 34% in terms of QoE and 80% in terms of stall rates.

In effect, Telescope presents a new potential for video streaming that preserves the advantages of distributed video systems while streaming videos with high QoE.

2 BACKGROUND AND RELATED WORK

2.1 Background: IPFS Retrieval

A detailed overview of IPFS network and its components can be found in Appendix A.1.1. In this section, we describe how content is retrieved from IPFS (Figure 1). A user requests content from the IPFS network by requesting a CID or a unique content identifier. For ease-of-use, IPFS allows the user request to be in HTTP; in this case, the request is sent to the IPFS gateway which converts the HTTP request to an IPFS request.

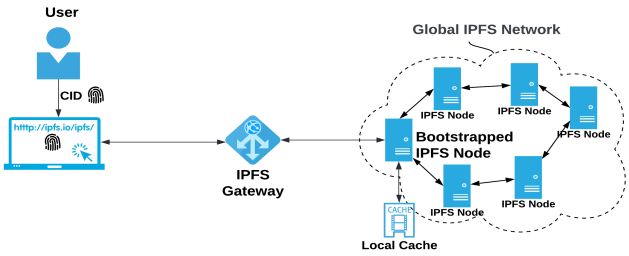

Figure 1: An overview of how content is retrieved over IPFS via the IPFS Gateway.

Every gateway is bootstrapped with an IPFS node, that we call *Bootstrapped IPFS node*, which is part of the Global IPFS network. Upon receiving a CID request from the gateway, the bootstrapped IPFS node checks its local cache. If the file corresponding to the CID is not in the cache, the gateway node first asks its connected peers using a protocol called Bitswap. If none of the peers have the content, the gateway node calculates the k closest peer nodes and requests these peers; each peer in-turn asks their k closest peers and so on. This procedure is repeated until the peer with the content is found.

The peer that has the content is also called a provider. A CID/file can be stored with multiple providers in the IPFS network. The bitswap protocol decides which provider to fetch the content from. If multiple providers have the content, the bitswap protocol prioritizes the original content provider first. The next priority is given to the content provider that has provided content frequently in the past.

2.2 Related Work: IPFS Video Streaming

A recent paper on IPFS [36], authored by the developers of IPFS, describe the architecture of IPFS. The paper also discusses the geographical distribution of IPFS nodes, IPFS bandwidths, the effect of caching, and content retrieval performance. IPFS has also been studied in the context of applications such as secure file sharing [23] and social networking [38].

Our focus is to study the decentralized video streaming performance over IPFS. Peer-to-peer (P2P) platforms have a similar goal as IPFS, and there has been considerable work on P2P video streaming [31, 37]. There are also popular commercial P2P video streaming services such as PPLive [6] and PPStream [1]. However, PPLive, and P2P video streaming in general, is seeing decreasing adoption. PPStream is still popular, but the focus of the service has shifted to centralized streaming.

In terms of video streaming over IPFS, DTube [4] is a popular open-source video platform that is built on top of IPFS and Steem Blockchain [21]. A study over DTube [18] shows the setup and connection performance of DTube and compares it with the popular video hosting site YouTube. However, in DTube, the video content is stored and streamed from a centralized server and is not truly decentralized.

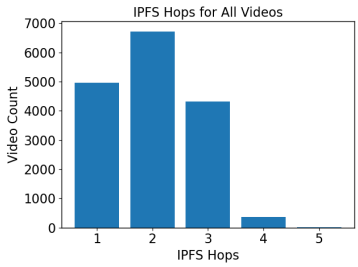

Figure 2: Video IPFS hops.

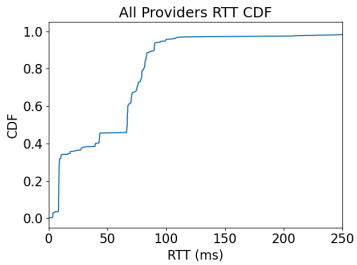

Figure 3: Video provider RTTs.

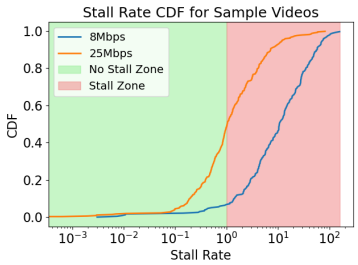

Figure 4: Videos stall rates under different network conditions.

3 IPFS VIDEO CHARACTERISTICS

In this section, we characterize IPFS video content and streaming challenges.

Measurement Methodology: We use IPFS-search, a search engine designed to search IPFS content to collect video data for our study. IPFS-search gets its data by deploying sniffers in the global IPFS network and retrieving any new CIDs advertised/distributed in the global network.

In this measurement, we use the API to search for all video formats including mp4, avi, webm, and mpeg. The collecting machine is located at U.S East Coast. We present the data collected from September 1st to October 4th of 2022. We collect 39,259 unique video CIDs from which we retrieve 28,652 videos; the remaining videos could not be retrieved.

3.1 Characteristics of IPFS Videos

The most common video resolution is 512 x 512, accounting for 36% of the videos. We examined the content of these videos and they are game videos such as Minecraft. IPFS is likely being used as a storage service. Of the remaining, nearly 30% of videos use common streaming resolutions with 1080P, 720P, and 480P being the 3rd, 5th, and 6th most popular resolutions, respectively. The remaining videos have non-standard resolutions. Less than 1% of the videos have a resolution of 2K and 4K.

In terms of video duration, 90% of the videos are less than 4 minutes while 5% of the videos are longer than 20 minutes. We further examine the duration of videos with common resolutions (480P, 720P, 1080P). In these video formats, 72% of the videos are less than 4 minutes and 10% are longer than 20 minutes.

3.2 IPFS Network Characteristics

Figure 2 shows that 30% of videos can be retrieved over one IPFS hop. Almost all the videos can be retrieved within 3 IPFS hops. However, the number of IP hops required to retrieve the videos that have 10 or more hops are 34.7% of the videos (not shown in figure). This is reflected in the RTTs. Figure 3 shows that the median RTT is 60ms. For 10% of the videos, the RTTs are over 100ms. In fact, this large RTT is the cause of video streaming problems as we see next.

3.3 IPFS Video Performance

To study the performance of IPFS videos under different network conditions and realistic stream settings, we randomly chose 500 videos from all 480P, 720P, 1080P videos we collected. We select videos of different lengths according to their proportion in the total video dataset.

We look at two bandwidths to retrieve the video: 8Mbps and 25Mbps. 8Mbps represents the median bandwidth seen by IPFS nodes [36] in America and EU and 25 Mbps represents an over-provisioned network. Figure 4 shows the stall rates when loading these videos. Under 8 Mbps bandwidth conditions, 90% of the videos are stalled and under 25 Mbps connection 50% of the videos are stalled. For the videos that are stalled, the median stall rate for the 8 Mbps connection is as high as 10 (this means that the time to play the video will be 9 times the original video duration). For the 25 Mbps connection, the median stall rate for videos that stall is 5.

There are two key reasons for the stalls. The first is that videos are encoded using a single quality and video streaming cannot adapt the quality according to the network conditions. The second problem is that the RTTs are high, resulting in high stalls. Of course, caching partially solves the problem with high RTTs, if the videos are cached close to the user. However, as we show in §5, caching does not completely eliminate poor video streaming performance. We next describe Telescope, a system that significantly improves video streaming over IPFS.

4 TELESCOPE DESIGN AND IMPLEMENTATION

Adaptive Bitrate Streaming (ABR) is the most commonly used algorithm for streaming videos and works by adapting the video quality based on the network condition. ABR would solve the problem in IPFS where all videos are streamed using a single quality irrespective of the network condition. However, because in IPFS the video segment can be streamed from multiple locations (cached at the gateway or fetched from different providers), vanilla ABR does not work well, as we describe next. Instead, we develop Telescope, an IPFS-aware ABR system. Importantly, Telescope does not require changes to the client ABR or the IPFS infrastructure.

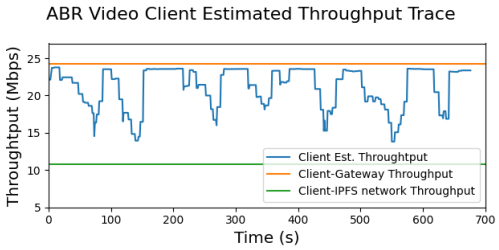

Figure 5: Trace of the estimated throughput at the ABR client while streaming a 10-minute DASH video. The throughput estimation at the ABR client fluctuates depending on whether there is a cache hit or a miss.

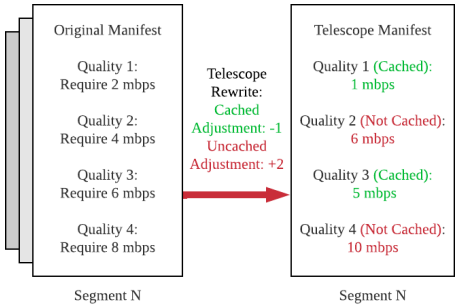

Figure 6: Example of Telescope rewriting media manifest file. The left shows the original manifest file. The right shows how the manifest file is manipulated, providing a positive adjustment to uncached segments and a negative adjustment to the cached segments.

4.1 ABR in IPFS

ABR: Adaptive bitrate algorithms or ABR [8] (and its associated protocol, DASH [14]) works by breaking down a video into multiple short video segments. Each segment is encoded under different qualities. ABR creates a manifest/MPD file to map the qualities to the bandwidth required to download the segment within a certain time. Figure 6 (left side marked 'original manifest') shows an example manifest file where segment N is encoded in 4 different qualities; the higher the quality, the more bandwidth is required to download it within the given time.

Common ABR algorithms [7, 16, 30] at their core work as follows: the client estimates the available throughput and the algorithm chooses the highest quality segment that can be streamed within the given time, given the throughput.

Problems in using ABR for IPFS: In traditional ABR, the segments are downloaded from a single server. However, IPFS retrieves videos in different ways (see Figure 1): 1) if the segment/block is cached, it is fetched from the gateway IPFS node. 2) if not, the video segments can be fetched from different IPFS providers.

The result is that ABR's estimated throughput is incorrect. Figure 5 demonstrates this issue. The ABR client estimates throughput based on the time taken to download a segment where some segments are cached while others are fetched from the IPFS provider(s). The estimated throughput therefore hovers between the client-gateway throughput, which is the throughput when the segment is cached, and client-IPFS network throughput, which is the throughput when the segment is not cached. This problem gets even more complex when the segments are downloaded from multiple IPFS providers.

Takeaway: The inaccurate throughput estimation at the client ABR causes two problems: 1) When ABR overestimates the throughput, it erroneously thinks that it can download a segment of high quality. But if this segment is not cached, it will have to be fetched from the provider which will take much longer than expected, resulting in stalls. 2) when ABR underestimates the throughput, it erroneously chooses to download a lower quality cached segment even if it could have downloaded a higher quality segment. In either case, the result is a drop in QoE.

4.2 Telescope

Telescope works by modifying the manifest file to account for the inaccurate throughput estimation at the client. We first describe the two main components of Telescope and then describe the end-to-end system. Figure 7 shows how Telescope interacts with the IPFS infrastructure. Telescope works between the client and the gateway or can be co-located with the gateway.

4.2.1 Modifying the manifest file. It is difficult to change how ABR estimates client throughput since this algorithm is used universally. Instead, Telescope works by changing the manifest file that enumerates the bandwidth requirement of each segment and is sent periodically to the client.

We provide an example to show how the manifest file is modified: Assume that as download progresses, a client's ABR estimates the network throughput to be 3 Mbps based on its segment download times. However, this is an incorrect throughput estimation and, similar to Figure 5, 3 Mbps is between the throughput of cached segments (client-gateway) and the throughput of uncached segments (client-IPFS provider). For simplicity, assume that uncached segments are downloaded from a single IPFS provider.

Telescope first estimates the throughput. In this example, say, the client-gateway and client-IPFS provider throughputs are 4 Mbps and 1 Mbps respectively. We explain how the throughput is estimated next. This means, the client underestimates the throughput for cached segments by 1 Mbps and overestimates throughput for uncached segments by 2 Mbps. To account for this, Telescope manipulates the manifest file and reduces the bandwidth requirement by 1 Mbps for cached segments, and increases the bandwidth requirement by 2 Mbps for uncached segments. Figure 6 shows this example. Telescope learns which segments are cached and which are uncached using an IPFS gateway API.

In a nutshell, if the client overestimates available throughput, Telescope *increases* the bandwidth requirement in the manifest file of all uncached segments by roughly the amount of throughput overestimation. We call this an uncached adjustment. For cached segments, Telescope *reduces* the bandwidth requirement of cached segments by roughly the underestimation amount. We call this cached adjustment.

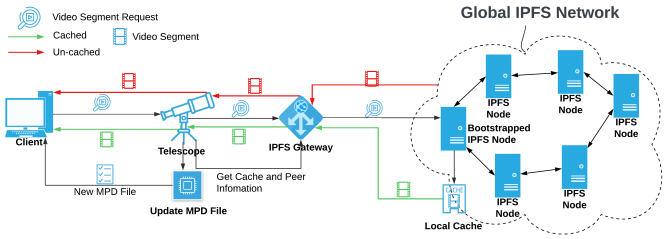

Figure 7: Telescope overview

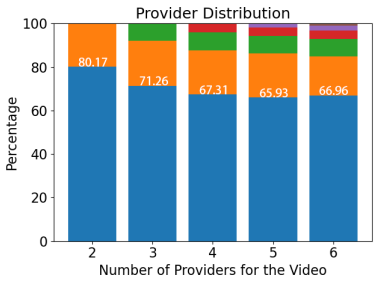

Figure 8: Distribution of providers when retrieving a video. The label represents the percentage of blocks retrieved from the dominant provider.

To modify the manifest file, Telescope has to (i) guess the client's estimated throughput and (ii) estimate the throughput for cached and uncached segments. The most challenging part here is the uncached segments. Recall that a segment itself is divided into different IPFS blocks and each block can be fetched from different providers.

4.2.2 Estimating throughput. Telescope works on the following observation: the fraction of blocks downloaded from each provider is roughly a constant for all segments. Figure 8 shows the fraction of times videos were downloaded from a single versus multiple providers in the experiment described in §3. When multiple providers are involved, there tends to be a single dominant provider that contributes to a majority of the data. Further, the fraction of providers does not change for subsequent segments. This is by design, since the IPFS bitswap protocol favors providers that have been historically stable.

Let P1, P2, .. be the list of past providers so far. Telescope can get the name of the provider from each segment using an added IPFS API. The throughput to fetch an uncached segment to the gateway is then estimated as

$$T_{P_1} \times R_{P_1} + T_{P_2} \times R_{P_2} + \ldots + T_{P_n} \times R_{P_n} \quad (1)$$

where T_{P_x} is the throughput of each provider (obtained via IPFS API), and R_{P_x} is the fraction of blocks that are historically fetched from each provider. To estimate the throughput between the client and the IPFS network, we add the above equation to the gateway throughput (described next).

The throughput between the gateway and the client or the gateway throughput is estimated based on the start and end times of the segments downloaded from the gateway. The gateway throughput is also the throughput of cached segments.

Now the next piece involves reverse-engineering the client's estimated throughput. Recall from Figure 5 that the client's estimated throughput depends on the number of cached and uncached segments it downloads. If the client downloads mostly cached segments, then its throughput is closer to client-gateway throughput, and if the client downloads mostly uncached segments, then the throughput is closer to client-provider throughput. Telescope keeps track of the number of cached and uncached segments fetched by the client (using the IPFS native API) and estimates the client throughput as the weighted sum of the cached and uncached segment throughputs

4.2.3 Telescope end-to-end system. At the start of the experiment, Telescope connects the client and the gateway and retrieves videos using traditional ABR. For uncached segments, the gateway contacts the IPFS network; if the segment is cached at the bootstrapped IPFS node, the gateway gets the cached segment directly from the bootstrapped node.

As the download progresses, Telescope collects information about throughputs and cached and uncached segments. At this time, Telescope modifies the manifest file with the new bandwidth requirements. When the manifest is set to dynamic mode, the client periodically requests the new manifest file, and Telescope sends this modified manifest.

Telescope adjusts the bandwidth requirement based on the client's estimated throughput and the adjustment based on the cache status of the segment. Let T_C, T_G, and T_N be the client's estimated throughput, client-gateway throughput, and client-IPFS network throughput, respectively. We described the estimation of these throughputs in the previous section. Then

$$\begin{cases} \text{cached adjustment} = T_C - T_G & \text{if segment is } \textbf{cached} \\ \text{uncached adjustment} = T_C - T_N & \text{if segment is } \textbf{uncached} \end{cases} \quad (2)$$

The cached or uncached adjustment is used to manipulate the manifest file. For cached segments, the adjustment is negative, reducing the bandwidth requirement, and for uncached segments, the adjustment is positive increasing the bandwidth requirement.

4.3 Implementation

Telescope is implemented as a proxy between the IPFS gateway and the client (see Figure 7). We use the Go default reverse proxy handler [20] and the Gin Web Framework [19] to implement Telescope. Telescope captures a unique client id to identify individual video playbacks and stores throughput histories of the gateway and the global IPFS network for each connected client. To distinguish between cached and uncached segments, Telescope uses the IPFS HTTP RPC API, which is available by default [5]. To obtain the providers' information, we added a new API to IPFS which returns the providers' information for a given downloaded segment. The API uses simple filtering of the IPFS debug output. We did not observe any impact on performance when using this new API. The Telescope implementation is available at this GitHub link: https://github.com/SBUNetSys/Telescope.

5 EVALUATION

We evaluate Telescope's streaming performance and compare it with vanilla IPFS and traditional ABR streaming. We conduct the experiment by deploying video files on IPFS nodes across the world and retrieving them from multiple clients. We need two more pieces of information to evaluate Telescope in a real-world environment: the typical bandwidth experienced by IPFS clients and cache hit rates that real clients see. Recent work on IPFS [36] characterizes both of these using data from millions of clients and thousands of IPFS instances. We use values from this study to ground our evaluation.

5.1 Setup

The client runs Chrome v89 with DASH.js and loads videos over Telescope. We run an IPFS gateway instance and bootstrapped IPFS node instance as an open-source IPFS daemon. We deploy 5 IPFS providers (all connected to the global IPFS network) via Google Cloud Platform (GCP) in Asia East, Europe Central, Middle East, South America, and US West. Our default client is in Northeast US. The RTT from the client to these vantage points are 191 ms, 102 ms, 132 ms, 125 ms, and 68 ms respectively. The client retrieves the video from the providers over the real IPFS network. We also experiment with clients in other locations. All experiments are repeated 5 times and we present the average.

We experimented with other IPFS hosting services including Pinata and Web3Storage but did not see a difference in performance, so we omit the results.

5.1.1 Video Player. We use the dash.js reference client v3.2.2 [16], an open-source JavaScript DASH video player. We encoded a 10-minute 4K video Big Buck Bunny [32] for all the experiments. The video is encoded into 11 video quality levels or bitrate levels, 1 Mbps to 25 Mbps (4K) in intervals of 2.5 Mbps. The encoder is FFmpeg and uses 5-second segments. This setup is similar to previous work and industry recommendations [14, 27].

5.1.2 Metrics. We present our performance result in terms of Quality of Experience (QoE). To calculate QoE, we measured three common video metrics: average segment quality, stall rate, and quality variation (or smoothness). All metrics are critical to user experience and have been used previously to evaluate streaming performance [9].

The average segment quality of downloaded segments ranges from 1 to 11 (11 bitrate levels). The average segment quality for a video is averaged over the quality of all downloaded video segments (127 segments in the case of the reference video we employ in our experiments). The stall rate is computed as follows:

$$Stall\ Rate = \frac{total\ playback\ time - video\ length}{video\ length} \tag{3}$$

The quality variation represents how video quality is changing over the course of the playback and is computed similarly to previous work [39].

$$Quality\ Variation = \frac{1}{N}\sum_{n=1}^{N} |q_n - q_{n-1}| \tag{4}$$

To calculate QoE, we follow previous works [28] and take into account average quality, stall rates, and quality variation as follows:

$$QoE = \frac{\sum_{n=1}^{N} q_n}{N} - \mu \times Stall\ Rate - Quality\ Variation \tag{5}$$

for a video with N segments. q_n is the quality number and we set $\mu = 6$. While previous work [39] has discounted the maximum video quality for a 1-second stall (ie. $\mu = 11$) under 1080p video, we show results for $\mu = 6$ to balance between quality and stall rates. We confirm that our QoE findings are only exacerbated with $\mu = 11$ (omitted for brevity).

5.1.3 Baselines. We compare Telescope with the following baselines:

- **IPFS-Vanilla:** The videos are encoded only using a single quality and this same quality is retrieved irrespective of the network condition. We experiment with three different static qualities: 1080P (regular), 2K (high), and 4K quality (max).
- **ABR:** In this case, the throughput-based ABR algorithm is used to choose the best quality video to fetch. This alternative is used as our default baseline.
- **ABR-BOLA and ABR-DYN:** We compare Telescope to two other ABR algorithms. ABR-BOLA [35] uses the buffer occupancy to decide which quality video to send next and ABR-DYN [2] is a dynamic ABR algorithm that switches between both Bola and throughput modes to decide which quality video to fetch.

5.1.4 Cache Levels. Cache plays a large part in IPFS performance by considerably improving retrieval time. Cache hits can happen in two places: at the IPFS gateway or at the Bootstrapped IPFS node. Globally, it has been shown that the average gateway cache hit ratio is 60% [36]. Gateway + Bootstrapped IPFS node cache hit rate increases to 80% [36]. Accordingly, we consider the following caching strategies:

- **Global cache**: We evaluate under the global cache hit rates of 60% and 80%. We use two strategies for caching: 1) Random cache where the segments to cache are chosen at random keeping in mind the hit rate, and 2) Non-random cache where we hand-select the cache to be evenly distributed across the video segments.
- **Progressive cache**: Progressive cache assumes that there is no cache in the beginning and the cache builds as videos are fetched over and over. This experiment evaluates the performance of a user who is fetching videos from a new gateway whose cache has to build up.

5.1.5 Network Conditions. Our goal is to experiment with network bandwidths experienced by IPFS clients in the real world, as reported by the recent large scale study [36]. Accordingly, we choose four bandwidths: 4 Mbps, 8 Mbps, 13 Mbps, and 25 Mbps. The median bandwidth experienced by a user in the US and Europe is 8 Mbps, so we choose this as our default. 4 and 13 Mbps bandwidths are the lower and higher bandwidth range experienced by users in other parts of the world (more details in §A.2). 25 Mbps is an over-provisioned bandwidth sufficient for streaming 4K video. We use Linux tc [26] to set the bandwidths.

We do not vary RTT because the distributed nature of IPFS already accounts for RTT variations.

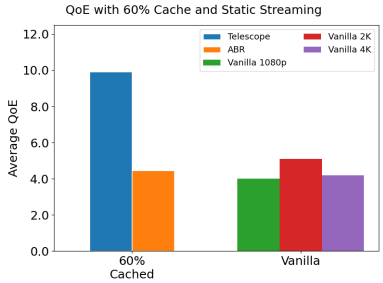

Figure 9: Comparing QoE of Telescope with traditional ABR and vanilla IPFS retrieval.

5.2 Telescope QoE

In this section, we evaluate Telescope under the average cache levels of 60% and 80% seen in global IPFS networks. We first compare Telescope with traditional ABR and vanilla IPFS retrieval for a 60% non-random cache (Figure 9). Recall that IPFS streams videos under static qualities, so we look at streaming at three different qualities. For all qualities, Telescope outperforms IPFS streaming by 94% in terms of QoE. For the rest of the evaluation, we omit results from vanilla IPFS (with static quality) because it performs poorly under all settings.

When compared to traditional ABR, Figure 10 shows that Telescope outperforms at both cache levels and for both random and non-random cache strategies. At 60% cache hits, Telescope outperforms ABR by 123%; at 80%, the difference is 54%.

QoE alone does not tell the whole story, so we next look at the stall rates, qualities, and quality variation that are used to estimate QoE (see §5.1.2). Figure 11 shows that Telescope reduces stall rate significantly across all cache levels. Under 60% cache level, Telescope reduces stalls by 95% compared to traditional ABR. This is because, when traditional ABR does not estimate throughput accurately, it may fetch high quality videos that are not cached, resulting in higher stalls. Figure 12 shows Telescope also fetches segments with higher average quality. Finally, Figure 13 shows that Telescope has a more consistent experience by having lower variation between each video segment (lower quality variation is better).

5.3 Varying Network Conditions

Telescope performs well under the median bandwidth of 8 Mbps as shown in the previous section. Here we present the results of other bandwidth conditions. We use the 60% non-random cache for these experiments. Figure 14 shows the results. Telescope's QoE consistently outperforms ABR throughout all WAN bandwidth settings while maintaining a low stall rate.

Telescope works particularly well under lower bandwidth settings with a QoE improvement of 287% because the penalty for fetching an uncached segment is extremely high in these settings. As bandwidth increases, traditional ABR performances increases because the penalty to fetch an uncached video segment is lower thus the QoE is higher. But even under 13 Mbps bandwidth, Telescope outperforms QoE by 62%.

Telescope has a lower stall rate across all network conditions. Even in the over-provisioned 25 Mpbs bandwidth condition, Telescope reduces stall rates by 82%. Telescope outperforms ABR for the average quality and quality variance measures as well, but we omit the results for brevity.

5.4 Different ABR Algorithms

Figure 15 compares Telescope with two other ABR algorithms that make streaming decisions using different parameters: BOLA [35] uses buffer occupancy and Dynamic [34] uses both buffer occupancy and throughput estimates. Telescope improves QoE over Dynamic (ABR-DYN) considerably.

However, ABR-BOLA performs reasonably well, even though Telescope improves QoE by 63% under 60% cache and 27% under 80% cache. BOLA is a buffer-based algorithm which means that the inaccurate throughput estimation has less effect on the algorithm.

5.5 Progressive Cache

Next, we evaluate Telescope performance under the progressive cache setting, where we continuously play the video 5 times. Recall that with progressive cache, we assume that there is zero cache to begin with and the cache builds progressively with each video retrieval. Figure 16 shows that Telescope improves QoE by 34% compared to traditional ABR. In addition, Telescope reduces stall rates significantly by 80%. While Telescope outperforms ABR in progressive cache, Telescope has less QoE improvement because Telescope favors the selection of cached segments rather than uncached segments. Therefore, video runs with Telescope will experience low stall rates but may not select higher-quality segments that are uncached.

5.6 Varying client location

Last, to understand how the client's geo-location impacts Telescope, we deploy the video client in Canada and Japan. As shown in Figure 17, the QoE difference between Telescope and ABR is still significant for geographically distributed clients.

6 CONCLUSION

InterPlanetary File System (IPFS) is a fully distributed, open, and secure media storage and retrieval network that uses the concepts of peer-to-peer networking and content-based addressing. Our work aims to understand the performance of IPFS for distributed video streaming. As a first step, we conduct a study with over 28,000 videos hosted across the IPFS global network. We find that videos can experience high degrees of stalling. We explore the use of adaptive bitrate (ABR) algorithms to improve the performance of video streaming over IPFS. However, traditional ABR algorithms cannot accurately estimate available throughput because they are designed for centralized video platforms. This inaccuracy leads to poor performance when using ABR. To address this problem, we present Telescope, an IPFS-aware ABR system that takes into account the inaccurate throughput estimation by ABR clients. Telescope reverse engineers video segment bandwidth requirements, forcing ABR to pick the best quality video segments to stream even with incorrect

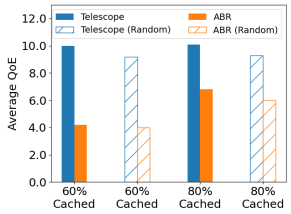

Figure 10: Average QoE across cache settings.

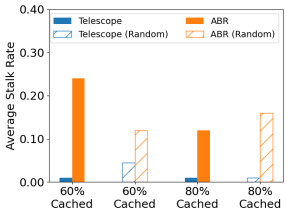

Figure 11: Stall rates across cache settings.

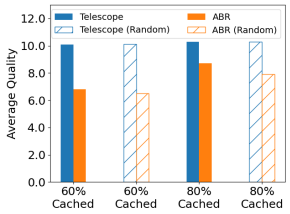

Figure 12: Video quality across cache settings.

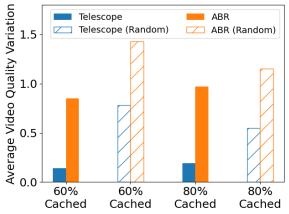

Figure 13: Quality variation across cache settings.

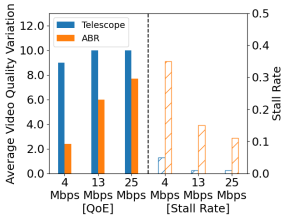

Figure 14: Telescope outperforms ABR across different bandwidth conditions.

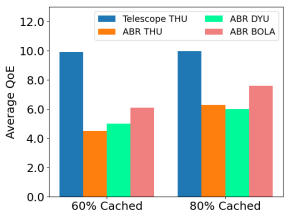

Figure 15: Telescope outperforms different ABR algorithms.

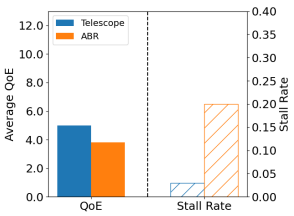

Figure 16: Telescope performs better than ABR in progressive cache.

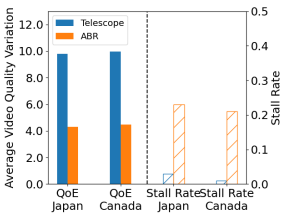

Figure 17: Telescope performs well even with different client locations.

throughput estimation. Importantly, Telescope does not require any change to the ABR client or any other component in the IPFS network, Our evaluation shows that Telescope outperforms the baseline significantly in terms of video Quality of Experience (QoE) under various network and cache conditions.

7 ACKNOWLEDGEMENTS

We thank all the reviewers for their constructive feedback. This work was supported in part by NSF grant CNS-1909356.

REFERENCES

[1] 2015. http://www.ppstream.com/
[2] 2018. https://github.com/Dash-Industry-Forum/dash.js/wiki/ABR-Logic
[3] 2021. DTube coin (DTC) presentation. https://token.d.tube/
[4] 2022. DTube. https://d.tube/
[5] 2022. HTTP RPC API reference. https://docs.ipfs.io/reference/http/api/
[6] 2022. PPTV. https://pptv.com/
[7] Adobe. 2022. Adobe HTTP Dynamic Streaming. http://www.adobe.com/products/hds-dynamic-streaming.html
[8] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. 2011. An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems* (San Jose, CA, USA) *(MMSys '11)*. Association for Computing Machinery, New York, NY, USA, 157–168. https://doi.org/10.1145/1943552.1943574
[9] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. 2018. Oboe: Auto-tuning Video ABR Algorithms to Network Conditions. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. Budapest, Hungary, 44–58.
[10] Fasiha Ashraf, Ateeqa Naseer, and Shaukat Iqbal. 2019. Comparative analysis of unstructured P2P file sharing networks. In *Proceedings of the 2019 3rd International Conference on Information System and Data Mining*. 148–153.
[11] Susie Batt. 2021. Your Files for Keeps Forever with IPFS. https://blogs.opera.com/tips-and-tricks/2021/02/opera-crypto-files-for-keeps-ipfs-unstoppable-domains/.
[12] Juan Benet. 2014. IPFS - Content Addressed, Versioned, P2P File System. *CoRR* abs/1407.3561 (2014). arXiv:1407.3561 http://arxiv.org/abs/1407.3561
[13] Brian Bondy. 2021. IPFS Support in Brave. https://brave.com/ipfs-support/.
[14] Cloudflare. 2021. What is MPEG-DASH? https://www.cloudflare.com/learning/video/what-is-mpeg-dash/.
[15] Bram Cohen. 2003. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, Vol. 6. Berkeley, CA, USA, 68–72.
[16] Dash-Industry-Forum. 2021. Dash.js Github. https://github.com/Dash-Industry-Forum/dash.js.
[17] Edgar Lee Dirk McCormick. 2020. New improvements to IPFS Bitswap for faster container image distribution. https://blog.ipfs.io/2020-02-14-improved-bitswap-for-container-distribution/.
[18] Trinh Viet Doan, Tat Dat Pham, Markus Oberprieler, and Vaibhav Bajpai. 2020. Measuring Decentralized Video Streaming: A Case Study of DTube. In *2020 IFIP Networking Conference, Networking 2020, Paris, France, June 22-26, 2020*. IEEE, 118–126. https://ieeexplore.ieee.org/document/9142739
[19] Gin-Gonic. 2021. Gin Web Framework Github. https://github.com/gin-gonic/gin.
[20] golang.org. 2021. The Go Programming Language. https://golang.org/.
[21] Barbara Guidi, Andrea Michienzi, and Laura Ricci. 2020. Steem Blockchain: Mining the Inner Structure of the Graph. *IEEE Access* 8 (2020), 210251–210266. https://doi.org/10.1109/ACCESS.2020.3038550
[22] Ragib Hasan, Zahid Anwar, William Yurcik, Larry Brumbaugh, and Roy Campbell. 2005. A survey of peer-to-peer storage techniques for distributed file systems. In *International Conference on Information Technology: Coding and Computing (ITCC'05)-Volume II*, Vol. 2. IEEE, 205–213.
[23] Hsiao-Shan Huang, Tian-Sheuan Chang, and Jhih-Yi Wu. 2020. A Secure File Sharing System Based on IPFS and Blockchain. In *Proceedings of the 2020 2nd International Electronics Communication Conference* (Singapore, Singapore) *(IECC 2020)*. Association for Computing Machinery, New York, NY, USA, 96–100. https://doi.org/10.1145/3409934.3409948
[24] Ipfs. 2021. IPFs/ipfs-companion: Browser extension that simplifies access to ipfs resources on the web. https://github.com/ipfs/ipfs-companion
[25] Ipfs. 2022. InterPlanetary Name System (IPNS). https://docs.ipfs.io/concepts/ipns/.
[26] kernel.org. 2021. tc - show / manipulate traffic control settings. https://linux.die.net/man/8/tc.
[27] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (Los Angeles, CA, USA) *(SIGCOMM '17)*. Association for Computing Machinery, New York, NY, USA, 197–210. https://doi.org/10.1145/3098822.3098843
[28] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. 2017. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM 2017, Los Angeles, CA, USA, August 21-25, 2017*. ACM, 197–210. https://doi.org/10.1145/3098822.3098843
[29] Wolfgang Merkle and Frank Stephan. 1996. Trees and Learning. In *Proceedings of the Ninth Annual Conference on Computational Learning Theory, COLT 1996, Desenzano del Garda, Italy, June 28-July 1, 1996*, Avrim Blum and Michael J. Kearns (Eds.). ACM, 270–279. https://doi.org/10.1145/238061.238118

[30] Microsoft. 2022. Microsoft Smooth Streaming. http://www.iis.net/downloads/microsoft/smooth-streaming
[31] Naeem Ramzan, Hyunggon Park, and Ebroul Izquierdo. 2012. Video streaming over P2P networks: Challenges and opportunities. *Signal Processing: Image Communication* 27, 5 (2012), 401–411. https://doi.org/10.1016/j.image.2012.02.004 ADVANCES IN 2D/3D VIDEO STREAMING OVER P2P NETWORKS.
[32] Ton Roosendaal. 2008. Big Buck Bunny. In *ACM SIGGRAPH ASIA 2008 Computer Animation Festival* (Singapore) *(SIGGRAPH Asia '08)*. Association for Computing Machinery, New York, NY, USA, 62. https://doi.org/10.1145/1504271.1504321
[33] Sandvine. 2022. Global internet phenomena. https://www.sandvine.com/phenomena
[34] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. 2018. From Theory to Practice: Improving Bitrate Adaptation in the DASH Reference Player. In *Proceedings of the 9th ACM Multimedia Systems Conference* (Amsterdam, Netherlands) *(MMSys '18)*. Association for Computing Machinery, New York, NY, USA, 123–137. https://doi.org/10.1145/3204949.3204953
[35] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. 2016. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*. 1–9. https://doi.org/10.1109/INFOCOM.2016.7524428
[36] Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. 2022. Design and Evaluation of IPFS: A Storage Layer for the Decentralized Web. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22)*. ACM, Amsterdam, Netherlands. https://doi.org/10.1145/3544216.3544232 ISBN 978-1-4503-9420-8/22/08.
[37] Long Vu, Indranil Gupta, Jin Liang, and Klara Nahrstedt. 2006. Mapping the PPLive network: Studying the impacts of media streaming on P2P overlays. (2006).
[38] Quanqing Xu, Zhiwen Song, Rick Siow Mong Goh, and Yongjun Li. 2018. Building an Ethereum and IPFS-Based Decentralized Social Network System. In *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. 1–6. https://doi.org/10.1109/PADSW.2018.8645058
[39] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. 2015. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, SIGCOMM 2015, London, United Kingdom, August 17-21, 2015*, Steve Uhlig, Olaf Maennel, Brad Karp, and Jitendra Padhye (Eds.). ACM, 325–338. https://doi.org/10.1145/2785956.2787486

A APPENDIX

A.1 InterPlanetary File System Background

A.1.1 IPFS Network. The global IPFS network represents a peer-to-peer network built over IPFS nodes. Each IPFS node runs the IPFS protocol (creating a local IPFS instance) and is connected to each other over the WAN. At initialization, they are assigned a cryptographic hash called *Peer ID*. Each IPFS node knows a subset of all the peers in the global network and will use these peers to search for content.

A.1.2 Storing Files Into IPFS. When the node (or the user) wants to store a file in the IPFS network, it provides this file to its local IPFS instance. The local IPFS instance first breaks the file into blocks that are organized in a Merkle tree [29]. Each block is assigned a unique content identifier (*CID*) which is a consistent hash value based on its content. Each CID along with Peer ID is stored as a record in the local Distributed Hash Table. The local IPFS instance will distribute this record to k closest peers; closest is defined based on the distance between the CID hash and the peer ID. By default IPFS nodes uses $k = 20$.

A.1.3 Retrieving Files from IPFS. When a client wants to retrieve a file from IPFS, the client must provide the CID of the content. The CID can be obtained from a DNS-like service called the InterPlanetary Naming Service (IPNS) [25]. For a given CID, the local IPFS instance uses the hash of the CID to calculate the k closest peer nodes. The instance then asks these peers for the CID's provider record. If the provider record is not located within the k closest peers, each peer in-turn contacts their k closest peers and returns the peer list to the local IPFS instance. The local IPFS instance then asks these peers for the provider record. The process repeats iteratively until a peer with the provider record is found. Finally, the local IPFS instance contacts the provider to retrieve the content.

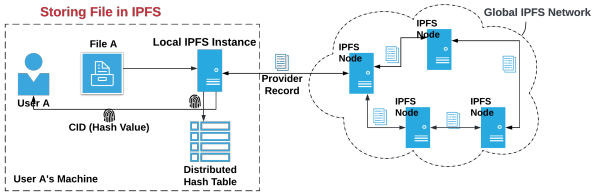

Figure 18: An overview of how content is stored in IPFS.

A.2 Real-World IPFS Throughput

We use representative bandwidths from a recent large-scale study on IFPS [36]. The study looked at both the lookup and download time to fetch a 0.5MB file from IPFS across different global regions. Based on this study, we identify 3 significant bandwidth speeds: 4 Mbps, 8 Mbps, and 13 Mbps. We compute these bandwidths by dividing file size with total download time:

- 4 Mbps is the 25th percentile throughput for Africa, South America, and Asia Pacific regions.
- 8 Mbps is the median throughput for the US, Europe, and Middle East regions.
- 13 Mbps is the 90th percentile throughput for Europe and Middle East regions.

A.3 Ethics

This work does not raise any ethical issues.