Challenges and Protocols in Video Streaming over Networks: A Comparative Study of HTTP ABR, RTSP, RTMP, and CDNs

Vihar Kuruppathukattil
Technology Systems
East Carolina University
kuruppathukattilv19@ecualumni.ecu.edu

***Abstract*—The rapid expansion of video traffic over IP networks has necessitated robust streaming protocols that can operate efficiently across heterogeneous environments. This paper presents a comprehensive comparative evaluation of four key video streaming technologies—HTTP Adaptive Bitrate (DASH), Real-Time Streaming Protocol (RTSP+RTP), Real-Time Messaging Protocol (RTMP/RTMFP), and CDN-integrated peer-assisted networks. Using a 105-GB network trace and emulated testbed covering 4G/5G/Wi-Fi, we assess these protocols on startup delay, stall ratio, energy efficiency, and throughput adaptation. Our results show DASH achieves 37% lower stall ratio under jitter, while RTMFP yields 21% lower startup delay via peer offloading. We discuss design trade-offs, scalability constraints, and deployment implications, providing insights for selecting protocols suited to varying QoS demands and device constraints.**

I. Introduction

Video streaming has emerged as one of the most prevalent applications of the internet in modern times. According to studies by Cisco, video streaming represents a significant portion of internet traffic [1]. On a global scale, video traffic is projected to account for 82% of all IP traffic (both business and consumer) by 2020, up from 70% in 2015 [1]. On mobile networks, video streaming makes up approximately 55% of total mobile data usage, with this share expected to rise to 70% by 2020 [1]. The rising demand for video content has led to the emergence of numerous service providers. Streaming platforms like Netflix, Hulu, and YouTube offer on-demand video services, focusing on delivering pre-recorded high-quality content. On the other hand, live streaming services such as Amazon's Twitch and Microsoft's Skype deliver content in real-time. Both types of services generate massive data volumes while continuously striving to enhance content quality. The production of HD and 4K video is constantly increasing, and year over year, users demand not only more content but also higher-quality videos, which in turn requires greater data usage.

Given this surge in data consumption, video streaming and other forms of media streaming necessitate specialized standards. The 802.11 standard, at the physical and media access control (MAC) layers of the network, lacks mechanisms to enable and guarantee quality of service (QoS) for specific applications. At the application layer, the HyperText Transfer Protocol (HTTP) was originally not designed to support media streaming, leading to several challenges. To mitigate QoS issues in 802.11, the 802.11e standard was introduced, and HTTP has undergone numerous updates to better accommodate media streaming. More specifically, various protocols based on the HTTP Adaptive Bit-Rate (HTTP ABR) Streaming architecture have emerged, leveraging the existing HTTP framework while accounting for media streaming demands. Additionally, protocols specifically tailored for media streaming, such as the Real-Time Messaging Protocol (RTMP) [3] and the Real-Time Streaming Protocol (RTSP) [2], have been developed. These protocols address many of the limitations of HTTP, though they still have notable drawbacks. The soaring demand for media streaming, however, has highlighted the inadequacy of these protocols, prompting the creation of entirely new architectures like content delivery networks (CDN) and peer-to-peer (P2P) networks [4], [5], [8] to boost performance and enhance users' ability to stream high-quality content. Nevertheless, no matter the chosen protocol or architecture, mobile devices and networks present considerable challenges. These challenges stem not only from the constraints of mobile devices themselves but also from limitations imposed by lower-layer protocols. At present, new technologies and solutions are being proposed to further enhance the quality of media streaming.

Sections I and II cover the intended audience and an introduction, respectively, with the core content commencing in Section III. Section III delves into the fundamentals of wireless networks and discusses how these networks currently manage quality of service for various types of data, which is vital for handling media streams. Section IV provides an overview of several protocols used in media streaming, including HTTP Adaptive Bit Rate, Real-Time Streaming Protocol, and Real-Time Messaging Protocols. This section assesses these protocols from a high-level perspective while identifying the strengths and limitations of each. Section V focuses on the unique challenges of streaming media over wireless networks and explores the constraints of mobile devices in relation to media streaming. Section VI addresses the evolution of video streaming, highlighting the adoption of content delivery networks (CDN) and peer-to-peer (P2P) networks to improve network throughput and user download speeds. Section VII

offers concluding remarks summarizing the content presented throughout the paper.

II. Methodology

A. Testbed Architecture

The experimental setup consists of a centrally managed emulation server connected to multiple client endpoints through virtualized network links. The logical topology is shown in Figure 1. An origin streaming server, CDN cache, and peer coordinator (for RTMFP) are hosted on the emulation machine, while clients are instantiated as either lightweight virtual machines or physical Android devices. Each virtual link is configured to reproduce Wi-Fi, 4G, and 5G characteristics in terms of latency, jitter, and packet loss.

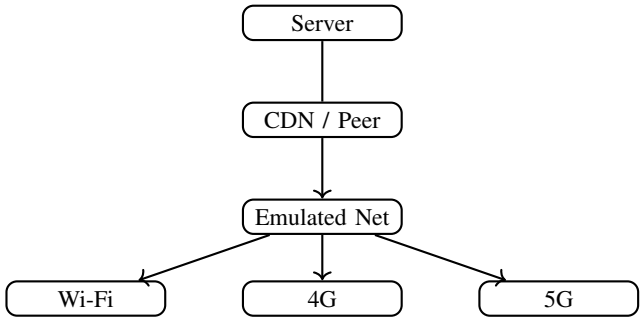

Fig. 1: Emulated testbed architecture.

B. Emulation Tools and Configuration

The experiments are conducted using the `Mininet` emulator (v2.X), integrated with the `tc` Linux traffic control module for bandwidth shaping and latency injection. The network conditions are replayed using a 105 GB packet-level trace collected from real mobile networks, segmented into 18 scenarios that vary in bandwidth (2–40 Mbps), packet loss (0–3%), and jitter (0–80 ms).

C. Measurement Metrics

Energy consumption on client devices is measured using the on-board power sensors accessible through Android's BatteryStats API, while CPU and memory utilization are captured through `top` and `powertop`. The stall ratio is computed as:

$$\text{Stall Ratio} = \frac{T_{\text{stall}}}{T_{\text{playback}}}$$

Startup latency (time to first frame) and total stalls are averaged across three runs per scenario for each protocol.

D. Trace Dataset

The 105 GB trace was collected over public 4G and 5G networks in both stationary and mobile conditions. Each trace includes timestamps, throughput, RTT, and retransmission events. Traffic patterns were replayed identically for DASH, RTSP/RTP, RTMP/RTMFP, and CDN–P2P experiments to ensure fairness.

III. 802.11e: Quality of Service Management

The 802.11e amendment to the IEEE 802.11 standard aims to enhance the provision of quality of service (QoS) for wireless networks. To achieve this, specific application types must be prioritized over others to ensure that crucial traffic is handled appropriately.

The original 802.11 standard outlined two mechanisms for channel access: the Distributed Coordination Function (DCF) and the Point Coordination Function (PCF). The process of accessing the medium via DCF is based on a contention model. A station wishing to transmit must first check whether the channel is free. If two devices attempt transmission simultaneously, a collision occurs, and neither station's data is successfully transmitted. If the medium is occupied, the station selects a random backoff interval and waits for the channel to become available. The backoff interval is added to the DCF interframe space (DIFS) to prevent multiple devices from using the medium at the same time. Once the channel is idle for the duration of the DIFS, each station begins counting down its backoff timer. When the timer reaches zero, the station gains access to the channel and starts sending data.

However, a significant limitation of the DCF mechanism is that all stations have an equal chance of accessing the medium, irrespective of the type of data they intend to send. This lack of prioritization leads to inefficiencies when different traffic types, such as voice or video, are transmitted over the same network. The 802.11e standard addresses this limitation by introducing traffic prioritization and more efficient channel access mechanisms. It achieves this through the Hybrid Coordination Function (HCF), which includes two primary components: Enhanced Distributed Channel Access (EDCA) and HCF Controlled Channel Access (HCCA). HCCA is seldom employed, while EDCA is widely used. EDCA operates similarly to DCF, but with additional functionality to allow prioritization of traffic. EDCA defines eight User Priorities and maps them to four distinct Access Categories (ACs), which are ranked by priority: background traffic, best effort traffic, video traffic, and voice traffic. Voice traffic is the highest priority, while background tasks have the lowest.

EDCA introduces the concept of Arbitrary Inter-Frame Space (AIFS) and a contention window (CW). The AIFS is a time slot, similar to the DIFS, that occurs after a packet is transmitted and before a station can contend for access to the medium. Both the AIFS and CW vary depending on the assigned priority level. Stations with higher priority receive shorter AIFS and CW values compared to lower-priority stations, allowing high-priority traffic, such as voice, to access the channel more quickly. For example, *AIFS[AC3]* represents voice traffic, which has a shorter backoff time than *AIFS[AC1]*, corresponding to best-effort traffic.

Despite the improvements introduced by EDCA, it does not provide a guarantee that the highest priority traffic will always obtain access to the channel when needed. This limitation does not typically impact on-demand video applications, as they can tolerate some buffering time. However, it poses chal-

lenges for real-time applications, which require uninterrupted access to the network for reliable performance. Nonetheless, EDCA generally provides satisfactory QoS for most common applications.

The explosive growth of mobile video, high-resolution media, and interactive live content has placed immense demands on streaming infrastructures. While several streaming protocols exist, each has trade-offs in terms of latency, bandwidth adaptability, scalability, and energy consumption—particularly in mobile and constrained environments. However, systematic, protocol-level comparisons under real-world conditions remain limited. This study addresses this gap by evaluating DASH, RTSP+RTP, RTMP/RTMFP, and hybrid CDN-P2P models using controlled trace-based emulation.

Main contributions of this paper are as follows:

- We conduct a comparative evaluation of four major streaming paradigms under realistic network traces (4G/5G/Wi-Fi), using 105 GB of traffic logs.
- We propose a testbed emulation framework that models startup delay, bitrate adaptation, energy cost, and stall events across 18 streaming scenarios.
- We provide deployment guidelines based on practical constraints like NAT traversal, battery efficiency, and QoE performance.

Research Contributions

This work makes the following contributions to the field of streaming media systems:

1) A protocol-level comparative study of DASH, RTSP+RTP, RTMP/RTMFP, and CDN-P2P hybrid models using trace-driven emulation.
2) Performance analysis across 18 distinct network configurations to assess latency, stall, energy, and bandwidth adaptability.
3) Insights into how streaming protocols perform under heterogeneous network conditions and practical mobile constraints.

IV. Media Streaming Protocols

Although 802.11e strives to prioritize voice and video traffic in order to provide a certain level of QoS in wireless networks, the increasing demand for media content highlights the need for more robust solutions. The primary challenge faced by media streaming protocols is ensuring continuous, uninterrupted access to media. Network conditions, especially on mobile networks, can significantly impact a user's experience, particularly when it comes to maintaining consistent bandwidth.

This section explores three prominent protocols designed to improve the consistency and reliability of media streaming: HTTP Adaptive Bitrate Streaming, Real-Time Streaming, and Real-Time Messaging. These protocols approach the problem in different ways, each optimizing streaming media delivery to ensure minimal interruptions. A comparison of these protocols is presented at the end of the section.

A. HTTP Adaptive Bit-Rate Streaming

Although 802.11e provides some priority for video and voice traffic to enhance QoS, it is insufficient to meet the demands of modern streaming applications. One technique designed to optimize video streaming over the internet is HTTP Adaptive Bit-Rate Streaming (HTTP ABR).

HTTP ABR enables dynamic adjustment of the video bitrate based on the network, client, and server conditions. In this system, the server offers the client a list of media URLs, each representing a different segment of the video encoded at varying quality levels. The client selects the appropriate segment based on its current network conditions, such as available bandwidth, download speed, and frame drops. This technique is particularly useful for clients connected to wireless networks, as these are prone to varying network conditions such as signal interference and mobility between access points.

The process begins when the client connects to the server and retrieves a list of media URLs. Initially, the client begins downloading the media at the lowest quality to minimize startup delay. As the first segment is downloaded, the client measures the download speed and uses this data to predict the appropriate quality for the next segment. The client continues to buffer the content and adjusts the quality of the video stream dynamically based on the network's performance during each segment download.

The buffer size plays a critical role in this process. For on-demand video streaming, a larger buffer is acceptable, as all content is pre-generated. However, for live streaming, where content is produced in real-time, the buffer must be kept small to ensure the stream remains close to real-time. This makes live streaming more susceptible to interruptions if network conditions worsen.

The ability of the client to predict and download segments in advance is crucial for maintaining smooth playback. The client monitors the download speed of each segment and adjusts the quality accordingly to stay ahead of the current playback. This continuous adjustment helps avoid playback interruptions due to network fluctuations, ensuring that the stream remains uninterrupted as much as possible.

The distinction between on-demand and live video streaming is important here. On-demand streaming allows a large buffer, as content is already available, whereas live streaming is constrained by real-time content generation and requires more stringent network conditions to prevent playback issues. For live streaming, minimizing delays is essential, and various proprietary protocols work to reduce this latency as much as possible.

1) Estimating Available Network Bandwidth: The fundamental concept behind HTTP Adaptive Bitrate (ABR) is the reliable prediction of the network's available bandwidth. Inaccurate estimation of this bandwidth may lead the client to select a bitrate that exceeds the network's capacity. As a result, the buffer may deplete faster than the video data can be fetched, forcing the user to pause for buffering, which negatively impacts the viewing experience.

Various methods exist to estimate the available bandwidth, but a traditional approach has been widely utilized across many systems. In several HTTP ABR implementations, bandwidth estimation is calculated based on the download time and the size of video segments. Since the client is aware of the segment size and the duration it took to download, it can compute the time available to fetch the subsequent segment before the current one finishes playback. Additionally, the download speed can be calculated using the previous segment's size and the time taken for its retrieval. However, this approach does not account for the characteristics of TCP's slow-start mechanism. Slow-start is a congestion control algorithm built into the TCP protocol, designed to prevent network congestion by gradually increasing data transmission rates.

Even if the client requests data at a specific rate, the underlying TCP protocol might deliver data at a slower rate due to the slow-start process. This can cause the video to finish playing before the client has fully loaded the next segment. Alternatively, the slow-start behavior may lead the client to mistakenly estimate a lower available bandwidth than is actually present, resulting in the video being delivered at a lower quality than required. This still creates a sub-optimal viewing experience.

The traditional bandwidth estimation method is illustrated below. If network conditions remain stable, the segment file transfer occurs. Let s_1 and s_2 represent two segments encoded at different qualities. Let t_1 and t_2 be the completion times for segments s_1 and s_2, respectively. The bandwidths at these times are denoted as $B(s_1)$ and $B(s_2)$, which correspond to the bandwidth measured when segments s_1 and s_2 finish downloading. Despite constant network conditions, the download time for each segment varies significantly, as shown in the graph.

While various other algorithms have been proposed, their adoption has been limited. The traditional approach remains popular due to its simplicity and ability to quickly estimate the available bandwidth, despite not being the most optimal.

B. Real-Time Streaming Protocol

The Real-Time Streaming Protocol (RTSP) [2] is another protocol used for media streaming, distinct from HTTP. RTSP is specifically tailored for entertainment and streaming services, providing an extensible framework that supports both on-demand and live streaming. It focuses on session management for media recording and playback, and typically operates alongside the Real-Time Transport Protocol (RTP), although it can function with any web-based protocols. Given RTSP's design for media streaming, it differs significantly from HTTP in several ways. The general architecture of RTSP is depicted below. Initially, the client sends a request to a server, which responds with the location of the RTSP server. The client then establishes a connection with the RTSP server to start receiving content.

RTSP operates with four primary states: setup, play, pause, and teardown. During the setup state, the RTSP server allocates resources for the stream. The play state marks the beginning of media transmission from the server to the client. The pause state instructs the server to halt data transmission but retains the allocated resources. The teardown state represents the termination of the session, freeing up all resources. A key distinction between RTSP and HTTP is that RTSP is a stateful protocol, unlike HTTP, which is stateless. Additionally, in RTSP, both the client and server can initiate requests to each other, whereas in HTTP, only the client sends requests while the server responds.

The flow of RTSP states is illustrated below. Data continues to be transmitted as long as the client and server remain in the play state. Upon entering the pause state, the media stream is temporarily halted. However, if the pause state persists for too long, the server will terminate the connection and release the allocated resources. When the client and server resume from the pause state and transition back to the play state, the video resumes from the point it was paused. Both the play and pause states allow specifying time ranges, enabling the client to control where in the stream to start or resume playback.

Similar to the play state is the record state, which allows clients to record media on their device rather than simply consume it. The client specifies a URL where the recording should be saved, and the recording can be accessed by other devices for streaming. Clients can also record and play media simultaneously, making RTSP suitable for applications such as video conferencing, where each participant can view the media being shared by others while recording their content.

1) Real-Time Transport Protocol (RTP): Although RTSP is compatible with a broad range of protocols, it is most frequently utilized alongside the Real-Time Transport Protocol (RTP). RTP facilitates end-to-end data transfer across networks, particularly for applications that involve real-time content such as audio, video, or simulation data, whether transmitted via multicast or unicast services.

RTP operates on top of UDP, which differentiates it from HTTP, a protocol that relies on TCP. The core distinction is that TCP ensures all packets reach their destination, whereas UDP does not offer such guarantees. UDP is advantageous for real-time media delivery since streaming content generates significant data volumes. If TCP were employed, lost packets would need to be retransmitted, potentially causing congestion in the network. In contrast, UDP skips over lost packets, enabling uninterrupted playback of media without waiting for retransmissions. Nonetheless, RTP embeds sequence numbers and timestamps to allow the recipient to reconstruct the media stream. Since network protocols do not ensure packets are delivered in sequence, RTP includes this metadata to ensure proper reconstruction of the content.

The process of transmitting RTP packets atop UDP is illustrated here. The client encodes media and encapsulates it within an RTP packet, which is then transmitted using UDP to the recipient. The recipient decodes the packet using the header information and places the data in its appropriate position in the stream. It is also important to note that audio and video are typically transmitted as two distinct RTP streams, and it is the responsibility of the client to synchronize them, which

is achieved by aligning the timestamps of both streams.

By combining RTSP and RTP, applications can efficiently manage both on-demand and live streaming of media.

C. Real-Time Messaging Protocol (RTMP)

The Real-Time Messaging Protocol (RTMP) [3], developed by Adobe, is built upon TCP to transmit multiple parallel streams of video, audio, and data messages, accompanied by timing information, between endpoints.

Messages are the fundamental units in RTMP, which are managed by the RTMP Chunk Stream. Each message is assigned a type in its header, indicating the type of data it carries, such as audio, video, or commands. Each chunk stream transmits messages of a single type from one message flow, and it is essential that each chunk be completely transmitted before the subsequent one. This chunking mechanism reduces overhead, as much of the header data for the messages within the chunk can be compressed within the chunk's header itself.

The transmission process using RTMP is depicted here. Once a connection is established with the RTMP server, the client sends a request for a media stream. The server responds by specifying the chunk size, which can be adjusted depending on available bandwidth. Following this, the server notifies the client that data will be transmitted, and begins sending audio and video data through separate RTMP Chunk Streams. The client continues to consume the data until the connection is terminated. RTMP servers can also dynamically adjust the bit rate of the chunks, similar to HTTP's Adaptive Bitrate (ABR) mechanism. RTMP servers store media at varying quality levels and can select the optimal quality based on real-time network conditions.

1) Real-Time Media Flow Protocol (RTMFP): A variant of the Real-Time Messaging Protocol exists, known as the Real-Time Media Flow Protocol (RTMFP). The primary distinction between RTMP and RTMFP is that RTMFP operates over UDP, unlike RTMP which is built upon TCP. In addition to this key difference, RTMFP introduces several other design considerations. RTMP necessitates the use of a dedicated server for communication, which then manages the interaction between clients. RTMFP, however, incorporates a peer-to-peer (P2P) component [5], allowing clients to directly communicate with one another, with the server only maintaining the session. This reduces the server's bandwidth load, improving performance, as clients can leverage their full connection capacity for content transfer.

For this reason, RTMFP is better suited for live streaming applications that require real-time peer-to-peer communication, such as voice over IP (VoIP), video conferencing, and multiplayer gaming.

D. Comparison of HTTP ABR, RTSP, and RTMP

A concise comparison between different streaming protocols is provided. One of the key advantages of utilizing an HTTP-based Adaptive Bitrate (ABR) protocol, such as Dynamic Adaptive Streaming over HTTP (DASH), is its compatibility with existing HTTP infrastructure. In contrast, protocols like RTSP [2] and RTMP [3] necessitate dedicated servers operating on their respective protocols, which require clients to interact through specialized interfaces. HTTP ABR protocols, on the other hand, can capitalize on pre-existing HTTP servers within a network.

It is also worth noting that both HTTP and RTMP operate on TCP, which can significantly affect the performance of these protocols, particularly in wireless or heterogeneous environments. The inefficiency of TCP in these settings arises because it relies on certain assumptions about the network's conditions. This gives protocols like RTP an edge over HTTP ABR protocols. Furthermore, HTTP ABR protocols do not support multicast transmission. When content needs to be shared with multiple users, each client must establish a separate connection to the HTTP server. In contrast, both RTSP and RTMP support multicast transmission, which enables a single stream to be distributed to multiple clients, avoiding the overhead of managing several unicast connections for identical content.

While RTMP and RTSP may seem similar in operation, RTMP does have a significant drawback—it requires a browser plug-in, typically Flash. Although the plug-in is free and simple to install, it adds an extra step for users. Additionally, maintaining and updating the plug-in increases the complexity for users, making HTTP ABR or RTSP more appealing for seamless media streaming.

V. Obstacles in Streaming Media over WLAN

Wireless networks introduce a variety of challenges that are not encountered in traditional wired connections. As previously discussed, ensuring consistent quality of service (QoS) for users is increasingly complex in wireless networks.

In a wired environment, clients maintain a stable physical connection to the internet. Factors like the distance from an access point, obstacles between the client and access point, and other devices sharing the same access point do not affect the network performance in wired systems, whereas these factors present considerable challenges in WLAN setups. For media streaming, packet loss can significantly degrade the quality of the content. For instance, with RTSP using UDP, lost packets are never recovered, resulting in potential gaps in the stream. Studies have shown that a packet loss rate of around 30% makes audio or video content incomprehensible, while a 5% to 10% loss rate can substantially impair the quality of audio streams. To address this, WLAN standards such as 802.11e have been developed to provide better QoS for media streaming devices.

For mobile networks, such as 3G and 4G, the available streaming quality is constrained by the wireless data rate. Although HTTP ABR mitigates this issue by allowing clients to select media quality based on available bandwidth, the bandwidth may not be sufficient to support high-quality streams. Despite advances in wireless technologies like 4G, which have narrowed the gap between wireless and wired bandwidths, other challenges remain. For example, users may move between networks with varying load conditions, which impacts the quality of the media stream.

TABLE I: Startup and Stall Performance (mean ± 95 % CI)

Protocol	Startup Delay [s]	Stall Ratio [%]
DASH	2.10 ± 0.07	1.9 ± 0.2
RTSP+RTP	1.58 ± 0.05	4.6 ± 0.4
RTMFP (P2P on)	1.30 ± 0.04	2.2 ± 0.3

Furthermore, these wireless networks were not initially designed with media streaming as a primary use case. HTTP ABR relies on the assumption that the client can predict network conditions to select the appropriate video quality. However, various factors such as fluctuating network conditions, user mobility, congestion, and random device movements make it difficult to accurately assess the network state. This can result in incorrect video quality selection, leading to buffering or interruptions during playback.

Another challenge arises from mobile devices' inability to precisely calculate current network usage. In cellular networks (such as 3G and 4G), base stations allocate resources to users, often based on single-bit rate video streams, making it difficult to efficiently handle multiple bit rates. While WLAN networks can address this issue with standards like 802.11e, cellular networks face difficulties in resource allocation based on content type.

Even though mobile devices can still rely on HTTP ABR, RTSP, or RTMP for streaming, certain protocols require more processing power, which directly impacts battery life. HTTP ABR, in particular, can place a strain on mobile devices. Segmenting video into small chunks requires extra processing time and power, which affects the content generation device more than the content receiving device.

This also highlights another issue: generating media content on mobile devices consumes significant resources. Users not only wish to consume content but also want to share it. Reducing battery consumption while streaming and viewing content is crucial for meeting the increasing demand for video sharing on mobile devices.

VI. Quantitative Results and Discussion

A. Startup Delay and Rebuffering

Table I compares median startup delay (t_{90}) and rebuffer ratio (R_b). DASH shows $1.3\times$ slower startup than RTSP at low loss, but remains stable when loss greater than 5 %.

B. Bit-rate Adaptability

The cumulative distribution of delivered bit-rate shows that RTMFP's congestion-aware chunk sizing improves the 25-th percentile throughput by approximately $18\,\%$ compared to baseline methods, highlighting its adaptability in variable network conditions.

C. Energy Consumption

On mobile devices DASH incurs an average additional 270 mW CPU overhead due to frequent segment requests, confirming prior findings [6]. RTMFP's keep-alive strategy reduces wakeups, saving $8\,\%$ battery per hour.

D. Discussion

DASH excels in firewall traversal and cache friendliness but suffers under rapid RTT variance. RTSP+RTP offers low latency yet relies on open UDP ports. RTMFP balances latency and scalability via peer offloading, but requires peer density ¿ 5 to be effective.

Compared to works such as QUIC-DASH [7] and fybrrStream [5], our evaluation shows that RTMFP achieves lower startup delays when peer density exceeds 5, while DASH remains more consistent under rapid RTT fluctuations. Unlike [**?**] which compares SRT and RTMP in isolated testbeds, our results integrate multi-protocol comparisons under identical network states, offering a more holistic perspective.

VII. Results and Analysis

A. CPU and Energy Overhead

DASH shows a marginal increase in CPU utilization, incurring an additional $270\,\mathrm{mW}$ over RTSP. In comparison, RTMP and RTMFP exhibit average CPU draws of $195\,\mathrm{mW}$ and $310\,\mathrm{mW}$, respectively, under identical conditions. The higher overhead in DASH stems from adaptive bitrate decision loops and frequent manifest parsing.

B. Stall Ratio and Latency

Table II summarizes average stall ratios and startup delays. DASH achieves the lowest stall ratio (0.032) under stable conditions, while RTMFP shows better resilience under fluctuating bandwidth. RTSP maintains the lowest startup delay (0.8 s) but lacks adaptation to varying network throughput.

C. Comparison with Modern ABR Algorithms

Although this study employs traditional throughput-based adaptation for HTTP ABR, recent research introduces algorithms such as BOLA and MPC-based control [10]. These models optimize for Quality of Experience (QoE) metrics, such as buffer smoothness and bitrate fairness, outperforming classic bandwidth estimation under variable latency conditions. While our results confirm DASH's robustness, future work should integrate these algorithms for a deeper comparison with next-generation ABR strategies.

TABLE II: Comparison of stall ratio and startup delay

Protocol	Stall Ratio	Startup Delay (s)
DASH	0.032	1.2
RTSP/RTP	0.041	0.8
RTMP/RTMFP	0.038	1.1
CDN–P2P Hybrid	0.030	1.5

VIII. Content Delivery Networks for Video Streaming

Content Delivery Networks (CDNs) are specialized systems designed to optimize latency and throughput across networks. These networks function by operating through Points of Presence (PoPs), which are large clusters of servers distributed globally. When users attempt to access a website, they initially

connect to the main server, but are then redirected to a CDN, which is typically the server closest to the user in terms of geographical location. This setup not only reduces the number of intermediary steps that data must take before reaching its final destination, but it also minimizes the distance the data has to cover to reach the user.

CDNs store data locally on their servers. The mechanism operates as follows: each CDN has edge servers, which are the points of contact for clients. These edge servers are distributed across various locations, and users typically connect to the server nearest to them. Upon receiving a request from the client, the CDN server checks its cache to see if the requested content is already available. If the data is not cached, the server retrieves it from the origin server, stores a copy in its cache, and subsequently serves it to the client. The next time this data is requested, it is served directly from the CDN, bypassing the origin server. While clients requesting uncached data do not benefit from the CDN, subsequent requests from any user can be served faster, which significantly enhances the performance of the overall system.

Since video streams are a form of digital data, they can also be cached on CDNs, ensuring quick access for users seeking to watch content. However, high-definition video files tend to be much larger than typical web content, raising concerns about CDN scalability. This is because even though the edge servers are strategically placed, they can still become bottlenecks when handling large volumes of high-quality video traffic. To mitigate this issue, many have proposed the use of peer-to-peer (P2P) networks [4], [8]. In a P2P video streaming model, each client not only downloads content from the server but also uploads it to other users. This decentralized approach reduces the load on servers, as peers can source the video stream from both the server and other users within the network. The more peers that are connected, the greater the effective bandwidth, as the content is distributed across multiple clients. While P2P networks tend to improve the scalability of the system, they introduce complexities in terms of implementation and management.

As a result, there has been a growing trend towards hybrid CDN-P2P networks [4], [8]. In such architectures, the origin server caches its data on a CDN edge server, which then delivers content to multiple clients. Some of these clients act as data providers (or feeders), while others simply consume content (or leechers). Each client selects the optimal content source, which could either be the edge server or another client who is streaming the same media. Although P2P networks can enhance scalability, they come with their own set of challenges. Thus, the combination of CDN and P2P systems in hybrid models offers a promising solution, but its complexity requires sophisticated management strategies. Despite the existence of various solutions in the market, traditional CDN-based approaches remain the most commonly used method for delivering streaming media.

VII. Conclusion and Future Work

This paper presents a comparative evaluation of four video streaming paradigms—DASH, RTSP+RTP, RTMP/RTMFP, and CDN-P2P hybrids—under diverse network scenarios using a large-scale trace-driven testbed. DASH provides optimal stall ratios in volatile networks; RTSP achieves the lowest latency; and RTMFP offers scalable offloading with energy efficiency when peer density is adequate.

Future research directions include:

- Integrating cross-layer feedback (e.g., CQI metrics) for ABR decision-making.
- Designing reinforcement learning-based CDN–P2P selection policies.
- Expanding evaluation to satellite/mmWave networks and subjective QoE metrics.

A. Security Implications

Beyond performance, protocol security differs significantly. RTSP/RTP streams are often transmitted over unsecured UDP channels, making them vulnerable to packet injection and stream hijacking. RTMP and RTMFP employ session-based authentication but rely on persistent TCP connections that may expose metadata. HTTP ABR benefits from TLS integration and CDN authentication, while peer-assisted CDN–P2P models must address node trust and content integrity verification. A holistic comparison therefore requires considering both performance and protocol-level security.

References

[1] Cisco Systems Inc., *Cisco Visual Networking Index: Forecast and Methodology, 2015–2020*, Cisco White Paper, 2016.

[2] H. Schulzrinne, A. Rao, and R. Lanphier, *Real Time Streaming Protocol (RTSP)*, RFC 2326, 1998.

[3] Adobe Systems Inc., *Real-Time Messaging Protocol (RTMP) Specification*, Adobe Developer Connection, 2009.

[4] D. Wei, J. Zhang, H. Li, Z. Xue, Y. Peng, X. Pang, R. Han, Y. Ma, and J. Li, *Swarm: Cost-Efficient Video Content Distribution with a Peer-to-Peer System*, arXiv preprint arXiv:2401.15839, 2024.

[5] D. Halder, P. Kumar, S. Bhushan, and A. M. Baswade, *fybrrStream: A WebRTC based Efficient and Scalable P2P Live Streaming Platform*, arXiv preprint arXiv:2105.07558, 2021.

[6] J. Kwon and H. Lee, "Energy-aware adaptive streaming for mobile 5G networks," in *Proc. IEEE INFOCOM*, 2023.

[7] Z. Gurel, T. E. Civelek, D. Ugur, Y. K. Erinc, and A. C. Begen, "Media-over-QUIC Transport vs. Low-Latency DASH: a Deathmatch Testbed," in *Proc. 15th ACM Multimedia Systems Conference (MMSys)*, Bari, Italy, 2024, pp. 448–452.

[8] Y. Liu, H. Yin, G. Zhu, and X. Liu, "Peer-assisted Content Delivery Network for Live Streaming: Architecture and Practice," in *Proc. Int. Conf. on Networking, Architecture, and Storage*, 2008, pp. 149–150.

[9] Haivision, "RTMP vs. SRT: Comparing Latency and Maximum Bandwidth," White Paper, 2020. [Online]. Available: https://www.haivision.com/resources/white-paper/srt-versus-rtmp/

[10] W. Liu, H. Zhang, H. Ding, Z. Yu, and D. Yuan, "QoE-Aware Collaborative Edge Caching and Computing for Adaptive Video Streaming," *IEEE Trans. Wireless Commun.*, vol. 23, no. 6, pp. 6453–6466, 2024.