

Aarya Tapaswi

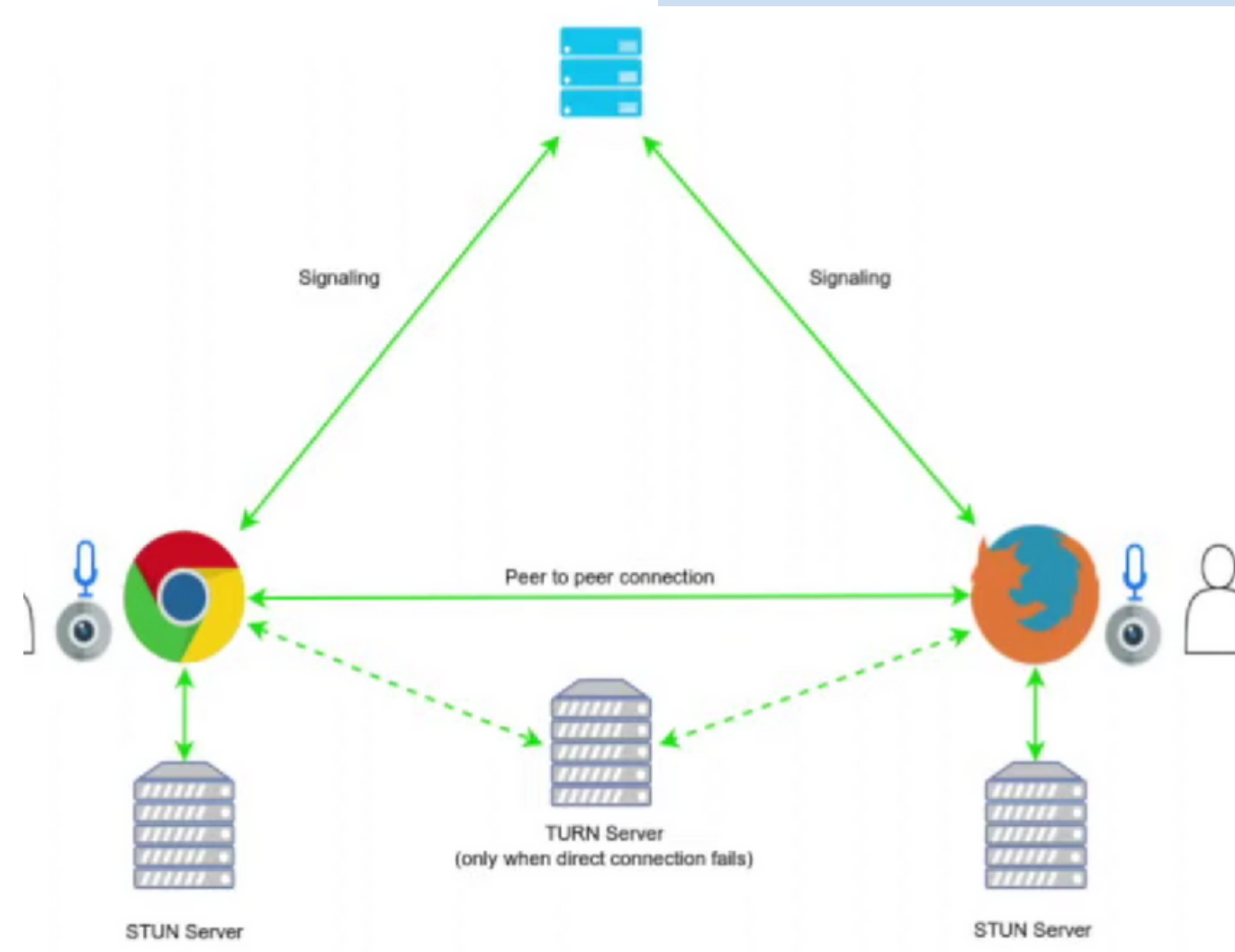
# LIST OF CONTENTS

- 01 ICE-TURN-STUN**
- 02 WHEN IS TURN USED**
- 03 HOW TURN WORKS**
- 04 VIDEO COMPRESSION**
- 05 VIDEO CODEC FORMATS**
- 06 CODECS AND FORMATS**
- 07 WEBRTC TOPOLOGIES**
- 08 HYBRID TOPOLOGY**
- 09 TCP VS UDP**
- 10 RTC VS HTTP**

# ICE-STUN-TURN

## STEP 1: CANDIDATE GATHERING

When two peers want to establish a connection, they gather a list of potential network addresses (candidates) through which they can communicate. These candidates include local IP addresses, public IP addresses, and port numbers. A STUN (Session Traversal Utilities for NAT) server is utilized in this step. Each peer sends a request to the STUN server, which responds with its perception of the peer's public IP address and port. This information helps peers understand how they appear to the outside network.



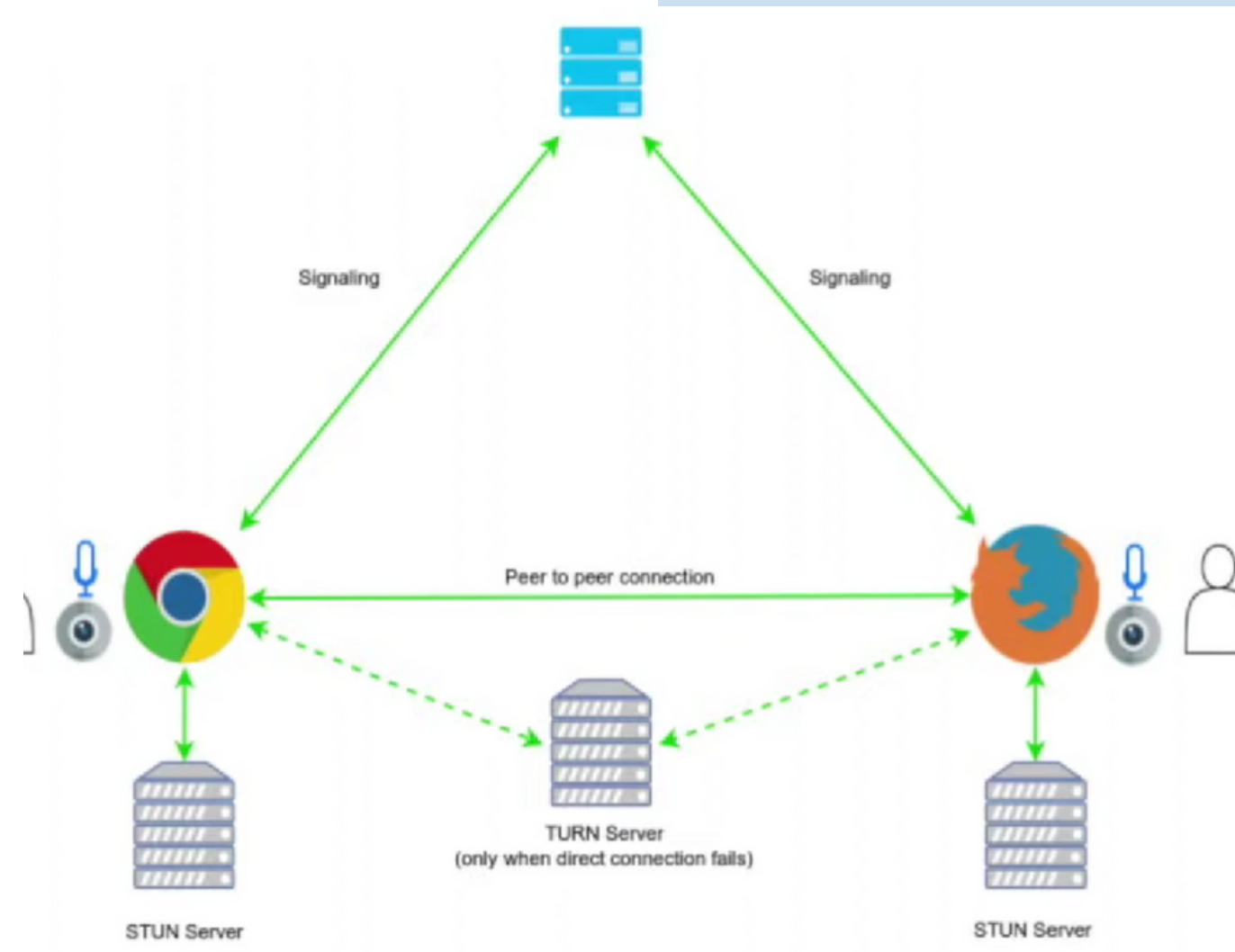
## STEP 2: NAT TRAVERSAL

Many devices are behind Network Address Translators (NATs) and firewalls, which complicate direct communication between peers. The gathered candidates often include private IP addresses that are not directly reachable from the public internet. ICE employs various techniques to bypass these NATs, such as UDP hole punching, which temporarily opens a pathway through the NAT. The candidates gathered from the STUN server and other sources are exchanged between peers.

# ICE-STUN-TURN

## STEP 3: CONNECTIVITY CHECKS

Peers initiate connectivity checks using the exchanged candidates to determine which paths are viable for communication. They send STUN requests to each other, using the candidate information, and monitor the responses. Successful responses indicate that the candidate can be used for communication. These checks continue for various candidate pairs until a successful connection is established. If a direct connection cannot be made due to restrictive NATs or firewalls, TURN servers come into play.



## STEP 4: FALLBACK TO TURN

If direct connectivity checks fail due to restrictive network conditions, the peers fall back to using a TURN (Traversal Using Relays around NAT) server. A TURN server acts as a relay, forwarding data between peers. The peers send their traffic through the TURN server, which relays the data to the other peer. While TURN introduces extra latency due to the relay, it ensures that communication can occur even in the presence of severe NATs or firewalls. TURN servers are used as a last resort when direct peer-to-peer communication is not possible.

# TURN

*When is it used?*

## **SYMMETRIC NATS:**

They map different internal IP-port pairs to different external IP-port pairs for each destination.

## **FIREWALLS AND PROXIES:**

firewalls or proxies are highly restrictive, they might block the types of traffic necessary for direct peer-to-peer communication.

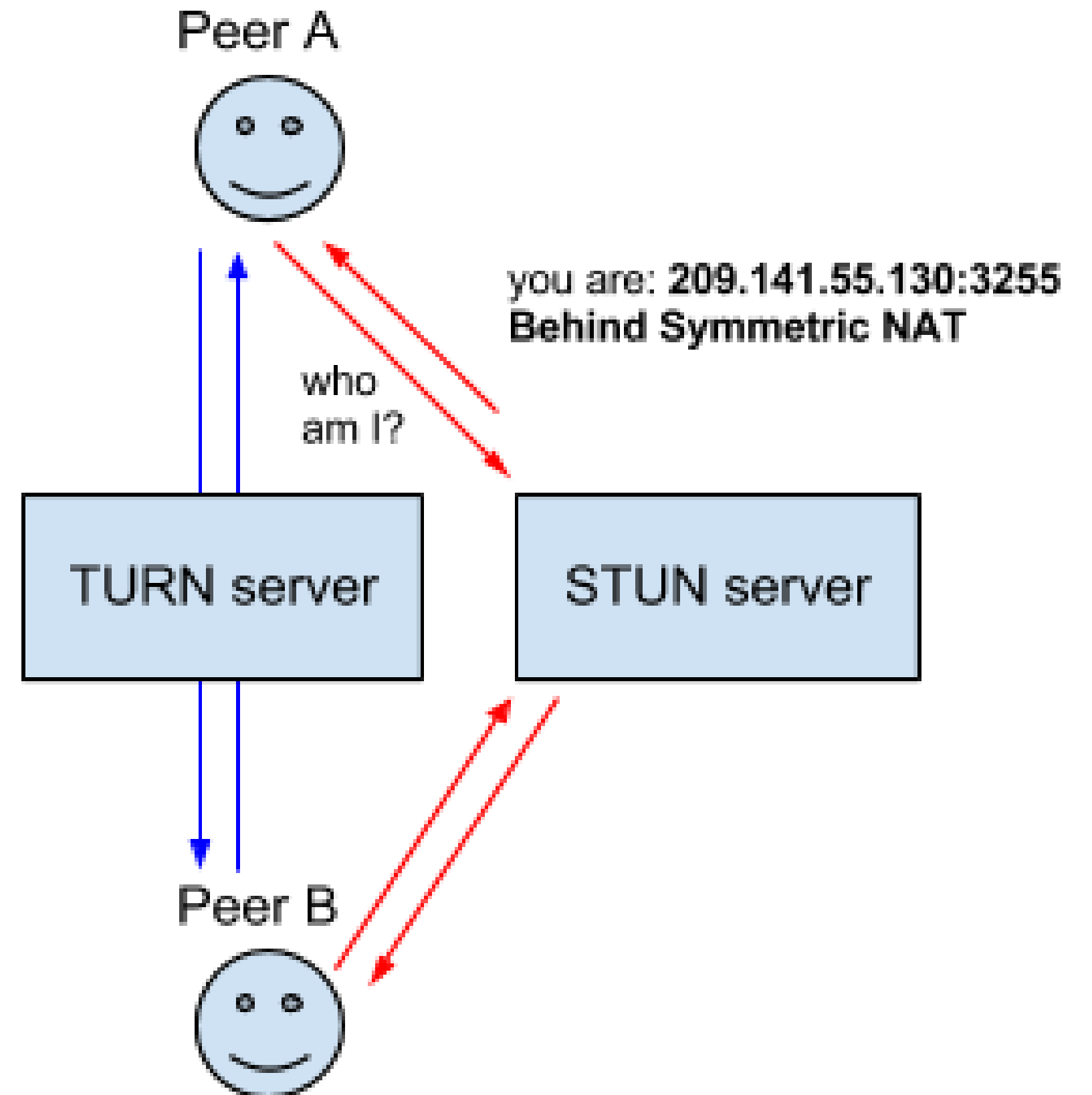
## **PEER MOBILITY:**

When peers change networks (e.g., moving from Wi-Fi to cellular data), their IP addresses can change abruptly.

## **UNREACHABLE PEERS AND NAT RESTRICTIONS:**

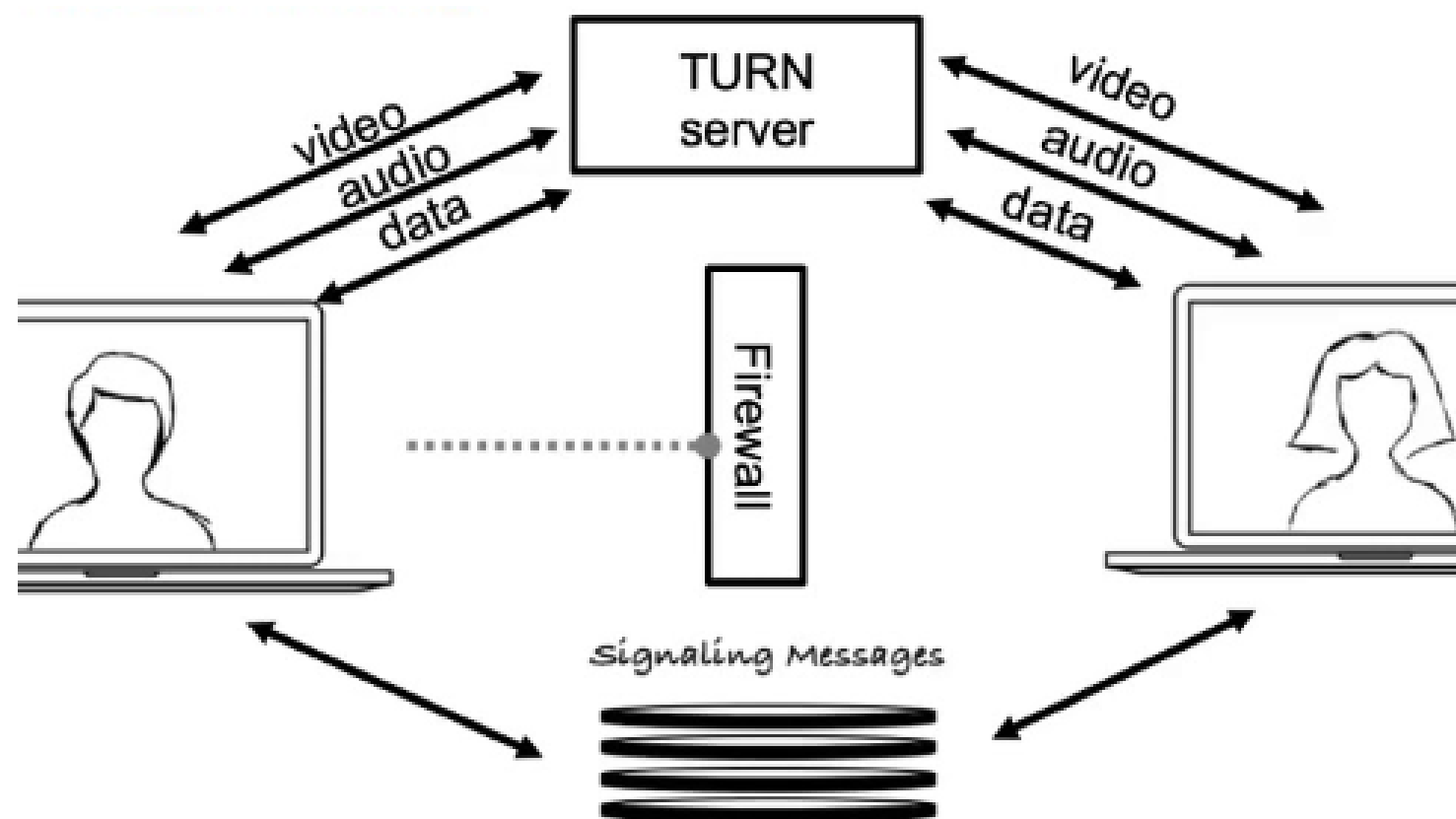
certain NATs might still cause issues with direct communication due to inconsistent behavior or limitations.

## **SECURITY AND PRIVACY**



# TURN

## *How it works*



**1.** Both Peer A and Peer B are aware of a shared TURN server's address. A TURN server is a publicly accessible server that acts as a relay for their communication. The TURN server has a public IP address and is reachable by both peers.

**3.** Peer A has a relayed transport address provided by the TURN server. Peer A informs Peer B about this relayed address. When Peer B wants to send data to Peer A, it sends the data to the relayed transport address of Peer A on the TURN server. The TURN server receives the data and relays it to Peer A using the allocated relayed transport address. Similarly, if Peer A wants to send data to Peer B, it uses the relayed transport address provided by the TURN server.

**2.** Peer A initiates the process by sending an allocation request to the TURN server. This request typically includes a request for a relayed transport address. In response, the TURN server assigns a relayed transport address (public IP and port) for Peer A to use.

**4.** The data flows in a triangular path: Peer B sends data to the TURN server, the TURN server relays the data to Peer A, and vice versa.



The metadata and compressed video data encoded with the codec are stored in the container file

# *video* **COMPRESSION**

encoding and decoding video data using different codecs to achieve desired formats, sizes, and qualities.

Video compression is the process of reducing the number of bits needed to represent a video without compromising its visual quality to facilitate its transmission over the Internet.

Codecs filter out additional data, such as frames of similar images, by grouping them into categories thus eliminating unwanted frames of data and encode the remaining data together, resulting in a loss of data bits and a reduction in file size.

## **01 Advantages**

- Requires less storage space
- Faster file reading/writing
- Faster file transfer

## **02 Disadvantage**

- Resolution drops
- Extra resources needed to compress /decompress

## **03 Codecs**

- H.264 (AVC)
- H.265 (HEVC)
- H.266 (VVC)
- MPEG-2
- AV1
- VP9
- DIVX
- XVID
- QuickTime
- WMV

# VIDEO CODEC FORMATS

1. H.264 (AVC): H.264, also known as Advanced Video Coding (AVC), is a widely used video codec that offers a good balance between video quality and compression efficiency. It's used for a variety of applications, including video streaming, broadcasting, and video conferencing. H.264 achieves efficient compression by analyzing motion between frames and using predictive coding techniques.
2. H.265 (HEVC): H.265, or High Efficiency Video Coding (HEVC), is an advanced video codec designed to improve compression efficiency compared to H.264. It offers better video quality at lower bit rates, making it suitable for high-resolution video and 4K streaming. HEVC is used in applications where bandwidth and storage are critical factors.
3. H.266 (VVC): H.266, also known as Versatile Video Coding (VVC), is a successor to H.265. It aims to provide even higher compression efficiency, allowing for reduced data sizes while maintaining quality. VVC is particularly useful for applications requiring high-quality video at low bit rates.
4. MPEG-2: MPEG-2 is an older video codec that was widely used for DVD and early digital television broadcasting. While it offers good video quality, it's less efficient in terms of compression compared to modern codecs like H.264 and H.265.
5. AV1: AV1 is an open-source video codec developed by the Alliance for Open Media. It's designed to provide excellent compression efficiency and video quality. AV1 is gaining popularity for online video streaming platforms due to its ability to deliver high-quality video even at low bit rates.

**READ MORE**

1. VP9: VP9 is an open-source video codec developed by Google. It's designed to be a competitor to H.265 and AV1, providing efficient compression and good video quality. VP9 is often used for web-based video streaming and is supported by major web browsers.
2. DIVX: DIVX is a video codec known for its high compression and good video quality. It gained popularity in the early 2000s for creating compressed video files that maintained a relatively small file size while preserving reasonable quality.
3. XVID: XVID is another video codec that gained popularity around the same time as DIVX. It's an open-source codec designed to offer MPEG-4 compatibility with efficient compression and decent quality.
4. QuickTime: QuickTime is a multimedia framework developed by Apple. It supports various video codecs, including H.264 and H.265, and is often associated with the .mov file format. QuickTime has been widely used for video playback and editing on Apple devices.
5. WMV: Windows Media Video (WMV) is a proprietary video codec developed by Microsoft. It's used for video compression and is often associated with the .wmv file format. WMV was commonly used for streaming and playback on Windows-based systems.



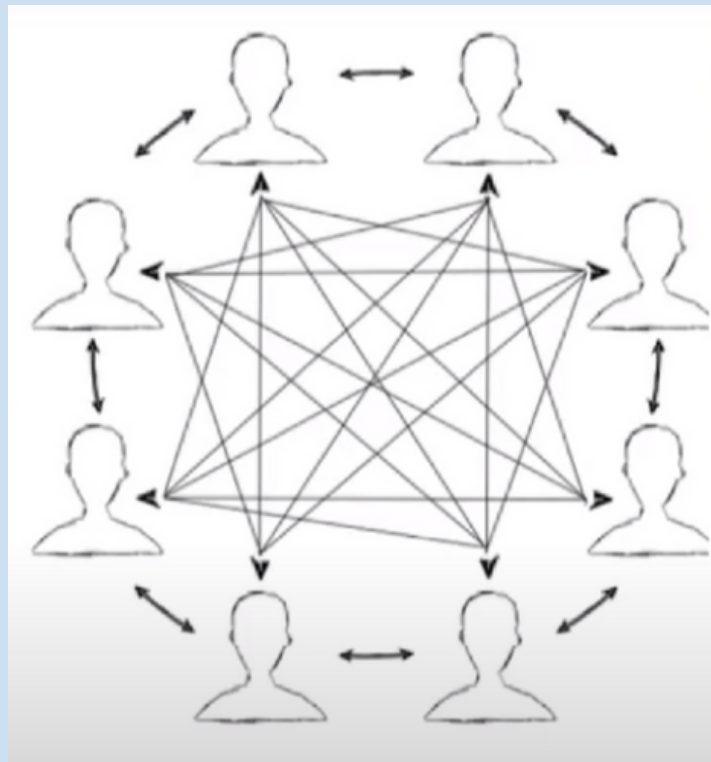
# CODECS AND FORMATS

## Browser Compatibility (Video & Audio codecs)

1. Chrome: Video (H.264, VP8, VP9), Audio: AAC, MP3, Vorbis, Opus
2. Firefox: Video (264, VP8, VP9 AAC), Audio: MP3, Vorbis, Opus
3. Internet Explorer: Video (H.264), Audio: MP3
4. Safari: Video (H.264), Audio: MP3, AAC
5. iPhone: Video (H.264), Audio: MP3, AAC
6. Android: Video (H.264, VP8), Audio: AAC, MP3, Vorbis
7. Chrome Android: Video (H.264, VP8, VP9), Audio: AAC, MP3, Vorbis, Opus

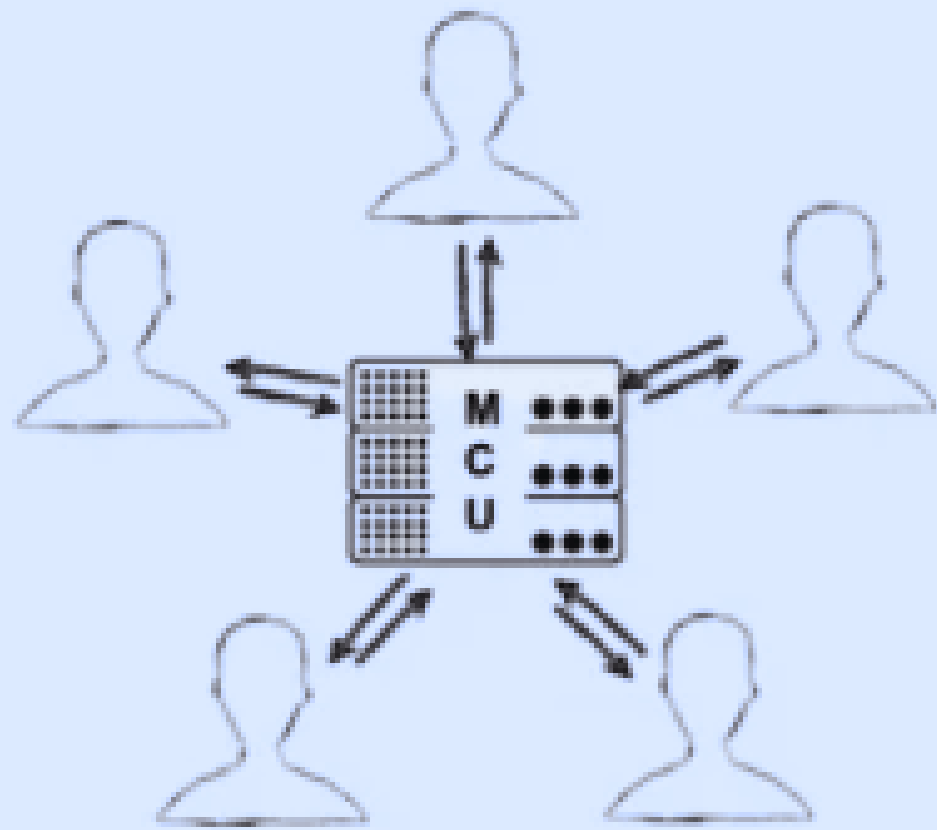
- Video: Based on the breakdown above, H.264 is a format that is more universally supported across different browsers. There are very little gains in switching to VP8 or VP9, unless your end device requires these formats (the original version of WebRTC supported just VP8).
- Audio: You might run into situations where the video works, but the audio simply does not. In our experience, this could happen with Firefox, Safari, IE-Edge, or Opera. As the table above shows, AAC finds more universal adoption (versus the default Opus).

# WebRTC TOPOLOGIES



- Best suitable for 2-4 participants
- Need powerful client.
- Cost Effective
- No server load
- Secure

## Peer-to-Peer



# SFU

1. When a participant sends their media stream to the SFU, the participant's device can adaptively encode the stream at different bitrates.
2. The SFU receives the multiple bitrates of the same stream from the participant.
3. The SFU analyzes the network conditions of the recipients. It determines their available bandwidth, packet loss rate, and other factors affecting the quality of the stream.
4. The SFU then intelligently selects the appropriate bitrate variant of the participant's stream to forward to each recipient. Participants with higher bandwidth and better network conditions may receive higher bitrate streams, while those with limited bandwidth receive lower bitrate streams.

# MCU

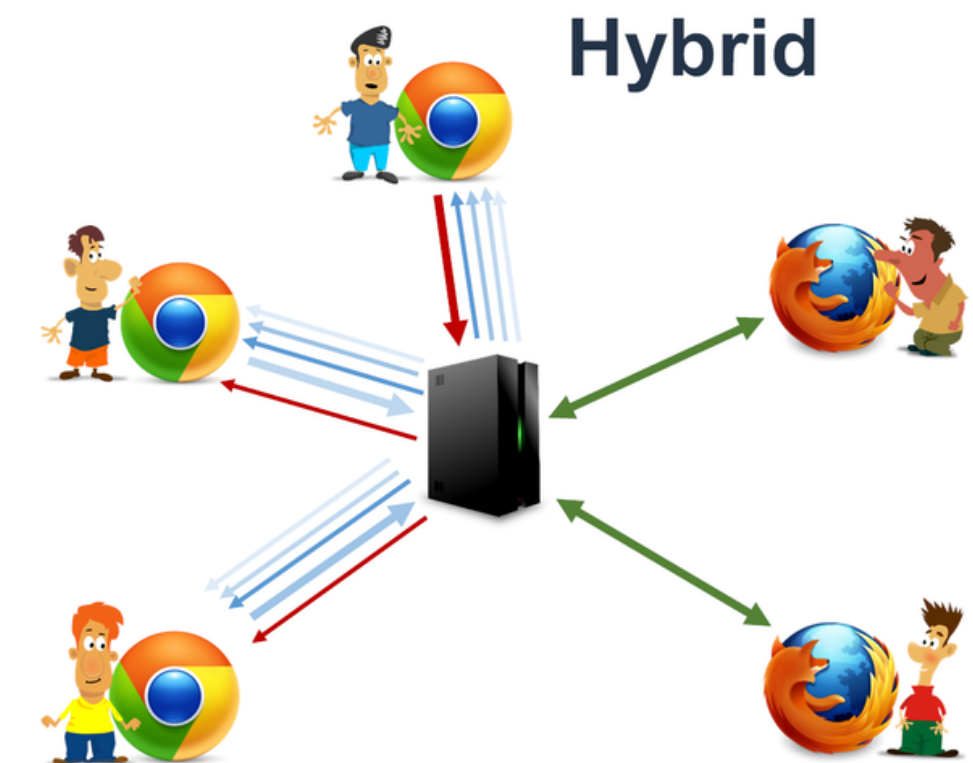
1. In contrast, MCUs tend to mix all incoming streams into a single composite stream before sending it back to participants. In most MCU setups, each participant's stream is typically mixed and encoded into a single common bitrate for the entire conference. This common bitrate is then distributed to all participants.
2. Participants in MCU setups generally receive the same mixed stream, regardless of their specific network conditions.
3. While some advanced MCUs might support adaptive bitrate techniques to a certain extent, the inherent nature of mixing multiple streams into a single composite stream can make achieving fine-grained flexibility in bitrates more challenging compared to the inherently stream-based approach of SFUs.






# HYBRID TOPOLOGY

Paper

Paper



Parameters	Full-Mesh	SFU	MCU
Definition	Direct exchange of media (audio/video /data) content between browsers includes NAT and firewall traversal using ICE, STUN and TURN techniques.	With SFU every participant sends his/her media stream to a centralized server (SFU) and receives streams from all other participants via the same central server. So acting like as router who is routing the require streams.	The MCU server is using transcoder (includes decoder followed by encoder) and mixer to create a single stream which get shared to all participants.
Media stream flow between server and browsers (client)			
5 participant (N = 5) A Num of connection in conference B Num of connection per user C Client end: Upstream D Client end: Downstream E Client end : CPU load F Server end : CPU load	20 8 4 4 ~65% (N-1:N-1) 0%	25 5 1 4 ~60% (1:N-1), SFU with Simulcast = ~80%. ~10% (only for routing logic)	10 2 1 1 ~20% (1:1) ~80% (for transcoding + mixing)
Bandwidth requirement End-to-End Latency Video Layout Cost of the solution	High Lowest Client decide Inexpensive	Medium Low Client decide Moderate server cost	Normal Accetable Server decide Expensive
Frameworks/Libraries  Examples	Hubl.in  Skype, Facetime	Jitsi-meet, MediaSoup, Janus, Medooze, Kurento  Talky, Vidyo, Trueconf	Kurento, Ant-media, Open-Vidu, FreeSwitch, Frozen Mountain  Fuze, Dialogic



# UDP vs TCP

Factor	TCP	UDP
Connection type	Requires an established connection before transmitting data	No connection is needed to start and end a data transfer
Data sequence	Can sequence data (send in a specific order)	Cannot sequence or arrange data
Data retransmission	Can retransmit data if packets fail to arrive	No data retransmitting. Lost data can't be retrieved
Delivery	Delivery is guaranteed	Delivery is not guaranteed
Check for errors	Thorough error-checking guarantees data arrives in its intended state	Minimal error-checking covers the basics but may not prevent all errors
Broadcasting	Not supported	Supported
Speed	Slow, but complete data delivery	Fast, but at risk of incomplete data delivery

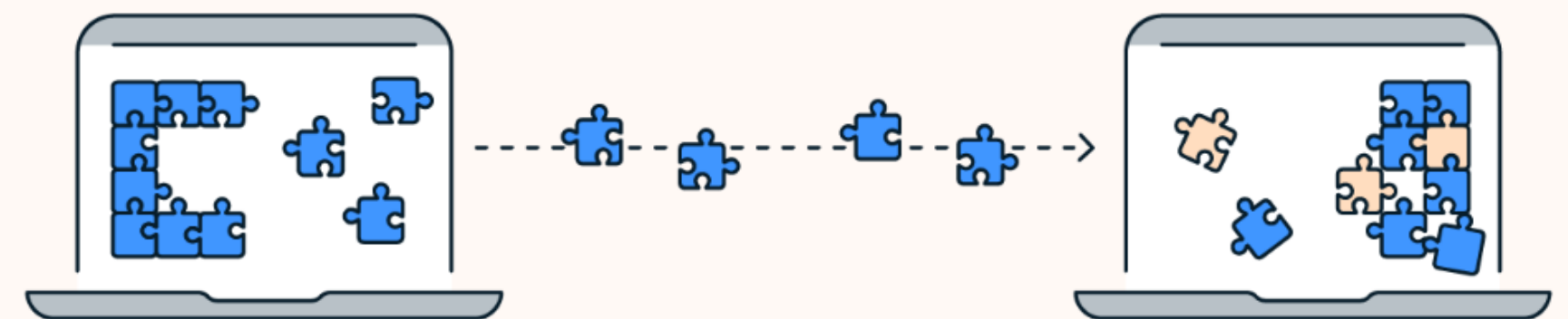
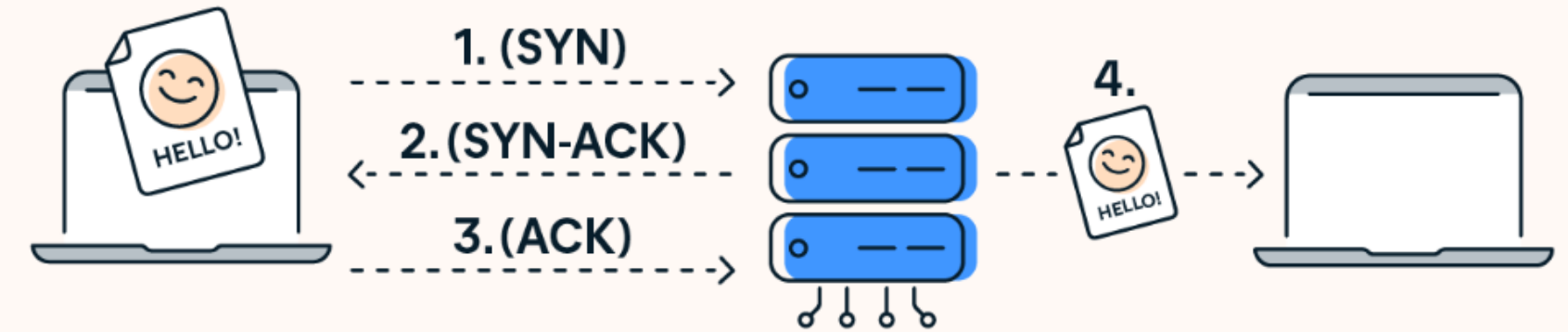
# TCP VS UDP

TCP is best for:

- Email or texting
- File transfers
- Web browsing

UDP is best for:

- Live streaming
- Online gaming
- Video chat

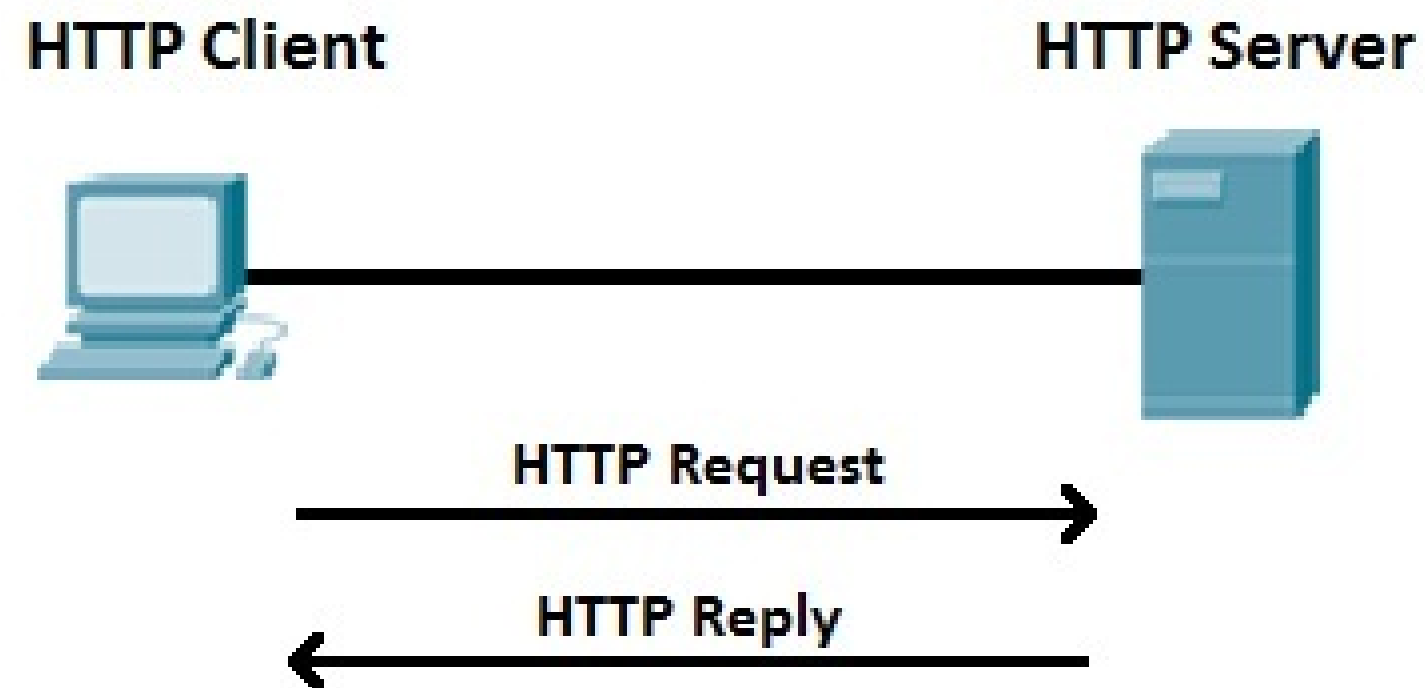


RTC  
VS  
HTTP

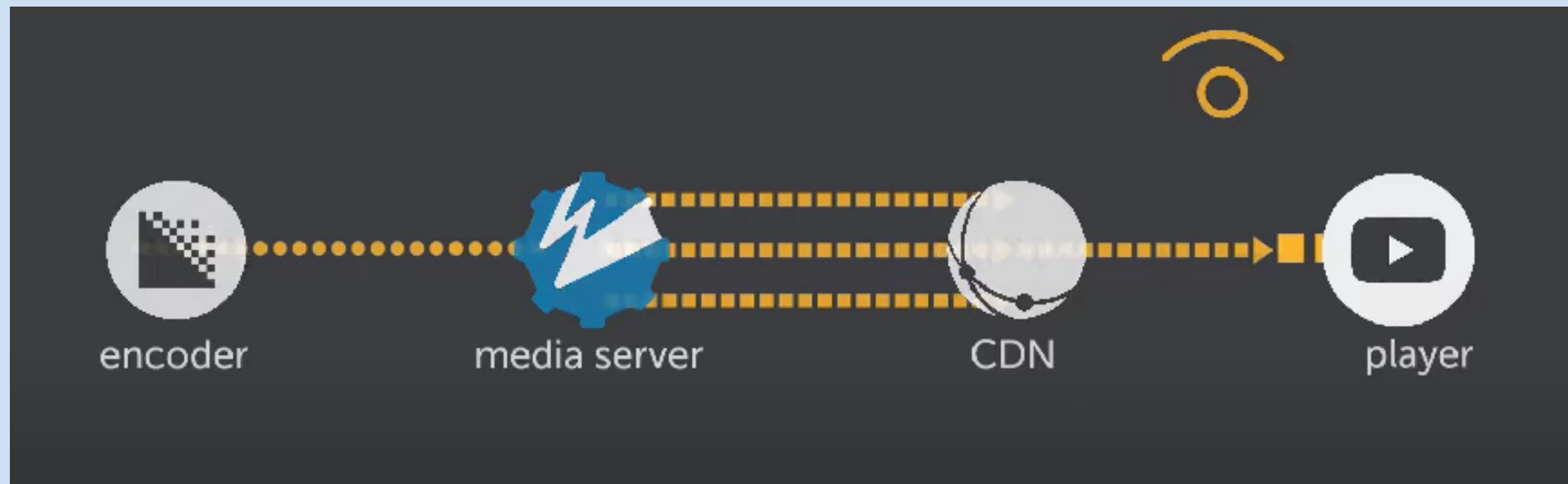
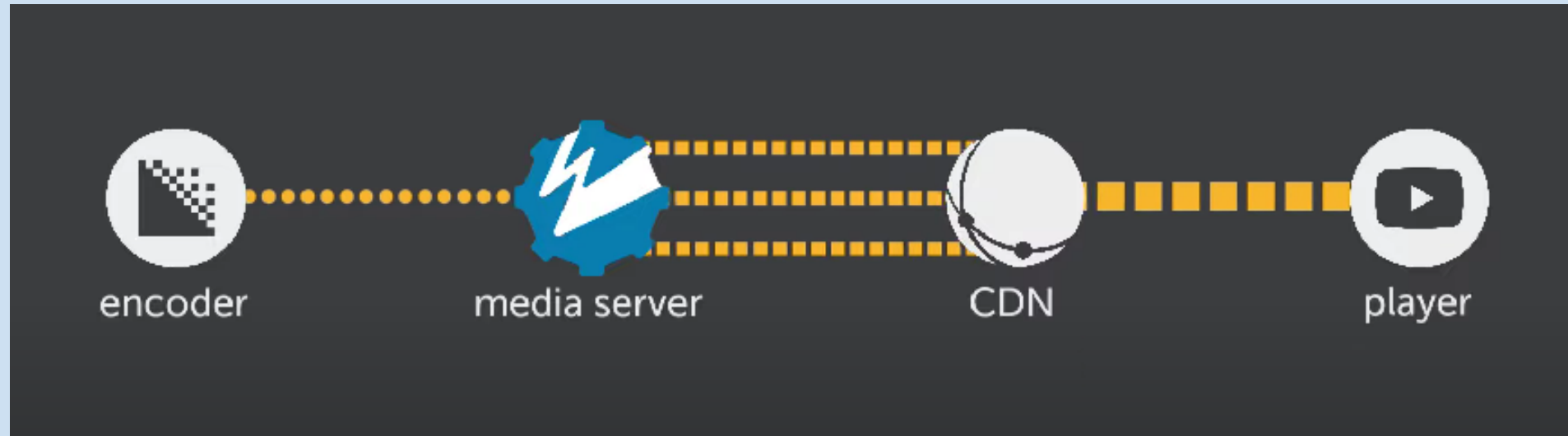
HTTP-based protocols are regular web servers sending out progressive downloads. They mainly use adaptive bitrate streaming to supply the best possible quality and viewing experience, regardless of connection, software, or device. Some standard HTTP-based protocols include MPEG-DASH, Apple's HLS, Adobe HDS, and Microsoft Smooth Streaming.

HTTP based streaming is in essence a very simple approach: you chop a video up in different files and serve everyone as if it was a static asset. In this sense, it turns live streaming into a sequence of files being loaded in exactly the same way as if loading images or scripts on a website.

In essence, WebRTC is a lot more advanced and a lot more complex compared to standard server to client HTTP based streaming. This makes sense. With WebRTC connections are set up between clients and in normal setups, servers are only used to coordinate peers connecting to each other.



# HLS



## Disadvantage

- HLS is primarily designed for one-way streaming, making it less suitable for interactive applications where real-time communication is required.
- HLS requires storing multiple encoded versions of the same content to support adaptive streaming. This can result in higher storage costs for content providers and increased bandwidth consumption for both content providers and viewers.
- HLS segments video and audio content into small chunks, and each chunk has its own HTTP request/response cycle. This can introduce overhead in terms of server load and network requests, potentially leading to inefficiencies.

# HTTP in VC

HTTP can be used for signaling and control purposes in video conferencing applications. This includes tasks such as session initiation, user authentication, and establishing communication parameters. For example, web-based video conferencing platforms often use HTTP for these purposes.

## BUT...

1. HTTP operates on a request-response model, where a client sends a request to a server, and the server responds. This model is inherently not designed for real-time communication because it introduces latency due to the time it takes for the request to reach the server and the response to return. Real-time communication requires a continuous, low-latency data stream, which HTTP does not provide.

HTTP is a stateless and connectionless protocol, meaning that each HTTP request is independent of previous requests. In contrast, real-time communication systems, such as video conferencing, require a persistent connection to transmit data continuously without interruption. The connectionless nature of HTTP makes it inefficient for maintaining a continuous audio and video stream.



