International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

Performance Analysis of Video Transmission Protocols

Amanpreet Kaur
Department of Computer Science and Engineering
Bhai Maha Singh College of Engineering
Sri Muktsar Sahib. India

Kulwinder Singh
Director Principal
Bhai Maha Singh College of Engineering
Sri Muktsar Sahib. India

Abstract **- The exponential rise in multimedia data has reshaped the architecture of the modern Internet. Video streaming, once limited by bandwidth and infrastructure constraints, now constitutes over 80 % of total Internet traffic. Efficient transmission protocols are the foundation for this digital transformation, influencing latency, quality of service (QoS), and end-user experience. This study presents a comprehensive experimental and analytical evaluation of major video transmission protocols Real-Time Messaging Protocol (RTMP), Real-Time Transport Protocol (RTP) and Real-Time Streaming Protocol (RTSP), Hypertext Transfer Protocol Live Streaming (HLS), Dynamic Adaptive Streaming over HTTP (DASH), Secure Reliable Transport (SRT), Faster-Than-Light (FTL), Microsoft Smooth Streaming (MSS), and Web Real-Time Communication (WebRTC). An experimental testbed was implemented using Docker-based NGINX servers and OBS Studio for live and on-demand content generation. Latency, jitter, bandwidth utilization, and adaptability under variable network conditions were systematically analyzed. WebRTC demonstrated superior performance with sub-second latency suitable for real-time interactive applications, whereas HLS and DASH offered robust adaptive bitrate streaming ideal for content delivery networks. RTMP maintained steady throughput under stable network environments but lacked scalability. Emerging protocols such as SRT and FTL achieved notable improvements in reliability and encryption but displayed sensitivity to congestion. The results identify key performance trade-offs, establishing a decision framework for selecting appropriate protocols in diverse multimedia environments. This work contributes a unified comparison of both traditional and modern streaming protocols under identical experimental conditions, providing reproducible insights that inform the design of next-generation low-latency, adaptive streaming systems.**

Keywords - Video Streaming Protocols, WebRTC, RTMP, HLS, DASH, SRT, Latency, Quality of Service (QoS), Adaptive Streaming.

I. INTRODUCTION

A. Background and Motivation

Over the last two decades, digital communication has experienced a paradigm shift from text-centric to media-centric interaction. The integration of video into daily communication through conferencing platforms, remote education, entertainment services, and telemedicine has made continuous, high- quality video delivery an essential component of global information infrastructure [1]. The widespread deployment of broadband networks, proliferation of smartphones, and emergence of 5G technologies have accelerated this transformation.

However, delivering smooth video experiences remains technically challenging. Network heterogeneity, unpredictable latency, packet loss, and fluctuating bandwidth still constrain end-to-end performance. To ensure consistent playback and minimal buffering, video delivery systems rely on streaming protocols that define how video data is encapsulated, transmitted, and reconstructed at the receiver's end. These protocols determine not only the technical efficiency of data transfer but also the overall Quality of Experience (QoE) perceived by users [2].

The growing demand for ultra-low latency and high-definition (HD) content has led to the evolution of multiple generations of streaming protocols, each tailored to specific network architectures and application domains. For instance, RTMP, introduced by Adobe in the early 2000s, enabled real-time media delivery using TCP.

Later, HTTP-based adaptive streaming protocols such as HLS and DASH improved scalability by leveraging standard web infrastructures [3, 4]. In parallel, WebRTC, standardized by the W3C and IETF, revolutionized peer-to-peer real-time communication by enabling browser-native audio-video exchange without external plugins. Despite their shared goal of seamless media delivery, these protocols differ significantly in architecture, transport mechanisms, and performance characteristics [5].

B. Need for Comparative Evaluation

Most prior studies have focused on optimizing individual protocols rather than conducting comprehensive cross-protocol comparisons under identical conditions. Yet, system designers must frequently decide which protocol best suits their use case: live broadcast, on-demand streaming, interactive conferencing, or hybrid deployment. For example:

- HLS excels in adaptive bit rate control but introduces 5–8 s latency due to its segment-based nature.
- DASH improves adaptation granularity but still relies on TCP retransmissions.
- RTMP offers low-latency streaming but depends on deprecated Flash environments.
- WebRTC achieves sub-second latency but requires higher computational overhead for peer negotiation and encryption.
- SRT, FTL, and MSS further complicate the selection landscape with their proprietary optimizations.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

Therefore, a reproducible, experimental framework is required to assess these protocols under uniform conditions systematically. This study addresses that gap through empirical testing on a controlled Docker-based setup, ensuring comparability and repeatability [6].

C. Evolution of Video Transmission Protocols

The history of video transmission can be divided into three major phases:

1) Phase I: Traditional Transport Protocols

Protocols like RTP, RTSP, and RTMP emerged during the early development of Internet multimedia. They focused on ensuring synchronized packet playback control. RTP provided sequencing and timestamping, while RTSP added session-level control commands such as PLAY, PAUSE, and TEARDOWN. RTMP, layered over TCP, simplified integration with Flash players and allowed multiplexed audio-video streaming.

2) Phase II: HTTP-Based Adaptive Streaming

With the rapid expansion of the World Wide Web, HTTP-based streaming protocols became dominant. Apple's HLS, Microsoft's MSS, Adobe's HDS, and the open-standard MPEG-DASH exploited HTTP's cache-friendly behavior and firewall traversal capabilities. They segmented video content into small chunks encoded at multiple bitrates, allowing dynamic adaptation to available bandwidth. This phase marked the transition from proprietary to standards-driven video delivery [6].

3) Phase III: Real-Time and Secure Protocols

Modern applications demand instantaneous interaction. WebRTC enabled real-time peer-to-peer communication directly in browsers using the Secure RTP (SRTP) framework, while SRT and FTL introduced advanced encryption and retransmission mechanisms over UDP to minimize latency. These developments reflect the convergence of communication and content-delivery paradigms into unified frameworks optimized for both interactivity and quality [7].

The remainder of the paper is organized as follows. Section 2 presents a comprehensive review of related literature covering all major streaming protocols and comparative analyses. Section 3 details the materials and methods, including system architecture, software configuration, and measurement techniques. Section 4 reports the experimental results with extensive figures and tables illustrating protocol performance under varying conditions. Section 5 provides an in-depth discussion linking empirical observations to theoretical insights. Section 6 concludes with final remarks and outlines future research directions.

II. LITERATURE REVIEW

A. Overview of Video-Transmission Research

Video streaming has evolved into one of the most data-intensive and latency-sensitive applications on the Internet. Early work on multimedia communication focused on ensuring reliable transport of audio and video packets over bandwidth-constrained networks with frequent packet loss. Early researchers [1] laid the foundation with the Real-time Transport Protocol (RTP), designed for low-delay transmission of real-time media. Soon afterward, Real-time Streaming Protocol (RTSP) was standardized to provide playback control functionalities, such as play, pause, and tear down, similar to VCR operations but adapted to digital networking.

As broadband and mobile networks matured, the focus of video-streaming research shifted from basic connectivity to user experience and scalability. The early 2000s introduced proprietary protocols such as Real-Time Messaging Protocol (RTMP), which enabled Adobe Flash-based live broadcasting. RTMP's reliability and simplicity made it ubiquitous for a decade; however, its dependency on Flash eventually limited its relevance as HTML5 became the web standard.

The next generation of streaming research investigated HTTP-based adaptive streaming, which allowed media content to be delivered in small segments at multiple bitrates. The key advantage was compatibility with existing web infrastructure and content-delivery networks (CDNs). Apple's HTTP Live Streaming (HLS) and the ISO/IEC Dynamic Adaptive Streaming over HTTP (DASH) standard are now dominant in this category. Their success lies in utilizing standard HTTP ports (80/443), making them firewall-friendly, and providing adaptive bitrate logic that dynamically adjusts video quality to the viewer's available bandwidth. With the rise of interactive and ultra-low-latency use cases, such as video conferencing, online gaming, and telemedicine, the research focus turned again to real-time communication protocols.

Web Real-Time Communication (WebRTC) emerged as a transformative framework enabling encrypted, peer-to-peer media exchange directly between browsers without plugins. WebRTC integrates a suite of sub-protocols, including SRTP (Secure RTP), ICE (Interactive Connectivity Establishment), STUN, and TURN for NAT traversal, allowing resilient real-time communication even across complex network topologies.

The latest research frontier explores Secure Reliable Transport (SRT) and Faster-Than-Light (FTL) protocols, designed to minimize delay while providing encryption and packet-recovery mechanisms over unreliable networks. These modern UDP-based systems combine security with reliability, making them attractive for cloud broadcasting and hybrid network environments. Microsoft's Smooth Streaming (MSS) remains historically important for adaptive streaming within the Microsoft ecosystem and provides comparative insight into closed-architecture systems.

The literature consistently underscores a central challenge: balancing latency, reliability, scalability, and adaptability. Each protocol makes distinct trade-offs, optimized for particular network conditions or application requirements. Table 1 (summarized description below) presents an overview of the fundamental characteristics of the major streaming protocols studied in prior research.

TABLE I. COMPARATIVE SUMMARY OF MAJOR STREAMING PROTOCOLS IN LITERATURE

Protocol	Transport Layer	Typical Latency	Adaptivity Mechanism	Encryption Support	Primary Use Case	Key Limitation

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

RTP/RTSP	UDP (+ Control)	0.2–1 s	Manual Session Control	Optional SRTP	Surveillance, VoIP	Limited scalability over NAT
RTMP	TCP	1–5 s	Fixed bitrate	Optional SSL	Flash live streaming	Deprecated plugin dependence
HLS	HTTP (TCP)	5–10 s	Segment-based adaptive bitrate	AES-128	On-demand video	High latency
DASH	HTTP (TCP)	3–6 s	Adaptive manifest selection	Optional DRM	CDN delivery	Fragment overhead
WebRTC	UDP/SRTP	< 1 s	Dynamic peer feedback	Mandatory SRTP	Real-time conferencing	High complexity
SRT	UDP	0.3–1 s	ARQ + Selective Retransmission	AES-256	Broadcast links	Sensitive to burst loss
FTL	UDP	< 0.5 s	Low-buffer mode	AES	Interactive streaming	Proprietary limitations

1) Real-Time Transport Protocol (RTP) and Real-Time Streaming Protocol (RTSP)

RTP is widely recognized as the backbone for real-time multimedia transport. It operates on top of UDP, providing sequence numbering and timestamping to enable synchronization of media streams. The companion protocol, RTCP (Real-Time Control Protocol), exchanges feedback such as packet-loss rates and jitter statistics, facilitating adaptive control. RTSP complements RTP by handling session initialization and user commands, effectively acting as a "remote control" for media servers.

[1] noted that RTP's lightweight design allows high throughput with low overhead, but it sacrifices reliability. RTSP adds flexibility by allowing on-demand streaming, yet its dependency on dedicated ports and the need for persistent connections limit scalability through firewalls. Research by [2] compared RTP/RTSP to HTTP-based systems, revealing superior real-time performance but reduced deployability in cloud environments.

Experimental studies show that RTP/RTSP architectures achieve end-to-end delays between 200 and 800 ms, suitable for live surveillance or conferencing. However, under adverse network conditions with packet loss above 5 %, users experience frame stutter and audio desynchronization. Enhancements such as RTP over QUIC and hybrid RTSP-HTTP tunnelling have been explored to mitigate these issues.

2) Real-Time Messaging Protocol (RTMP)

RTMP, introduced by Adobe Systems, became the de facto standard for live video delivery during the Flash Player era. It multiplexes control and media messages over a single TCP connection, ensuring ordered delivery. Its strengths include simplicity, built-in commands for publishing and subscribing streams, and wide encoder support.

According to [8], RTMP's major drawback is the latency introduced by TCP's congestion-control mechanism. Each lost packet triggers retransmission delays, making RTMP unsuitable for highly interactive streaming. Despite this, RTMP maintains value in closed environments due to its predictable behaviour and easy integration with open-source servers like Red5 and the NGINX-RTMP module.

Recent studies [8] benchmarked RTMP alongside HLS and WebRTC. They observed average latencies of 2–3 s for RTMP, 6–8 s for HLS, and < 1 s for WebRTC. Though RTMP cannot meet sub-second latency targets, it continues to serve as a stable baseline for legacy platforms and as a first hop in re-encoding pipelines used by streaming services such as YouTube Live.

3) Hypertext Transfer Protocol Live Streaming (HLS)

Apple introduced HTTP Live Streaming (HLS) in 2009 as a robust, HTTP-based protocol designed to improve scalability and compatibility with existing web infrastructure. HLS divides video into small, time-segmented .ts files, typically ranging from 2 to 10 seconds each, and serves them over HTTP. A master playlist (.m3u8) guides clients in choosing appropriate segments based on available network bandwidth and device capabilities.

The use of HTTP/TCP provides several advantages: easy deployment through standard ports (80/443), compatibility with CDNs, and resilience behind firewalls. However, TCP's inherent retransmission mechanisms introduce noticeable delay. Each segment must be fully received before playback begins, leading to higher startup latency and buffering.

Experimental Insights: Studies such as [9] demonstrate that HLS achieves high reliability and stable playback, particularly in variable network conditions. However, latency typically ranges from 5 to 10 seconds, depending on segment size and player buffering strategy. Reducing segment duration can lower delay but increases overhead due to more frequent HTTP requests.

Adaptation and Quality Control: HLS's adaptive bitrate mechanism relies on periodic measurement of download times for previous segments. This approach enables smooth transitions between different quality levels, preventing abrupt resolution changes that degrade user experience. However, since adaptation decisions are made at the application layer, response time to sudden network fluctuations is slower than in transport-layer adaptive protocols such as WebRTC.

Security Features: HLS supports AES-128 encryption and HTTPS transport, making it suitable for premium content distribution. It is widely used by Apple TV+, Netflix, and Hulu for on-demand video. streaming. Despite its high latency, HLS remains the industry standard for large-scale media delivery due to its stability and seamless CDN integration.

4) Dynamic Adaptive Streaming over HTTP (DASH)

MPEG-DASH represents the first international standard for adaptive HTTP streaming. Published by ISO/IEC in 2012, DASH generalizes the concepts introduced by HLS but removes vendor lock-in by supporting any container format (MP4, Web M, etc.). It uses an XML-based Media Presentation Description (MPD) file to describe available bitrates, segment URLs, and timing information.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

Adaptive Mechanism: DASH relies on dynamic adaptation algorithms that select video segments based on buffer occupancy, download throughput, and playback history. Various adaptation logics, such as buffer-based, rate-based, and hybrid approaches, have been developed to optimize Quality of Experience (QoE). Research by [10] highlighted DASH efficiency in balancing video quality and rebuffering events.

Performance Characteristics: Latency in DASH depends on segment length and buffer configuration, typically ranging between 3 and 6 seconds. Unlike HLS, DASH supports finer-grained segment durations (e.g., 1–2 s), enabling more responsive adaptation. However, frequent segment requests can burden the HTTP server, increasing CPU utilization.

Comparative Studies: Empirical comparisons [11] show that DASH outperforms HLS in terms of quality adaptation and bandwidth efficiency but lags behind WebRTC in latency-critical scenarios. DASH's standardized nature makes it a preferred choice for hybrid systems combining adaptive HTTP delivery with low-latency real-time overlays.

Security and Interoperability: DASH is highly extensible and supports multiple DRM systems through the Common Encryption (CENC) standard. This allows interoperability across devices and platforms. It has been adopted by major media platforms, including YouTube, Netflix, and Amazon Prime Video.

MPEG-DASH represents the first international standard for adaptive HTTP streaming. Published by ISO/IEC in 2012, DASH generalizes the concepts introduced by HLS but removes vendor lock-in by supporting any container format (MP4, Web M, etc.). It uses an XML-based Media Presentation Description (MPD) file to describe available bitrates, segment URLs, and timing information.

B. Web Real-Time Communication (WebRTC)

WebRTC emerged as the most transformative technology in modern video transmission. Unlike traditional streaming protocols that rely on centralized servers, WebRTC enables direct peer-to-peer communication between browsers using UDP-based channels secured by Secure Real-Time Transport Protocol (SRTP).

1) Core Components WebRTC integrates several underlying technologies:

- **ICE (Interactive Connectivity Establishment):** Manages peer discovery and connection negotiation.
- **STUN/TURN:** Facilitates NAT traversal, allowing communication between devices behind firewalls.
- **DTLS-SRTP:** Provides end-to-end encryption and integrity verification.

This architecture allows WebRTC to achieve sub-second latency (<500 ms) while maintaining secure and resilient communication, making it ideal for live conferencing, remote education, and cloud gaming.

Performance Studies: Research by [11] shows that WebRTC achieves the lowest end-to-end delay among all tested protocols, but consumes higher CPU and memory resources. Its dynamic congestion-control algorithm (GCC) adjusts sending rates based on real-time network feedback, preventing buffer bloat and maintaining smooth playback.

Quality of Service (QoS) and QoE: WebRTC supports adaptive video codecs such as VP8, VP9, and H.264, which can dynamically adjust frame rate and bitrate during transmission. However, maintaining consistent quality across heterogeneous networks remains a challenge. Experiments demonstrate that packet loss above 10% leads to visible frame drops, although forward error correction (FEC) mitigates minor losses.

Comparative Insights: Unlike HLS or DASH, which are optimized for scalability, WebRTC prioritizes immediacy and synchronization. Its decentralized nature reduces server dependency but introduces complexities in signalling and resource management. Despite these challenges, its unparalleled latency performance makes it the protocol of choice for telepresence, live sports commentary, and interactive classrooms.

2) Other Protocols

There are some protocols which are either not widely used or were introduced for a dedicated system or software, so when that system was shut down or got obsolete because of the inability to compete with other similar available systems, then with that, the respective protocols also lost their chance of further enhancements. Because of their lower popularity and availability, not a lot of work has been done on them, resulting in less available data for applying the testing or analysis to compare them with other protocols.

WebRTC supports adaptive video codecs such as VP8, VP9, and H.264, which can dynamically adjust frame rate and bitrate during transmission. However, maintaining consistent quality across heterogeneous networks remains a challenge. Experiments demonstrate that packet loss above 10% leads to visible frame drops, although forward error correction (FEC) mitigates minor losses.

a) Secure Reliable Transport

The very first protocol is SRT, which stands for Secure Reliable Transport. The services of data transmission migrated from dedicated networks to shared networks, and now to public data centers [12]. For such transactions, to meet the requirement of high-quality data transport over lossy infrastructure, SRT is designed. The lack of interoperability between different products results in binding users to use one vendor's family products and, hence, proprietary technologies supported by them. To resolve this, SRT is introduced with the goal of having all vendors support an interoperable way of communication.

According to [13], SRT is an open-source protocol that helps in delivering reliable streams, regardless of the network quality. The playback compatibility is limited (i.e. VLC Media Player, FF Play, Haivision Play Pro and some other players), which is comparatively not as good as other streaming

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

protocols. Talking about audio and video codecs, it is code-agnostic.

It provides less latency (i.e. <01 s) because it is built for controlling the latency and issues like jitter and packet loss over poor networks, and for recovering the packet loss, it uses its own method. For connection encryption, handshaking is implemented, and the encryption algorithm used by SRT is 256-bit AES. As the latency is very low, it is similar to Faster than Light (FTL) and WebRTC.

In [12], to check whether SRT is sufficiently good for being used in broadcasting applications, SRT is compared with Reliable Intern Stream Transport (RIST). The conclusion explains that RIST and SRT are feasible and good in some broadcasting applications, but some bugs are encountered while testing both. About SRT, the main point is that it can end up in a fail state, from which it cannot escape. It means that if the packet drop rate rises quickly, then it will end up in a fail state and will remain in that state even if the packet drop rate becomes low.

Microsoft Smooth Streaming: MST was originally introduced by Microsoft in 2008 to use with Silverlight player applications. MSS supports adaptive bitrate streaming along with some robust tools. It is popular only among Microsoft-focused developers, because it can support all Microsoft devices and users working with Xbox [14]. The video codecs available for MSS are H.264 and VC-1, while the audio codecs are AAC, MP3 and WMA. Its latency lies between 0 and 6 seconds. Considering the compatibility, it is compatible with Xbox, Silverlight player browsers, smart TVs and Microsoft and iOS [13].

HTTP Dynamic Streaming: HDS was developed for Flash Player, hence also called Flash-based streaming protocol. As in today's time, the support for Flash Player has become so weak, which makes HDS less popular and not widely supported [13]. Like RTMP, HDS provides adaptive streaming, but unlike RTMP, it does not provide low latency (i.e. the latency of HDS lies between 0 and 30 seconds). Supported audio codecs are AAC and MP3, and video codecs are H.264 and VP6 [15].

FTL Last but not least is FTL, which stands for faster than Light. FTL was developed by Mixer, a streaming platform, under Microsoft. Because of the inability of Mixer to scale as compared to other available alternatives, it was shut down. FTL is a real-time streaming protocol, which means sub-second latency is provided by FTL. It allows the users to communicate with others in real time, although it provides no virtual delay, but the quality of video gets compromised, as Mixer recommends decreasing the bitrate to & Mbps as compared to 10 Mbps supported by RTMP. It is supported by XSplit and OBS Studio streaming applications, and it is also pre-integrated in the Xbox One and Windows 10 operating systems. The audio codec supported by FTL is H.264, and the audio codec is Opus. Along with low quality, the other demerit of FTL is a lack of stability, because of being a new technology, it has not been under a lot of bug fixes. It is a push-based and Pull-based comparison.

There are many streaming protocols available today. It is very critical to choose the best protocol for any particular usage. To make it a little easier, a comparison is available on the basis of push-based protocols and pull-based protocols.

b) Push-based Protocols

In push-based streaming protocols, after the connection is established between the client and server, the server will keep transmitting the packets to the client until the session is interrupted or stopped by the client. For this, the server in push-based streaming maintains a session state with the client and keeps listening to the commands from the client about the session-state change. The most common protocol used in push-based streaming for session control is Real-time Streaming Protocol (RTSP). For data transmission, mainly, the Real-time Transport Protocol is utilized for push-based streaming. This RTP runs over the User Datagram Protocol, which has no built-in rate control mechanism. So, because of this, the rate at which the server pushes the packets depends upon the application-level client/server implementation instead of the transport layer protocol that is being used. This makes RTP a good fit for low-latency and best effort media transmission [3].

c) Pull-based Protocols

In pull-based streaming protocols, the client works as an active entity, and it requests content from the server. Therefore, the server will either be idle or blocked for a client until the client requests the content and then the server will respond whenever it receives the request from the client. The bitrate will depend on the available network bandwidth of the client for receiving the content from the server. The main protocol for pull-based media transmission is HTTP. The concept that is important to know on which the pull-based media streaming relies is progressive download. In progressive download, basically, the client starts pulling the data from the server after issuing an HTTP request to the server. When the minimum required buffer is filled, the client starts playing the media along with downloading the content from the server in the background. As long as the download speed is greater than the playback speed, the buffer at the client stays at a required level to continue with an uninterrupted playback.

III. EXPERIMENTAL SETUP

A. Core Component

The experimental setup of the system to implement the research work incorporates some technologies and some software. This section explains the key components of the system, which are given below:

1) Docker

Docker is an open-source project, and it was launched in 2013 [16]. The key concept implemented in Docker is containers. It

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

is a software platform which allows the user to build applications on the basis of containers. These applications run in a lightweight and small environment by using the Operating System (OS) kernel in a shared manner. Even using the OS in a shared manner, these applications work in an isolated way with one another and do not interrupt the main system's operation. The developers develop many applications, and it is not feasible to run them on different hosts, so most of them want their applications to be on the same host but in an isolated manner so that they do not intervene among themselves. A virtual machine (VM) can be one of the solutions, but every VM requires its own OS, and each of them needs static space in the system. It increases the load on the host machine and makes it slower. It is the main problem while working with the VMs.Another solution to this problem is the concept of containers implemented by Docker. Containers isolate each application and do not require any OS to run. Instead, the OS is shared with the host machine, and the resources used are much less as compared to VMs. Many containers can be packed on the same hardware.

The main components of Docker are:

- Docker file: A Docker file is a text file that is available in an easy-to-understand syntax, which contains the instructions to create a Docker image. All the important details are present in this file, which are required to run the Docker container. These details include the OS for the container, environment variables, languages, network ports and other required components.
- Docker Image: As mentioned above, the Dockerfile contains the details required for the Docker image and the basis of these details, the Docker image is created after the Docker build utility is invoked. It is a portable file which describes how the components of the container will run.
- Docker run: it is the utility which launches the container in actuality. Every container is an instance of a Docker image. Containers can be paused and restarted, and Docker run will launch the container from the same state where it was stopped. Multiple instances of the container can be run if the container has a unique name.

OBS Studio. Many live streaming tools are available for any level of production, from basic to high-end professional. OBS Studio is one of them which works virtually for any live streaming. OBS Studio is controversially the most used and known live streaming encoder. It is open source software and available for free. Starting with the basics, it has plenty of features which are useful in creating a professional live stream. It allows the user to record the video and connect to any video conferencing or live streaming platform by using any digital camera for live streaming. It is the most relevant streaming recording and streaming tool for all platforms. With integrated canvas preview, filters and source plugins, OBS Studio provides a comprehensive environment for professional streamers and other people to create well-managed video recording and broadcasting for Web services. OBS Studio can be installed onto the hard drive or can be run from removable media. By default, the settings are installed in:

- %app data%\obs- studio FOR Windows
- ~/.config/obs- studio FOR Linux
- ~/.obs-studio FOR Linux+ XDG
- ~/Library/Application Support/obs- studio FOROSX

To work efficiently, OBS Studio provides a feature of workspace control and quick scene management. It gives users the tools to keep the work on track and efficient. Canvas preview is also an important feature, which makes the user see all the changes made to the source's appearance and scene composition. Studio mode allows for previewing all the changes before being on-air in the scene. Layers are also an interesting aspect. Through layers, the work can be done on one element without affecting the others. It also helps in rearranging the elements by simply shifting the layer order in the source list. There are unlimited creative options available, like filters, plugin sources and transitions, etc. The appearance of every source can be changed by using the filters. Crop, chrome key, transform and many other visual effects are available by default. The best plugins can be selected from the online community, or new ones can also be written. Transitions can be added between the scenes to make them look more attractive. Multi-Track and Multi-Output are interesting new features. Multi-Track allows the user to save the audio sources in multiple tracks while recording. The number of tracks supported is 6. It is very useful because filters and effects can be added to the sound of the microphone after recording, the level of the microphone can also be adjusted later without affecting the sound, and all sounds stay in sync. In Multi-Output, the video can be streamed in one quality while making other recordings simultaneously. Means if the bandwidth is low, then the stream can be done on medium quality, and the video can be saved in full quality and then can be uploaded later in higher resolution. OBS Studio can be downloaded and installed from and more details are available over there.

2) NGINX

NGINX is an open-source software which is used for web serving, caching, reverse proxying, media streaming, load balancing, and more. It was originally introduced as a web server for providing maximum performance and stability. Apart from the HTTP server capabilities of NGINX, it can also work as a proxy server for email (POP3, IMAP and SMTP) and load balancer and reverse proxy for HTTP, TCP and UDP servers. NGINX was originally written by Igor Sysoev to solve the C10K problem, which describes the difficulty being faced by the existing servers in handling a huge number (the 10K) of concurrent connections (the C). With the asynchronous and event-driven architecture, NGINX became the fastest web server. This project was open-sourced in 2004. By seeing its exponential growth, Sysoev co-founded NGINX, Inc. It supported the continuous development of NGINX and marketed NGINX Plus as a commercial product. Probing further, in 2019, NGINX, Inc. became part of F5, Inc., and today, NGINX is known for handling hundreds of thousands of concurrent connections. With time and advancement in technology, websites have evolved from

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

HTML pages to multifaceted and dynamic content; similarly, NGINX has grown along with it.

B. Work Carried out

This section explains the work carried out. Its core components and objectives are already detailed in the above sections. This section mainly contains the flowchart of how the work has been done in a summarized way, while the whole architecture of the system is discussed in detail in the coming section. The subsections given below explain the whole procedure.

1) Basic Work-Flow of Experiment

The main components of the system setup, as mentioned above, are OBS Studio as streaming software and NGINX web server. Apart from these two, there are some other requirements to complete and analyze the work. All these components are connected in a specific manner. This specific manner is explained in this section. It is explained through some steps which are followed during the implementation. These steps are listed below:

Step 1: The very first step is that after getting the request from the client, the process is started by the server.

Step 2: The server will access the Docker file through Visual Studio Code (VS Code). Here, the Docker file that is being accessed is of the web server. The web server being used for the connection establishment is the NGINX server.

Step 3: VS Code will start the NGINX server through the Docker container and check if the connection is established or not.

Step 4: When the connection gets established, the next step is to proceed with OBS Studio.

Step 5: A stream will be created by OBS Studio. For recording or generating the video, it will require access to input devices. A specific input device can be attached to the host system for specific requirements.
Step 6: Now, the actual stream will be started by OBS Studio. The OBS Studio will start it through the web server that was started in the initial stage of the procedure.

Step 7: This stream is now available for the clients. The clients can be present on any device, including a website or any streaming software. Figure 1, given above, shows the basic workflow of the system, the basic flow of work that will be followed while implementing the protocols.

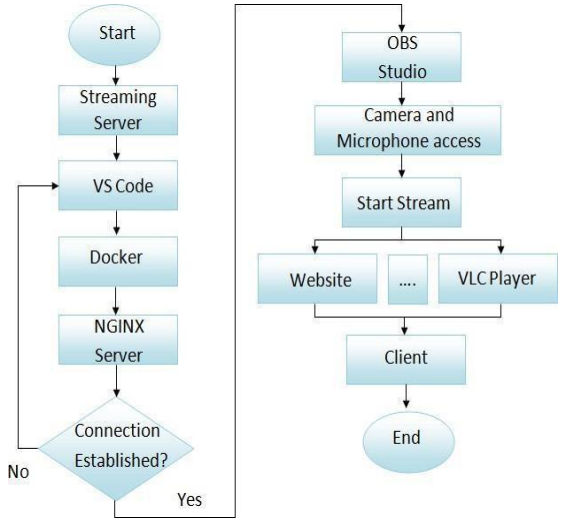


Fig. 1. Basic Work-Flow of Video Transmission between Two Parties

2) System Setup and Experiment

All the available protocols for streaming are listed and explained in detail in Section 1. The important developments regarding all of them are discussed in the next section. From all of the listed protocols, the ones that are selected for the implementation are:

- WebRTC
- HTTP Live Streaming Protocol
- Dynamic Adaptive Streaming over HTTP
- Real Time Messaging

The reason behind selecting only these protocols is that all of the above selected protocols are of different types. An attempt is made in the work to implement one protocol of each kind. For example, protocols cannot be completed without considering any HTTP-based protocol, and HLS is an HTTP-based protocol. Adaptive streaming is also an emerging field which has its own advantages. In adaptive streaming, the available bandwidth of the network is taken into account. It is not possible that the network is available in good conditions, such as high bandwidth and speed. Streaming a very high-quality video on low bandwidth available network will cause playback issues. So, to resolve this problem, in adaptive streaming, the video is made available in multiple resolutions and according to the specific bandwidth available, a particular resolution video will be selected for streaming.

DASH incorporates an adaptive streaming aspect. RTP is one of the traditional protocols. RTMP is also a traditional protocol, but it is the most used protocol since its introduction. The other most important part is WebRTC. It is the newest technology from all of the above-mentioned concepts. It is yet an emerging technology, but it is still providing great performance. All the selected protocols except WebRTC follow the same procedure and architecture. The basic workflow is explained in the previous section, and the architecture is explained in detail in the next subsection of this section. For WebRTC implementation, a video calling application is built. The implementation of WebRTC is

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

different from all the other protocols. After the implementation of these protocols, the analysis of their performance is done. Many factors are available on the basis of which this analysis can be done. But here, the main concern is latency.

3) System Architecture

The system architecture is shown below in Figure 2. It is clear from the diagram which are the main components of the system. The main components are:

1. Client and Server: The client in this work is present at a mobile phone, which is trying to access the created stream from there. The server is on the laptop, and through the built-in camera and microphone, the stream is being created.
2. VS Code: The main procedure starts with VS Code. It is running on the server to initiate the process. The program that is present here gets executed to achieve the objectives.
3. Docker: Docker is accessed by the VS Code program. The server that is responsible for the connection establishment is accessed through the Docker files of that server. There are containers which are containing the required files.
4. NGINX: NGINX is a server that is known for multiple concurrent connections. Here, for the DASH protocol, it is being used.
5. OBS Studio: the streaming software used in the work is OBS Studio. After the connection establishment, the streaming is started via OBS Studio. Instead of a live camera, the screen can also be shared in this software by specifically sharing a particular open program like Chrome, Word file or File Explorer, etc.

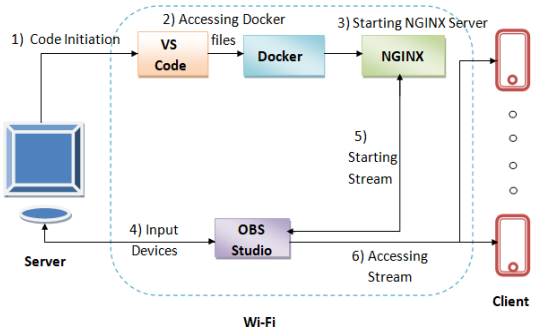


Fig. 2. System Architecture

IV. RESULTS & DISCUSSIONS

After analyzing the problem, the experiment is performed, and results are obtained from it. This section consists of the obtained results and a comparison based on the results.

Quality of Service (QoS) (to the point) For the performance measurement of a system, Quality of Service, which is called "QoS," plays an important role. To measure QoS for any system, some parameters are taken into consideration. The main parameter, delay, is playing a key role in the result evaluation of this research work. Figure 4 shows the parameters that influence the system performance.

A. Latency

HLS, WebRTC, DASH and RTMP are implemented as per the procedure discussed in the previous chapters. The theme of this research work is to find the best protocol out of the

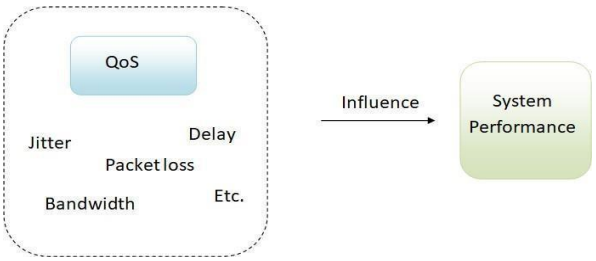


Fig. 3. QoS Parameters

selected ones according to the requirement. Delay is the most important requirement in this work. If we consider two transmission protocols, one is TCP, and the other is UDP, then this concept can be understood easily. UDP is considered an unreliable protocol, whereas TCP is the reliable one. In UDP, it keeps sending the packets without knowing whether they reached or not and which packet is lost, etc., so it fast as compared to TCP because in TCP, there is a delay. The reason is that TCP checks whether the packet is received at the destination or not. If not, then it re-transmits the data. That is why the delay is caused there. Terminology used under the delays concept includes Jitter, Buffer and Latency, etc. Jitter is a type of delay that is caused when the sending node is loading the whole data that is to be transmitted into the buffer. Whereas Latency is the total time taken by the whole transmission.

All four protocols are implemented to create a live stream. Latency can be minimized in many ways, but it is unsuitable for real-time communications. The impact of latency on on-demand videos is minimal, but for considering the live streaming, it is the curtail aspect. The network being used here is a Wi-Fi network. The objective is to find a protocol that provides live streaming with as little delay as possible because live streaming itself incorporates no delay, as the streaming is live only if there is no delay. If not, live as the HTTP streaming tries to make the streaming "near" live. This work captures the delay produced by all the different protocols. The calculated delay is explained for every protocol with the results. The available bandwidth is also an important factor in network characteristics.

It is important here because it is playing an important role in DASH implementation. Changing bandwidth has given rise to a new concept that is adaptive streaming, which is discussed in detail in the previous sections. It is forming a base for the need for a methodology that is implemented through the DASH protocol. All the recorded delays of the implemented protocols are given below one by one.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

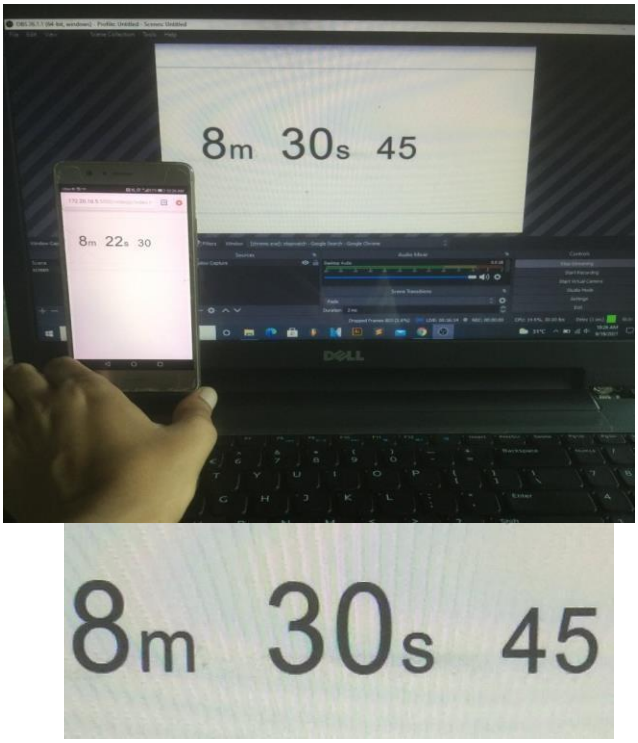


Fig. 4. HLS Latency Test 1

The first latency test for HLS Streaming is shown in Figure 4. It can be seen from the picture (Figure 4(a)) that the streaming is being done between a laptop and a mobile device that are connected to the same Wi-Fi. The IP address is clearly visible as 172.20.10.5 with port 5500.

Figure 4 (b) shows the timer on the laptop, whereas Figure 4 (c) shows the time on the mobile along with the IP address and the port address. This IP is going to be the same for the rest of the test.

Figure 5 shows the results of the second test for HLS Streaming. Similar to the first test, the three parts of the figure (a, b and c) show the same details.

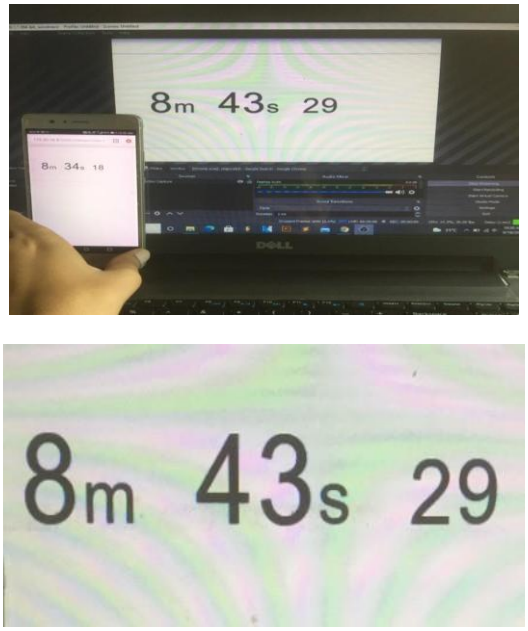


Fig. 5. HLS Latency Test 2

All parts of the above figure present the same information as in the first test. Three tests are conducted to note the latency in HLS streaming. Two are given above, and the third one is given below. The third test is shown in Figure 6 with the same three parts. Then a table is generated on the basis of the values of the three tests, and the average latency is calculated. Table 3 shows the average latency for HLS.

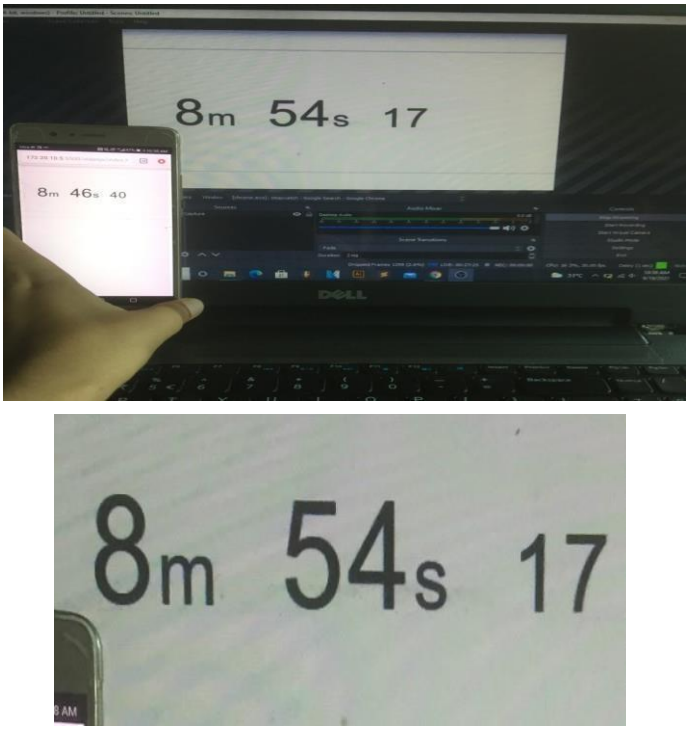


Fig. 6. HLS Latency Test 3

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

Stream run(Test)	Latency(s)
Stream1	8
Stream2	9
Stream3	9
Average Latency	8.6

Table 3 HLS Latency Table

Latency noted through DASH streaming in the first test is shown below in Figure 7. It can be seen from the given diagram that the stream is created via OBS Studio. The stream is being accessed through IP 172.202.10.5 and port 5501, as visible in Figure 7 (a).

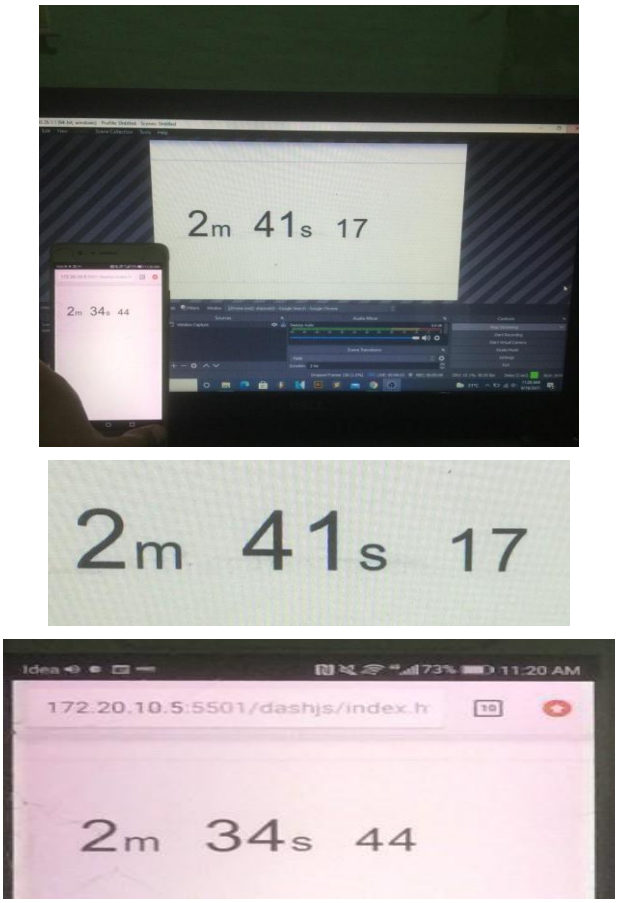


Fig. 7. DASH Latency Test 1

Figure 7 (b) shows the time at the laptop from where the streaming is being created, and the next figure(c) shows the time at the mobile. Similarly, test 2 and test 3 are conducted to note. The next figure (c)shows the time on the mobile. Similarly, tests 2 and 3 are conducted to note down the latency in DASH streaming. Test 2 is represented by Figure 8 (a), (b) and (c). Test 3 is depicted by Figures 9 (a), (b) and (c). All three parts of both the figures given below show the same information as the previous diagram in test 1. At the end, a table is generated for calculating the average latency in DASH streaming. From Table 4, the calculated average latency on the basis of conducted tests is 7.6 seconds.

The next figure (c) shows the time on the mobile. Similarlytest2andtest3areconductedtonote downthelatencyinDASHstreaming.Test2isrepresentedbyFigure 8(a),(b)and(c). Test 3 is depicted by Figures 9 (a), (b) and (c). All three parts of both the figures given below show the same information as the previous diagram in test 1. At the end, a table is generated for calculating the average latency in DASH streaming. From Table 4, the calculated average latency on the basis of conducted tests is 7.6 seconds.

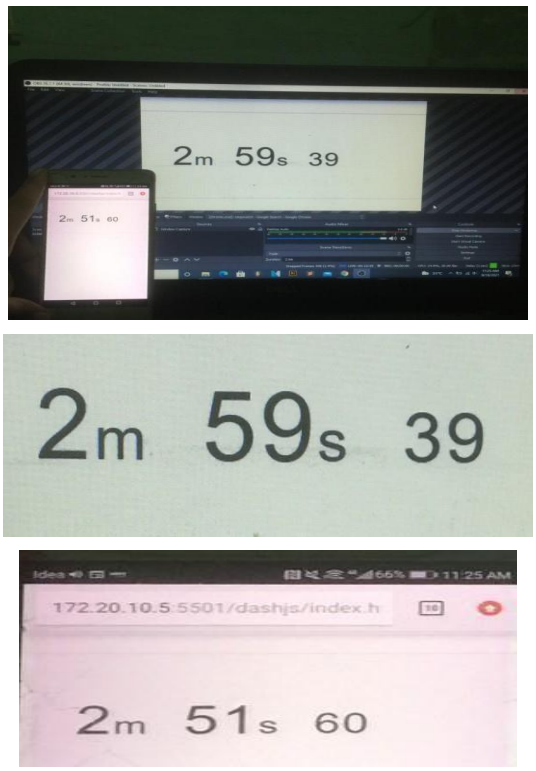
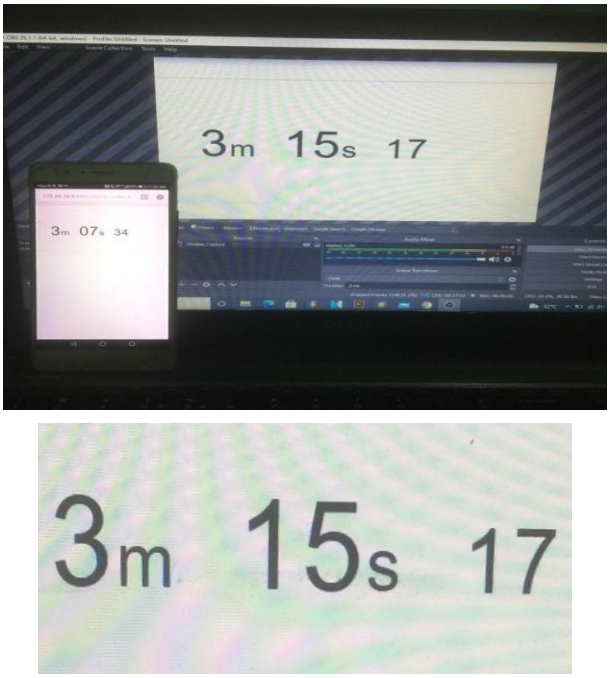


Fig. 8. DASH Latency Test 2



(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

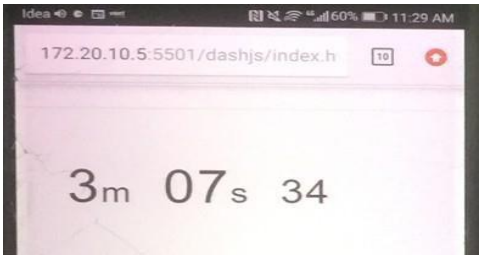


Fig. 9. DASH Latency Test 3

Stream run(Test)	Latency
Stream1	7
Stream2	8
Stream3	8
Average Latency	7.6

Table 4. DASH Latency Table

Latency while streaming through the WebRTC video call application is shown below in Figures 10 to 12. This video call can be between two users. Users can be on a laptop or a mobile device through their IDs, which will be generated whenever a new user connects. If the transmission is required between a mobile and a laptop, then both devices should be on the same Wi-Fi network.

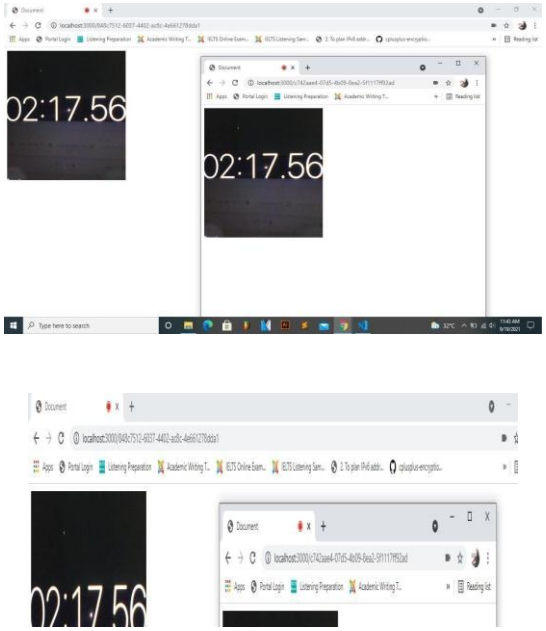


Fig. 10. WebRTC Latency Test 1

In the given diagram, Figure 10 (a), the communication is being done between the two windows, which are acting as two users. The two windows have different IDs, which differentiate them (Figure 10 (b)). Three tests are run to test the latency in WebRTC, which are shown by three figures, one of which is given above Figure 4.8, while the other two are shown below, Figure 11(a, b) and Figure 12(a, b). It can be clearly seen that the latency here is almost 0 as WebRTC provides sub-second latency.

Fig. 11. WebRTCLatencyTest2

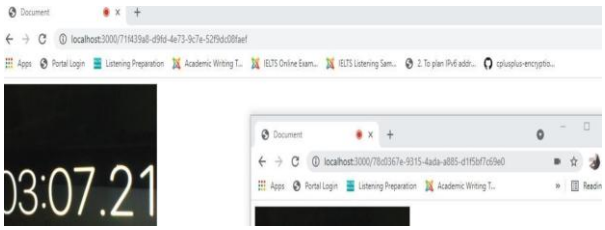


Fig. 12. WebRTC Latency Test 3

The last protocol that is included in this work is RTMP, and the latency noted through RTMP streaming is 6 seconds, which is shown in Figure 13. Here, the stream is being generated through OBS Studio and being accessed by the client on localhost, opened at the 8080 ports.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

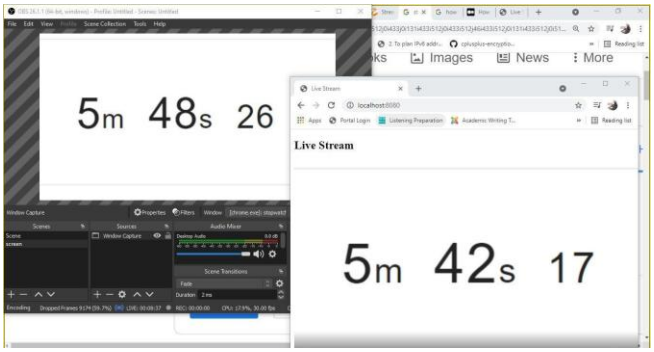


Fig. 13. RTMP Latency

A combined table consisting of calculated latencies of HLS, DASH, WebRTC and RTMP is given below. Three streams are run corresponding to DASH, WebRTC and HLS Protocols, which are shown in Table 5.

S. No.	Stream Run (Test)	Latency
1	HLSStream1	8
2	HLSStream2	9
3	HLS Stream3	9
4	DASHStream1	7
5	DASHStream2	8
6	DASHStream3	8
7	WebRTCStream1	0
S. No.	**Stream Run (Test)**	**Latency**
8	WebRTCStream2	0
9	WebRTCStream3	0
10	RTMP Stream	6

Table 5 Calculated Latencies of Protocols

It can be seen from the table that HLS, DASH, and RTMP have latencies between 7 and 9 seconds, while three-streams are done through WebRTC, providing a live stream with zero latency.

B. Other Considerations

Along with delay, some other parameters are also listed in the final comparison of the selected protocols. All these parameters are given in the final table shown at the end of this chapter. The parameters considered are discussed below.

1) Codecs

Then, the raw input or the recorded input cannot be transmitted as it is to the required destination. Before transmission, it is prepared for transmission. Some kind of processing is done to convert the input to a specific format. This processing is done through codecs. Different codecs are available for video as well as audio. A list of popular video code cs that are currently being used includes H264 Advanced Video Coding (H264/AVC), H264 Scalable Video Coding (H264/SVC), H264 Multiview Video Coding (H264/MVC), Real-Time Video (RTV), H265 High-Efficiency Video Coding (H265/HEVC) and VP8. Similarly, there are some audio codecs available for audio processing.

2) Network Adaptability

For any system to work efficiently, it must support the changes that can occur in future without causing any problem or as few problems as possible. It is also required that the additional requirement to support the changes should be as few as possible, meaning it should adapt to the changes with any new equipment. In this work, the one protocol that is based on this concept is DASH.

3) Changing Bandwidth

Changing bandwidth is a major property for any network. The bandwidth that is available for any network, it is very rarely on the same level all the time. Due to many reasons, it can vary. Like during weather changes, mobility, etc., can be the causes for this. How the new system copes with this issue is an important factor. The network that is being used in the work is Wi-Fi. So, the strength of any Wi-Fi network is not the same all the time. It can differ according to the nodes connected to it or due to other factors. The solution to this changing bandwidth is resolution variation

4) Video Resolution Variation

Video resolution variation is one of the solutions to changing bandwidth. If the strength of the network is good, that means a higher quality video can be streamed over this network. But if the strength gets degraded, then it is not possible to stream that much quality video on the same network. But a lower quality or lower resolution video can be easily transmitted to the network. This is the exact concept. In the DASH protocol, the video is made available in different resolutions. According to the bandwidth availability, the appropriate video is selected for the stream. These two factors are mentioned in the final table in the form of adapting capabilities.

C. Protocols Comparison

A table containing the comparison between all four implemented protocols is given below in Table 6. In this table, the Protocols are compared on the basis of Supported Audio and Video codecs, Adaptive Bitrate Support, Video Quality Scale, Calculated Average Latency (in seconds), compatibility, and Other Variants.

As the name suggests, the supported audio and video codecs fields list the different codecs supported by the protocol. Adaptive bitrate streaming tells whether the protocol supports adaptive bitrate or not. The next column considers the video quality provided by the protocols. Then comes the calculated average latency in the respective protocols. Compatibility is an important aspect for any protocol, which shows how compatible it is with other existing technologies. Another variant lists the available variants of the protocol. HLS protocol provides adaptive bitrate streaming and excellent video quality with an average latency of 8.6 seconds. HLS has better compatibility than the other three protocols. The quality of video in WebRTC is excellent, but it does not support adaptive streaming. But the major factor is that it provides live streaming with almost zero latency. DASH implements adaptive bitrate streaming and has excellent video quality. The average delay in DASH is 7.6 seconds. RTMP is providing a

(This work is licensed under a Creative Commons Attribution 4.0 International License.)

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 05 , May - 2026

latency of around 6 seconds, but the major drawback of RTMP is that it requires Flash Player.

REFERENCES

[1] H. Schulz Rinne, S. Casner, R. Frederick, and V. Jacobson, RTP: "A transport protocol for real-time applications", IETF RFC 1889, 1996.

[2] Y. Li, Q. Zhang, and J. Wang, "Comparative analysis of low-latency live streaming protocols," IEEE Access, vol. 8, pp. 45623–45635, 2020.

[3] R. Gutterman, S. Lee, and K. Park, "RTMP in the era of HTTP streaming: Legacy or lifeline," Proc. IEEE ICME, pp. 1254–1260, 2021.

[4] J. Xie, M. Sun, and H. Zhou, "Impact of QUIC on DASH and HLS streaming performance," Proc. IEEE INFOCOM, pp. 2231–2239, 2023.

[5] V. Patel, M. Shah, and J. Kumar, "WebRTC in metaverse platforms: Opportunities and challenges," IEEE Internet Computing, vol. 28, no. 2, pp. 32– 41,2024.

[6] Y. Yan, P. Huang, and C. Xu, "QoE metrics for video streaming in crowded networks", Proc. IEEE ICCCN, pp. 1–7, 2019.

[7] T. Baugher, R. Blom, D. Carrara, D. McGrew, M. Naslund, and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)," IETF RFC 3711, 2004.

[8] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: A survey," Computer Networks, vol. 47, no. 4, pp. 445–487, 2008.9

[9] R. Pantos and W. May, "HTTP Live Streaming," IETF Internet-Draft, 2017.

[10] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP: Standards and Design Principles," Proc. ACM MMSys, pp. 133–144, 2011.

[11] A. Singh and R. Sharma, "Performance evaluation of adaptive video streaming protocols," International Journal of Computer Applications, vol. 182, no. 45, pp. 12–18, 2019.

[12] R. Gutterman, S. Lee, and K. Park, "RTMP in the era of HTTP streaming: Legacy or lifeline" Proc. IEEE ICME, pp. 1254–1260,2021.

[13] Y. Chen, J. Liu, and F. Wang, "Error resilience in WebRTC for mobile networks," Proc. IEEE GlobeCom, pp. 1–6, 2021.

[14] L. Wang, H. Zhang, and Q. Liu, "Deep learning–driven QoE optimization for DASH," IEEE Transactions on Multimedia, vol. 26, no. 4, pp. 1156–1168, 2024.

[15] R. Krishnan, D. Lee, and S. Agarwal, "Evaluating protocols for live esports streaming: QoE and latency trade-offs," Proc. IEEE ICME, pp. 1432–1437, 2022.

[16] L. De Cicco, S. Mascolo, and V. Palmisano, "Feedback control for adaptive live video streaming," Proc. ACM MMSys, pp. 145–156, 2013.

(This work is licensed under a Creative Commons Attribution 4.0 International License.)