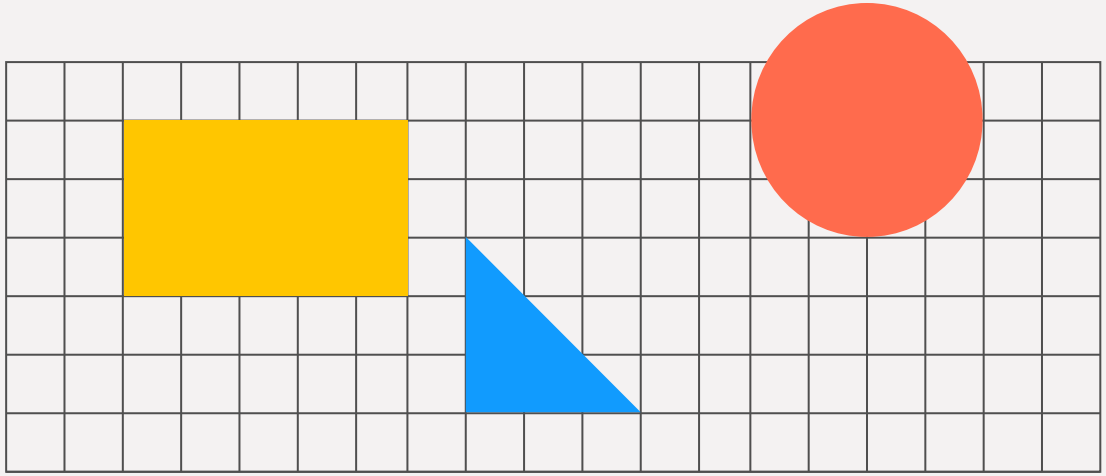
Partially Reliable Video Live Streaming Based on QUIC in Mobile Platform

Rendy Arya Kemal - 2106639945
Aushaaf Fadhilah Azzah - 2106630063
Rahfi Alyendra Gibran - 2106705764

Supervisor:
Made Harta Dwijaksara, S.T., M.Sc., Ph.D.



Introduction

Live Streaming

Over time, live video streaming services have become increasingly popular, with applications in numerous sectors of daily life. As a result, they have become a significant aspect of our daily lives.

In previous research, there have been successful efforts in **simulating** live stream scenarios using Media over QUIC as the transport between server and client. Thus, we will implement and evaluate whether the existing architecture is viable to do actual live stream.

Mobile

Mobile internet has been on the rise since the 2010s. By mid-2016, mobile internet traffic globally overtook desktop traffic and has been growing ever since. Between 2009 and 2024, global mobile internet traffic skyrocketed from **0.72% to 62.99%**,

Therefore, we decided to bring the existing WARP Media over QUIC Transport (MoQT) to mobile platforms. This way, we can see how well it works on mobile

Introduction

Reliable and Unreliable QUIC

The QUIC protocol supports two transport modes: reliable and unreliable, named "stream" and "datagram" respectively. In previous research, both implementations of the modes have been done successfully, with the addition of hybrid mode, a partially reliable transport mode that made use of both stream and datagram at the same time. However, the implementation is not perfect due to the characteristics of datagrams.

The advantages of QUIC in this case is its ability to switch between modes without the need of connection migration, as we can listen to both at the same time.

In this research, we will work further on hybrid mode, as that makes use of both reliable and unreliable transport, and discuss our approach on improving it, as well as evaluating our implementation.

Problems

Live Stream

Can the proposed infrastructure improve the existing MoQT functionality to accommodate live stream scenarios?

Hybrid Improvement

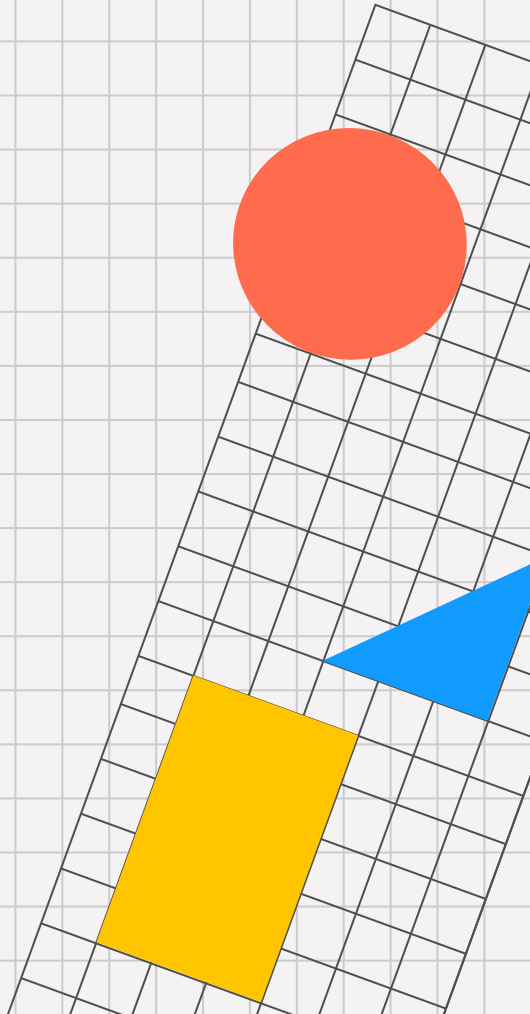
Can the proposed hybrid transport mode improve live streaming Quality of Service (QoS) performance?

Mobile

Can the proposed infrastructure and scenario be implemented in mobile devices? How is the overall performance compared to Desktop?

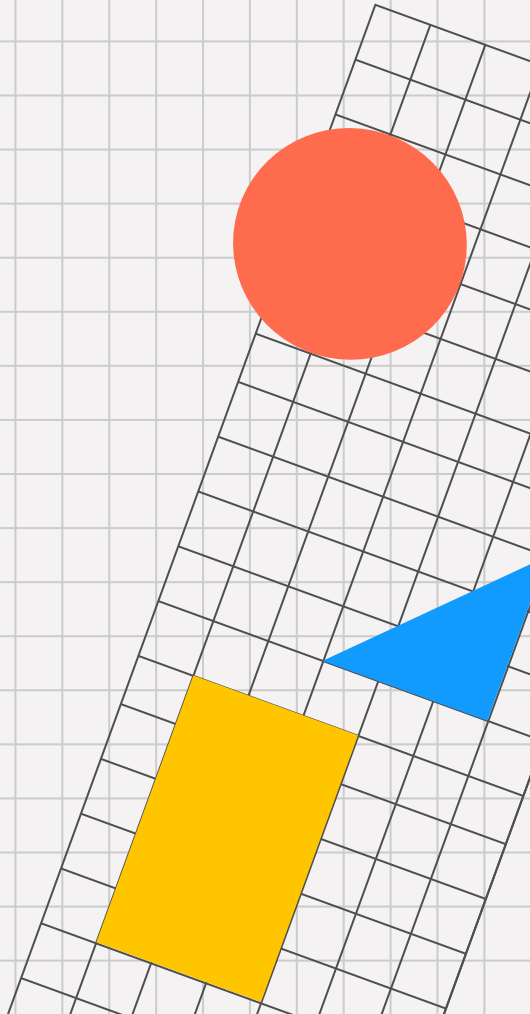
Scope

1. The main scope of the problem that will be focused on is MoQT for fragmented MP4 media.
2. The development focus will be on improving WARP MoQT from the previous research.
3. Mobile platform testing will be performed on an Android device.
4. The live stream media in this research will refers to video that is streamed to the server, much like YouTube Live and Twitch.
5. Live stream server can only process 1 live stream at a time.



Purpose

1. Evaluating the performance of the protocol in live streaming situations.
2. Design a better hybrid transport mode implementation than the current version.
3. Implement and evaluate The WARP MoQ on Android platform.



Roles

Rendy Arya Kemal

- Live stream infrastructure implementation
- Testing and data collection

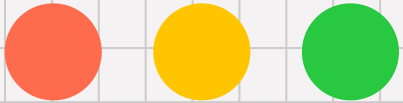
Aushaaf Fadhilah Azzah

- Hybrid mode improvements

Rahfi Alyendra Gibran

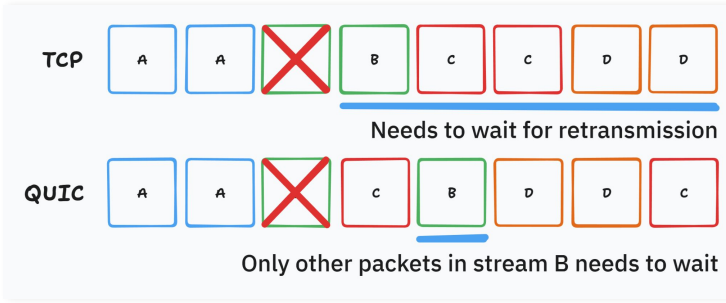
- Android platform implementation
- Testing and data collection

Literature Study

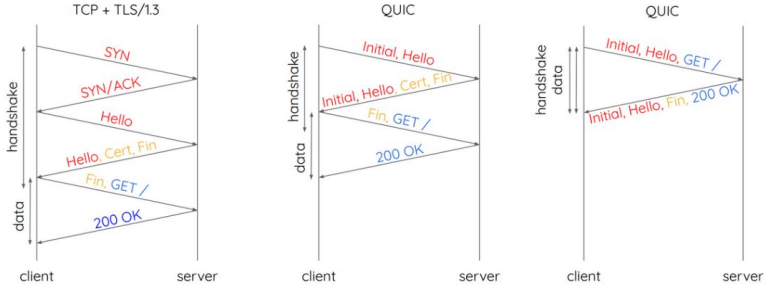


QUIC

- QUIC is a general purpose transport protocol built on top of UDP, made by Google.
- It is designed to address the limitations of TCP, particularly RTT delay and Head of Line (HoL) blocking.



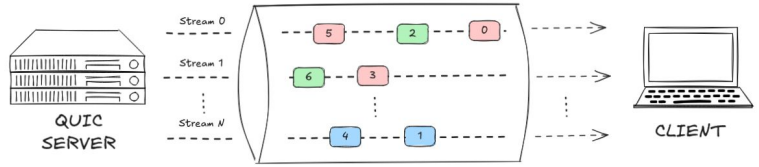
Stream Delivery in TCP vs QUIC



Connection Handshake in TCP + TLS vs QUIC

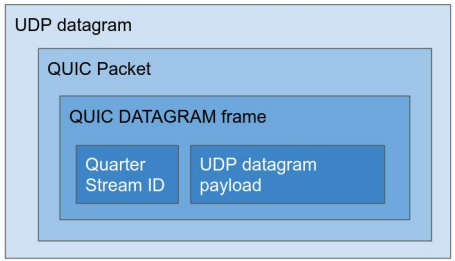
QUIC (cont'd)

Streams



- Also known as reliable QUIC transport
- It is similar to TCP, retransmissing any packet drops to ensure client receives all data
- Can be multiplexed through multiple streams in a single QUIC connection

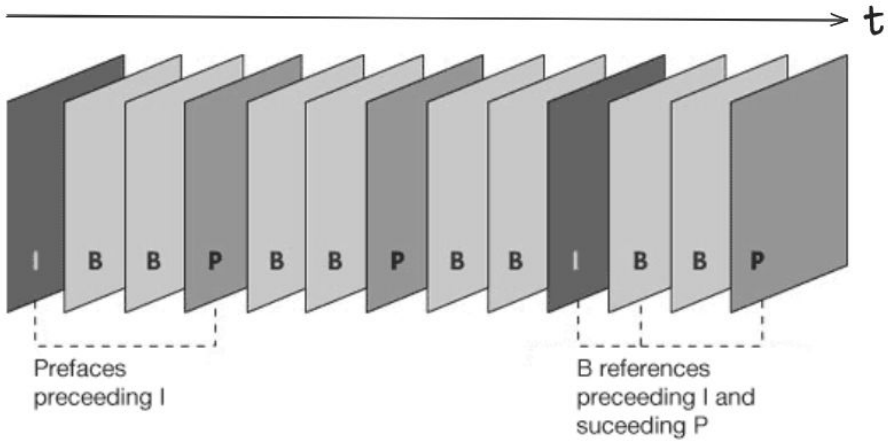
Datagram



- Also known as unreliable QUIC transport
- Does not retransmit any packet drops
- Data will be unordered, client needs to handle it themselves

H264 Video Frames

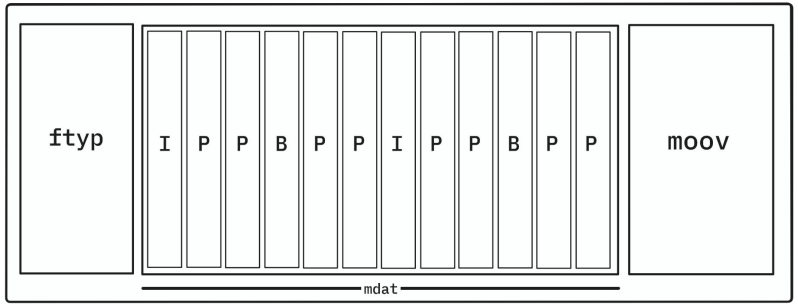
- H.264 is a video codec which uses lossy compression
- It categorizes video frame onto 3 type, I-frame, P-frame, and B-frame
- It leverages pixel-wise difference on encode and decode process to reduce file size



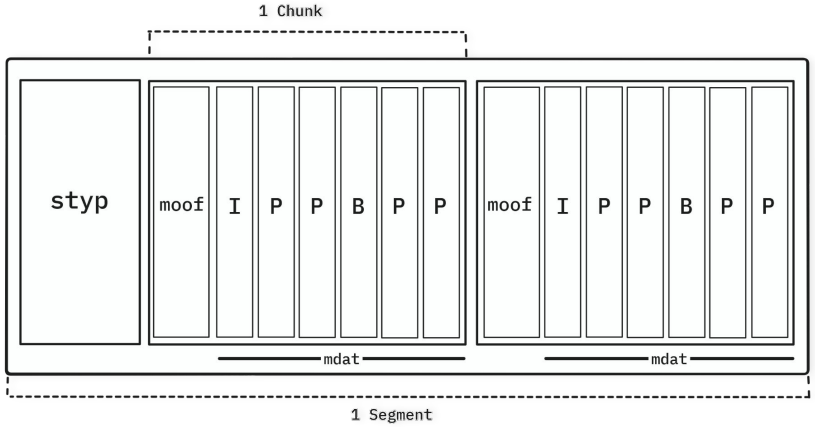
H.264 Video Frames

MP4

- MP4 is a widely used multimedia container with an object-oriented structure
- It is made up of "boxes" or "atoms"
- The structure of mp4 is usually: ftyp-mdat-moov

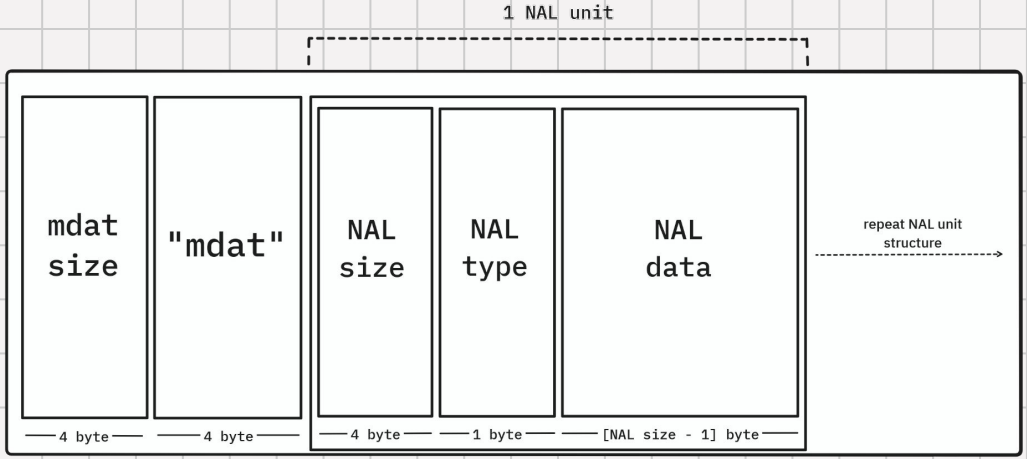


- Fragmented MP4 splits these big mdat chunks into smaller ones
- Introduces a new atom: moof
- Usually done to effectively stream large videos



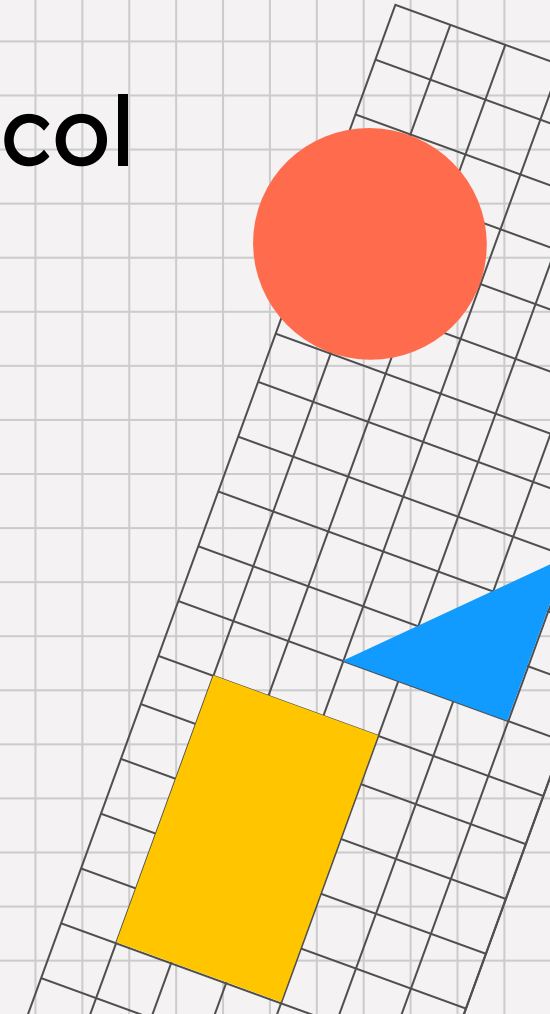
NAL Units

- Inside of MP4's mdat, there are small units of frames called **NAL Units**.
- This information is decoded by the decoder in-order.



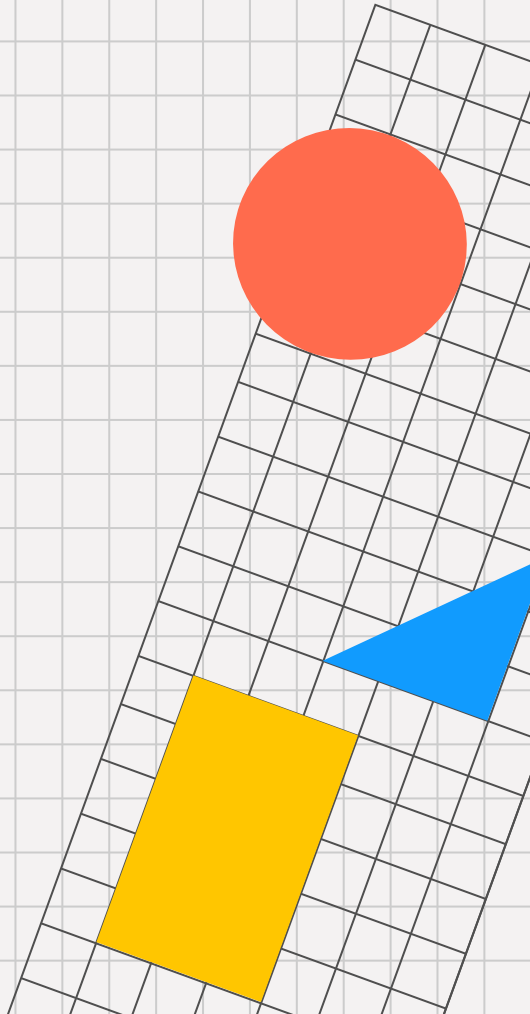
Real-Time Messaging Protocol (RTMP)

- RTMP is an application-level protocol designed for multiplexing and packetizing multimedia transport streams
- Uses reliable transport such as TCP and support bidirectional communication with guaranteed ordering
- It implements media chunking to offer low-latency performance and scalability purposes

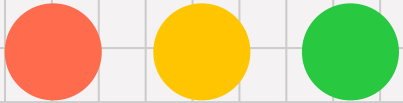


Webview

- WebView is a component that displays web pages within a native application
- It uses the system's default browser engine and functions similarly to a browser
- Easy to use with Jetpack Compose, Android's native UI framework



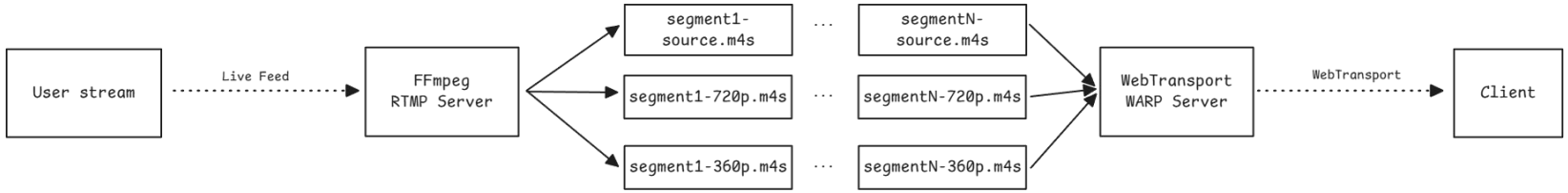
Research Methodology



Approach and Research Phases

Input	Phase	Methods	Output
• General problem • Prior research	Problem Identification	Literature study	• Research problems and objectives
• Prior research • Research problem and objectives	Literature Study	Review, compare, contrast, and synthesize	• Theoritical basis of system infrastructure design • Theoritical basis of experiment
• Existing system and infrastructure • Literature study theoritical basis	System Infrastucture Design and Development	Incremental development	• Android application • Live streaming feature implementation • Partial-Reliable transport mode enhancement
• Application and infrastructure prototype • Evaluation design	Evaluation And Analysis	Direct observation and comparison	• Experiment analysis
• Evaluation and analysis result	Research Conclusion	Analysis	• Conclusion • Research implication

Proposed Architecture



Live Streaming Architecture

- User will stream to FFmpeg's RTMP server
- FFmpeg will process the stream and split it into LL-DASH
- WARP server will process all the segments and sends it to client
  - **Initially, the server will return the last connection**
  - **After that, it will wait for new segments and publish it once detected**
- Client will then retrieve the segments and decode it in the browser

Proposed Architecture

Hybrid

- We will do preliminary experiment on the most effective number of fragments to be sent through datagram
- The number will then be used as threshold for when to send the data via datagram or stream

Android

- We will implement and test the performance through the system's WebView
- Improve the code's performance to be less resource-intensive so it could be run in mobile environment

Methods of Evaluation

Latency

- End-to-end latency
  Latency from streamer to viewer
- Chunk-to-chunk latency
  Latency to send one chunk from server to client

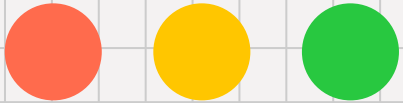
Jitter

Absolute difference between two consecutive chunk-to-chunk latency

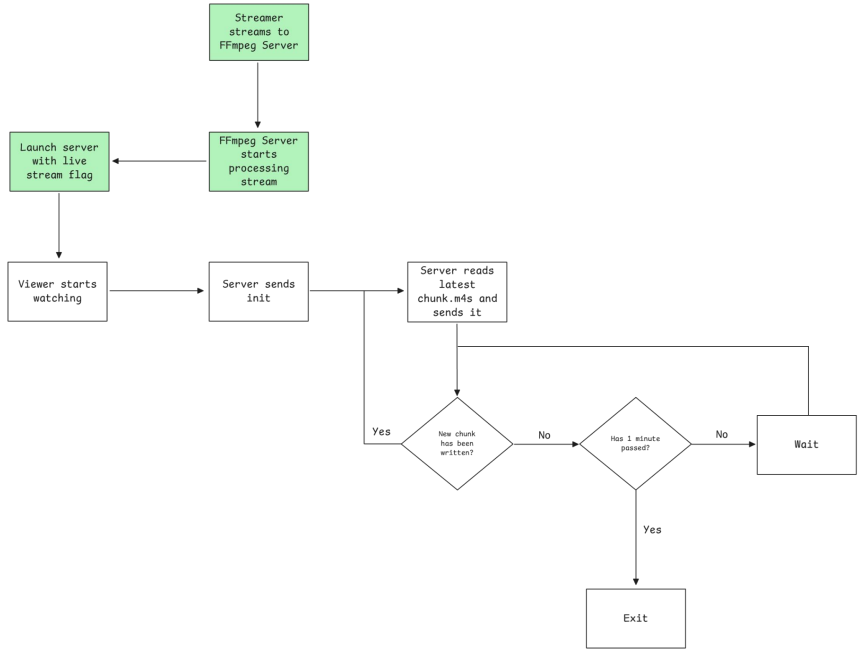
Download Speed

The rate at which the network can download over time

Implementation

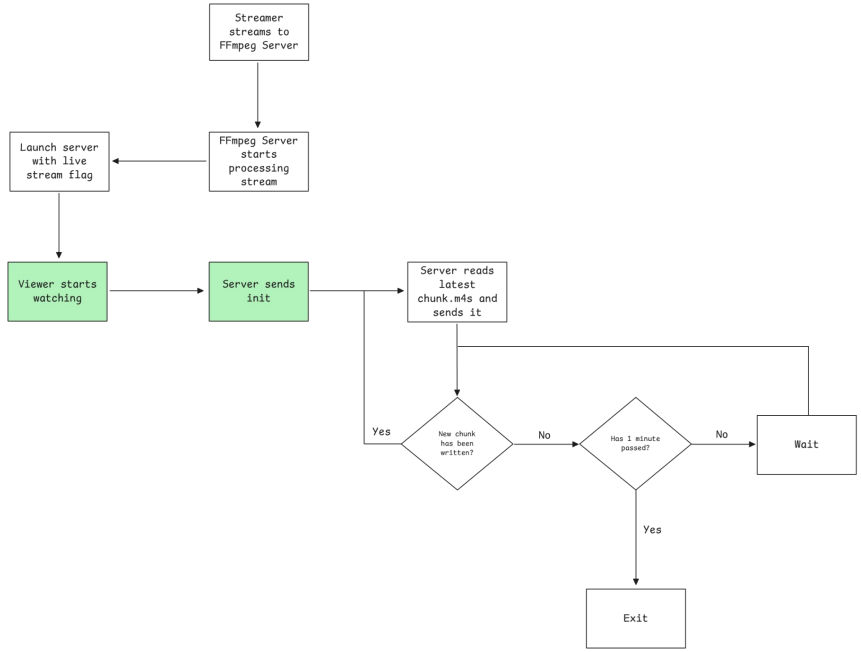


Live Stream



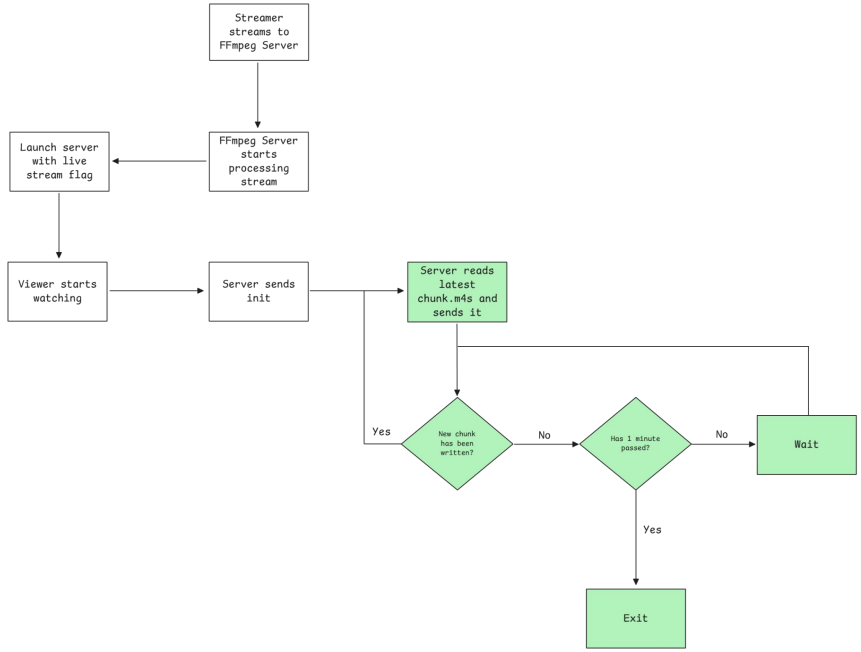
- First, the FFmpeg RTMP Server will start first and starts listening
- The streamer will connect to the RTMP server and starts streaming
- FFmpeg will process the stream and convert it into LL-DASH
- It will then start the WebTransport server with the live stream flag

Live Stream



- When a viewer starts watching the stream, the server will start sending the init files required for the player to play later chunks

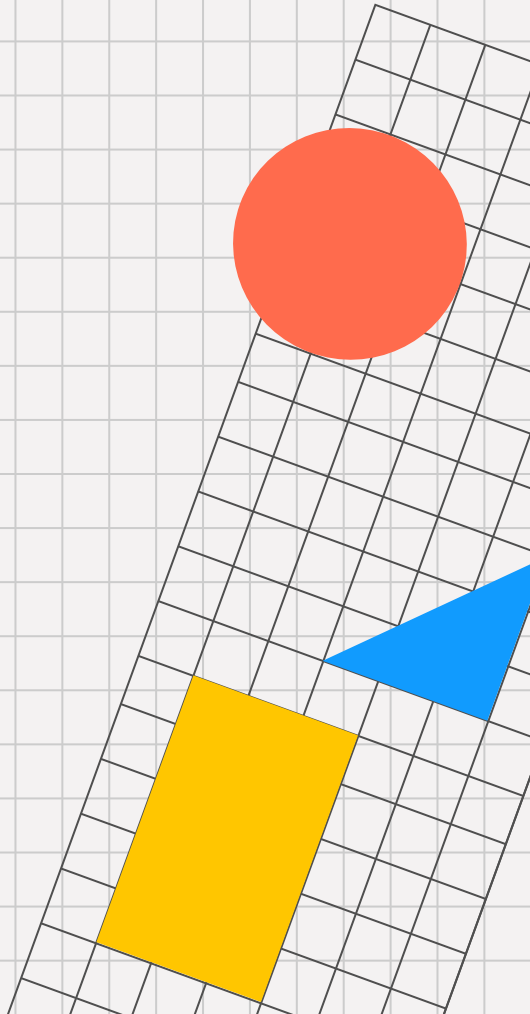
Live Stream



- The server will start sending the latest chunk to the client.
- Once a chunk has been fully sent, it will check for new chunks:
  - If there is, then loop back again to send it
  - If not, then check if it has been 1 minute:
    - If yes, assume the streamer has been disconnected, so we close the connection and exit
    - If not, wait until new chunk has arrived or 1 minute, whichever comes first

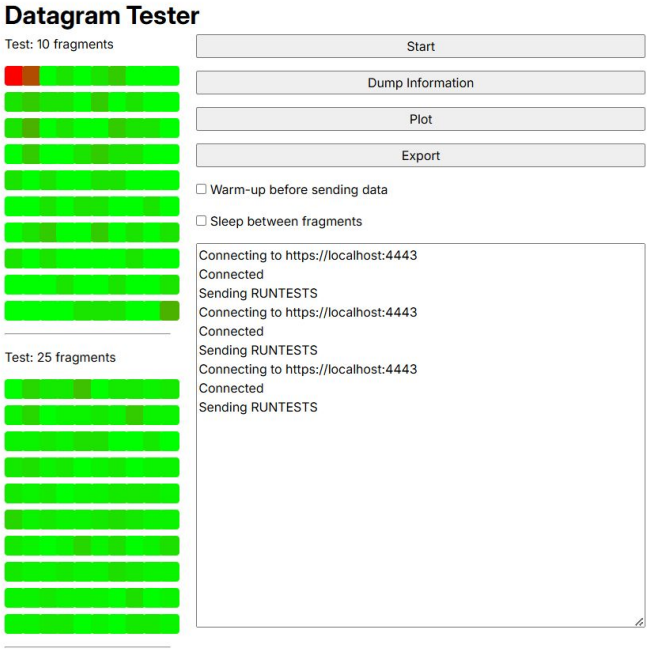
Hybrid: Overview

- Hybrid mode combines:
  - **Reliable stream transport (I-frames)**
  - **Unreliable datagram transport (P-frames, B-frames)**
- Challenge: QUIC Datagram Maximum Transmission Unit (MTU) restriction of **~1200 bytes**
  - **Requires manual fragmentation for chunks larger than MTU, losing a single fragment renders the whole chunk unplayable**
  - **Significant latency increase on chunks with large number of fragments**
- We propose a new fragmentation algorithm to better mitigate this challenge.

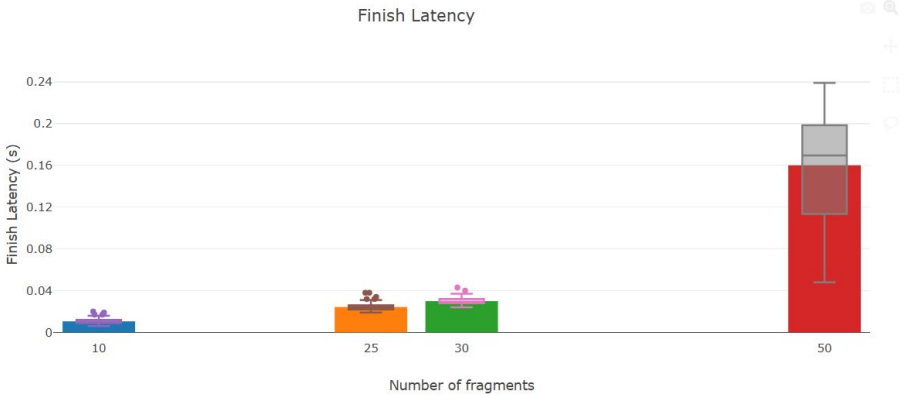


Hybrid: Preliminary Experiment

- Experiment is done using datagram testbed on local network with simulated **5%** packet drop rate, and through **100 test** iterations.
- The time taken to finish getting all the fragments **(Finish Latency)** will be measured across 4 different number of fragments: **10, 25, 30, 50**.
- The finish latency must not exceed **0.04 second** as it's the FFmpeg Server's chunk duration



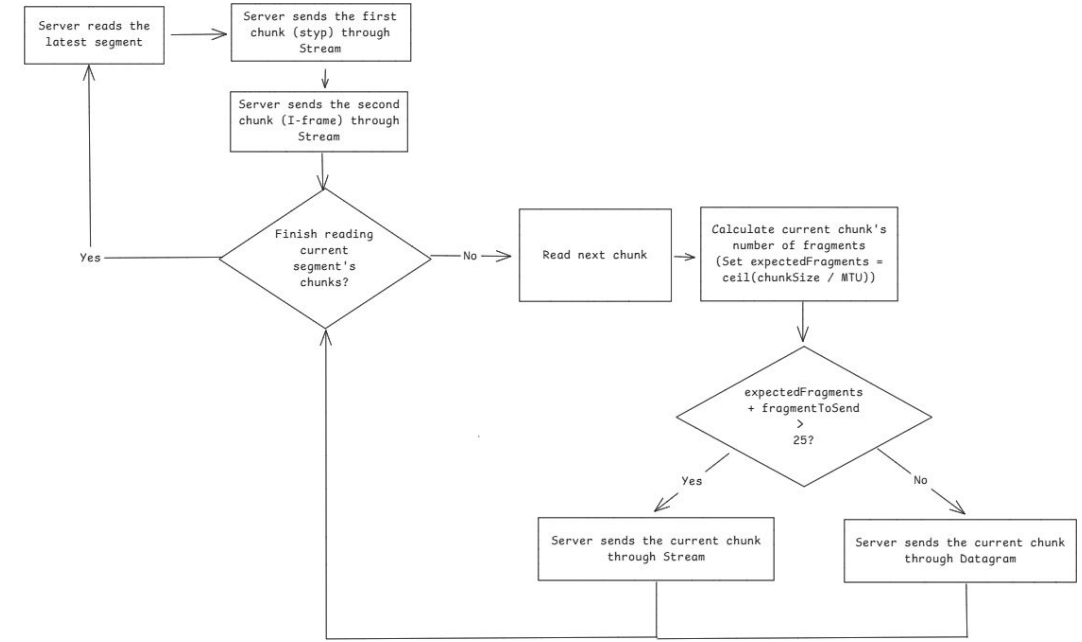
Hybrid: Experiment Result



Number of Fragment	Mean Finish Latency (s)	Max Finish Latency (s)
10	0,0106	0,02
25	0,0243	0,038
30	0,0299	0,043
50	0,1601	0,239

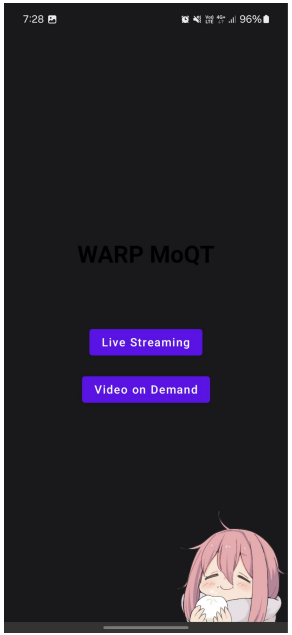
- 50 fragments: **Infeasible** because the mean exceeds 0.04 second threshold
- 30 fragments: **Feasible** but **some chunks** exceed the 0.04 second threshold
- 25 fragments: **Feasible** and **no chunks** exceed the 0.04 second threshold
- 10 fragments: **Feasible** but would underutilize datagram mode's capability
- **25 fragments** will be used as the threshold

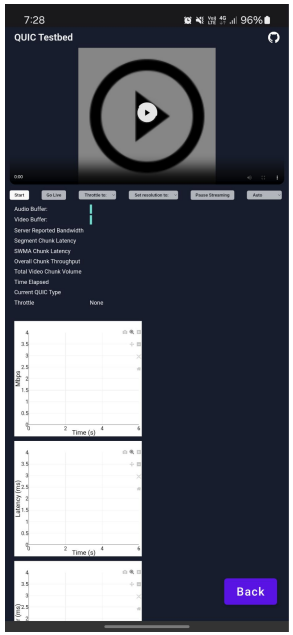
Hybrid: Improvement Implementation



- Following the steps from Live Stream, the latest segment will first be read.
- The first chunk (styp) and second chunk (I-frame) will be sent through Stream
- The third and the following chunks are then subject to the **25 fragments** threshold:
  - **If transmitting current chunk over datagram would exceed 25 fragments in the datagram queue, then transmit through stream**
  - **If not, then transmit through datagram**

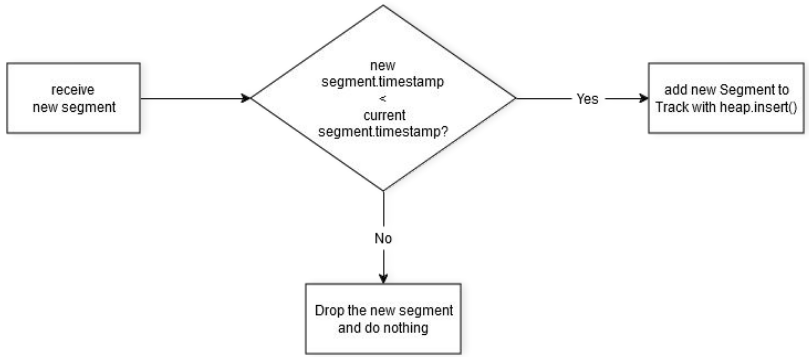
Android Implementation





- There are 2 modes available on the application's home page, Live Streaming and Video on Demand (VoD)
- Both of the button will redirect to the same webpage, only the video source url differs.

Player Improvement



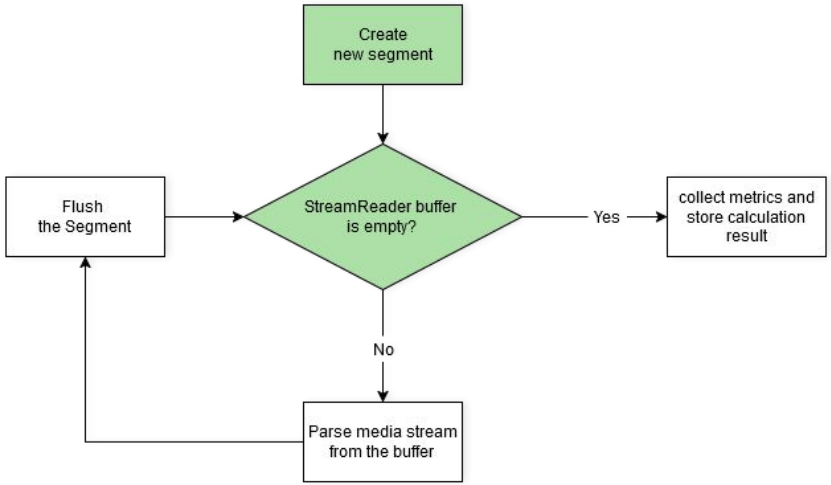
Previously, every single flush calls a sort to the segment array, now we make use of heap.

Inner workings:

- Upon receiving new segment, player will compare the new segment's timestamp with current play time.
    - **If the new segment is newer, add the new segment to Track.**
    - **If the new segment is older, drop the segment and do nothing**

Player Improvement (cont'd)

Flush New Segment Instantly



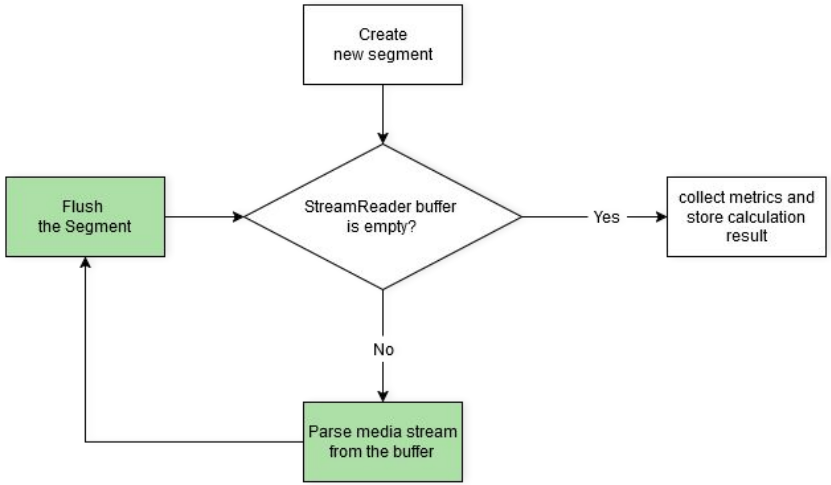
Previously, segment is flushed after the entire segment has been received. Now, it will flush every chunk received.

Inner workings:

- Upon receiving new media stream, player will create a new Segment.
- Player relies on the StreamReader to consume the media buffer needed.

Player Improvement (cont'd)

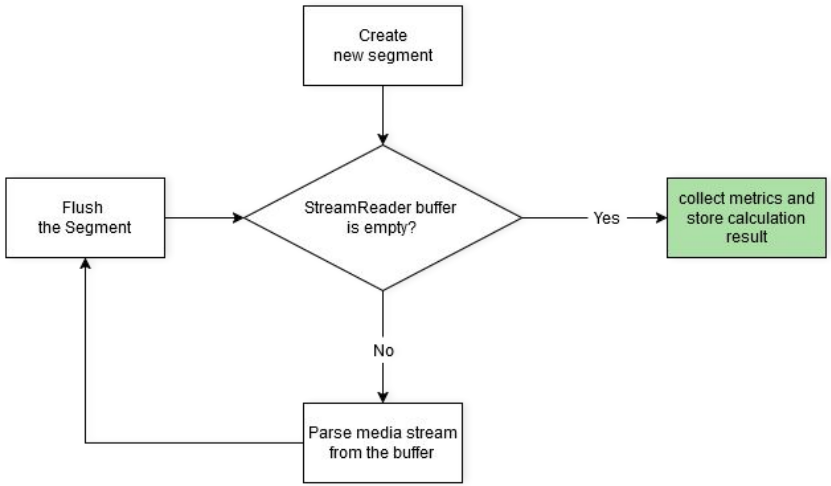
Flush New Segment Instantly



- **While StreamReader's buffer is not empty**, player will keep reading the buffer in fixed size.

1. Player will parse the media stream from the buffer.
2. The media stream is immediately flushed
3. Go back to step 1

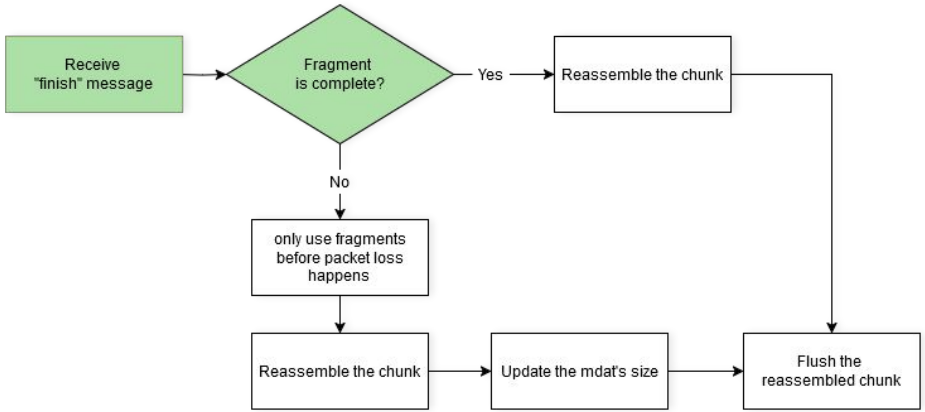
Player Improvement (cont'd)



- If there's no more buffer to consume, player will stop processing the segment
- Player will finally collect metric and store the calculation results used for the evaluation.

Player Improvement (cont'd)

Dynamic Adjustment on Fragment Reassembly During Packet Loss on Hybrid Mode



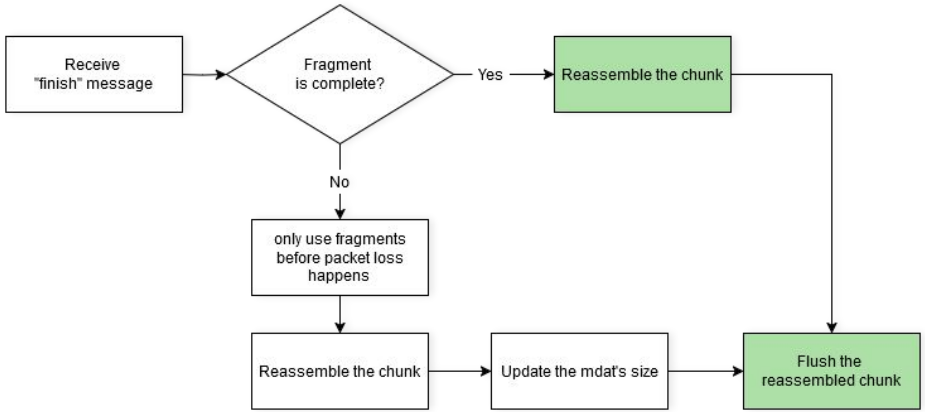
Previously, on packet loss, the entire chunk is discarded. Now, it will still process it partially.

Inner workings:

- The Server will send "finish" message for after sending all fragment in current chunk
- Upon receiving "finish" message, player will then determine the next step based on the fragments completeness

Player Improvement (cont'd)

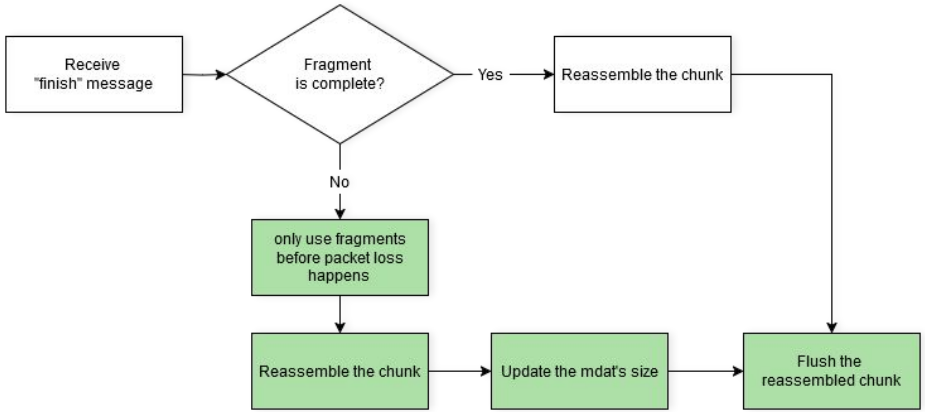
Dynamic Adjustment on Fragment Reassembly During Packet Loss on Hybrid Mode



- If the Fragment is complete (all fragments received)
  - Reassemble the fragments on to chunk
  - Immediately flush the chunk

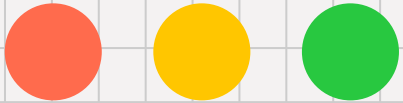
Player Improvement (cont'd)

Dynamic Adjustment on Fragment Reassembly During Packet Loss on Hybrid Mode



- If the Fragment is incomplete (packet loss happens)
  - Only take fragments before packet loss
  - Reassemble the fragments onto chunk
  - Update the original mdat's size to current size
  - Immediately flush the chunk

Evaluation



Evaluation Design

Live + Hybrid

- We will evaluate the design using local network in CSL Lab
- Two PC in CSL will serve as streamer and viewer, DGX server will be used as the live stream server.
- All devices' time is synchronized.
- Streamer will show a video with an overlay of the machine's current time

Example Video Stream

Evaluation Design

Android

- We will evaluate the design using home Wi-Fi and mobile data network
- A high-end consumer grade PC is used to stream the video
- An Android phone with Android version 14 and chrome version 131 is used to watch the live stream video.
- Testing will be done using Wi-Fi 5Ghz.

Example Video Stream

Evaluation Design

Live Stream

- Take screenshots every 10 seconds
- Calculate the difference between the machine's time and current live stream timestamp.

Hybrid

- There are 5 packet drop scenarios that will be used: 0.25%, 0.5%, 1%, 2%, and 5%
- For each scenario, we take QoS metrics for both Stream and Hybrid:
  - **Chunk Latency**
  - **Jitter**
  - **Download Speed**

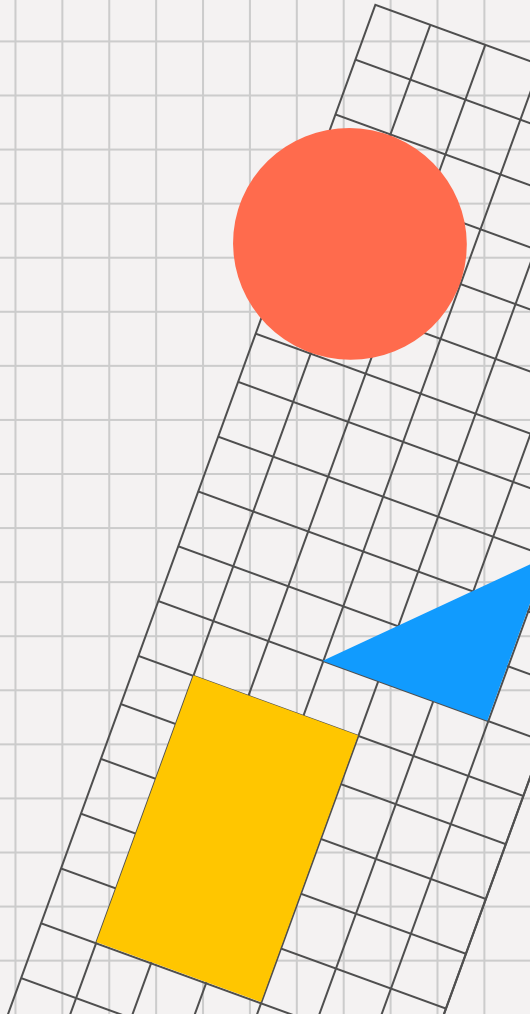
Android

- Compare QoS performance between Desktop and Android, which will be tested in:
  - **Google Chrome**
  - **WebView**
- Compare QoS between various network conditions:
  - **Mobile Data 4G**
  - **Mobile Data 5G**
  - **Wi-Fi 2.4Ghz**
  - **Wi-Fi 5Ghz**

Evaluation Design

Preliminary downlink test for Android testing

Type	Downlink Rate (Mbps)
Wifi 6 with 2.4 GHz Band	59,5
Wifi 6 with 5 GHz Band	63,3
Mobile Data 4G	33,3
Mobile Data 5G	44

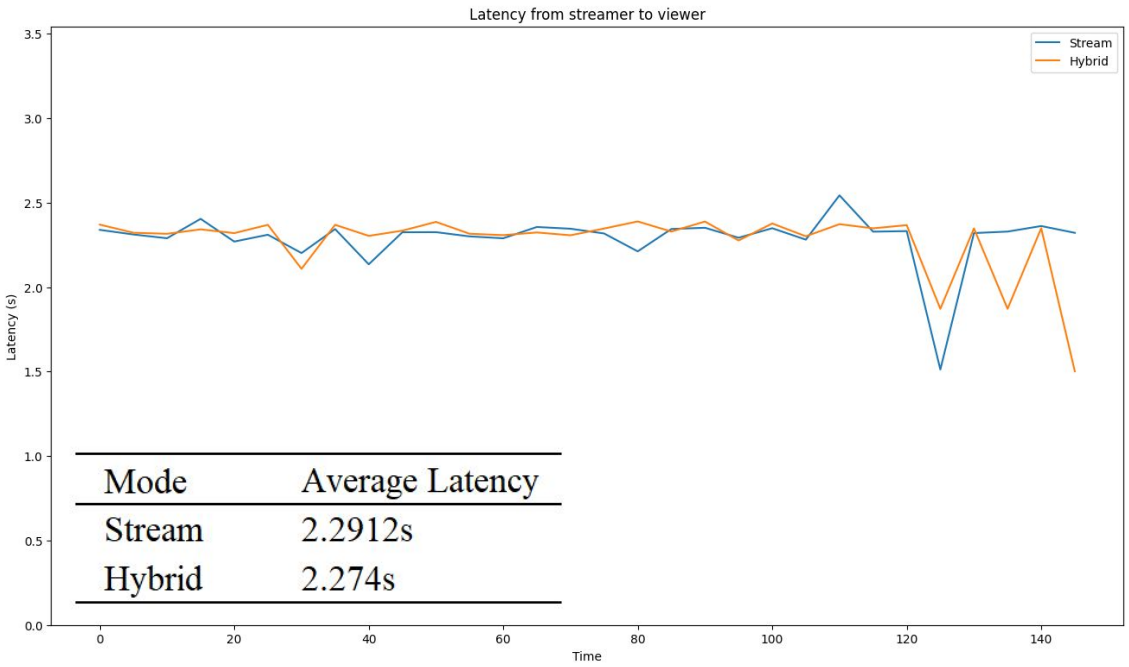


Evaluation Result

Live Stream

Based on testing, we found the latency for end-to-end (from streamer to viewer) sits **around 2.3s**, with the maximum at 2.5s

This is competitive compared to other live streaming services. For example, YouTube with ultra low latency setting claims that latency is "below 5s".

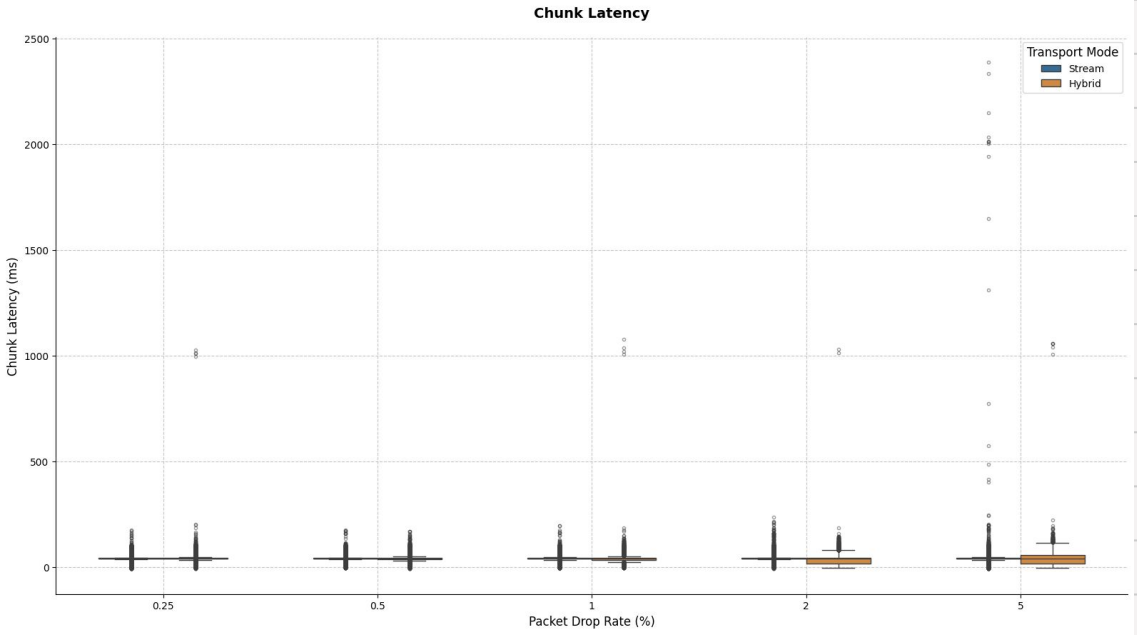


Mode	Average Latency
Stream	2.2912s
Hybrid	2.274s

Evaluation Result

Hybrid (Latency)

- Stream mode have large latency spike (up to **2500ms**) at 5% packet drop rate
- Because of stream mode's retransmission, some chunks are sent extremely late, causing bandwidth inefficiency



Evaluation Result

Hybrid (Latency)

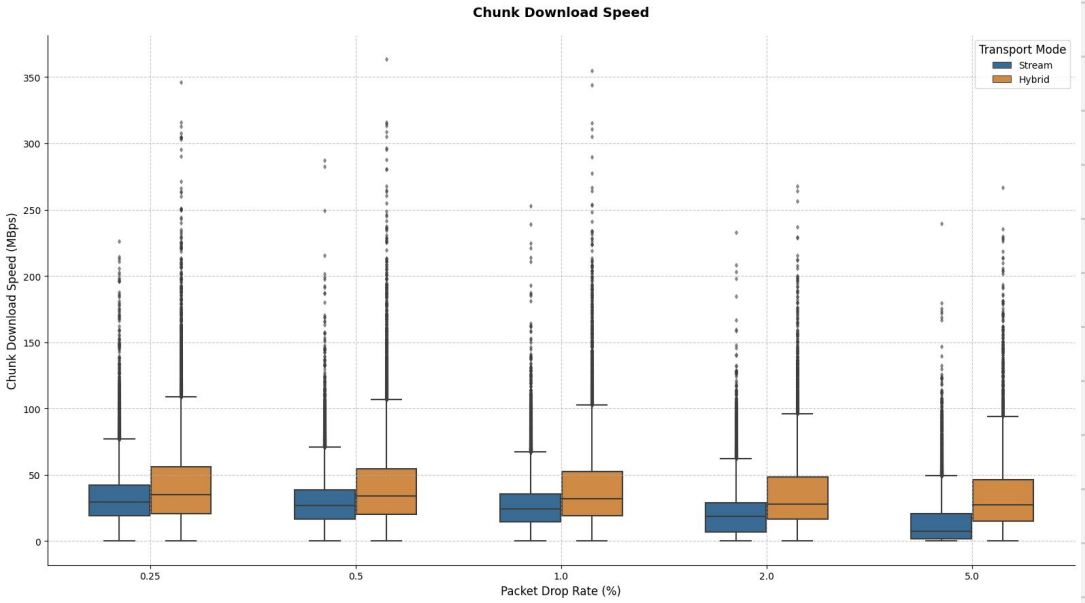
- Both transport modes show similar mean latencies between **35-41** ms across most packet drop scenarios.
- Hybrid mode achieves lower median latencies of **39ms** vs stream's **40ms** at higher packet drop rates (2% and 5%).

Drop rate	Mean		Median		Standard Deviation	
	Stream	Hybrid	Stream	Hybrid	Stream	Hybrid
0,25%	37.02	37.26	40.00	40.00	14.34	21.05
0,50%	38.32	35.06	40.00	40.00	13.01	19.20
1%	36.45	36.89	40.00	40.00	15.33	25.72
2%	37.96	35.96	40.00	39.00	14.69	28.86
5%	37.75	41.38	40.00	39.00	41.42	43.54

Evaluation Result

Hybrid (Download Speed)

- Hybrid mode maintains **consistent and higher** mean and median download speeds across all packet drop rates.
- Stream mode shows significant download speed **degradation** as packet drop rates increase.
- Hybrid mode demonstrates **better bandwidth utilization**, especially in poor network conditions.

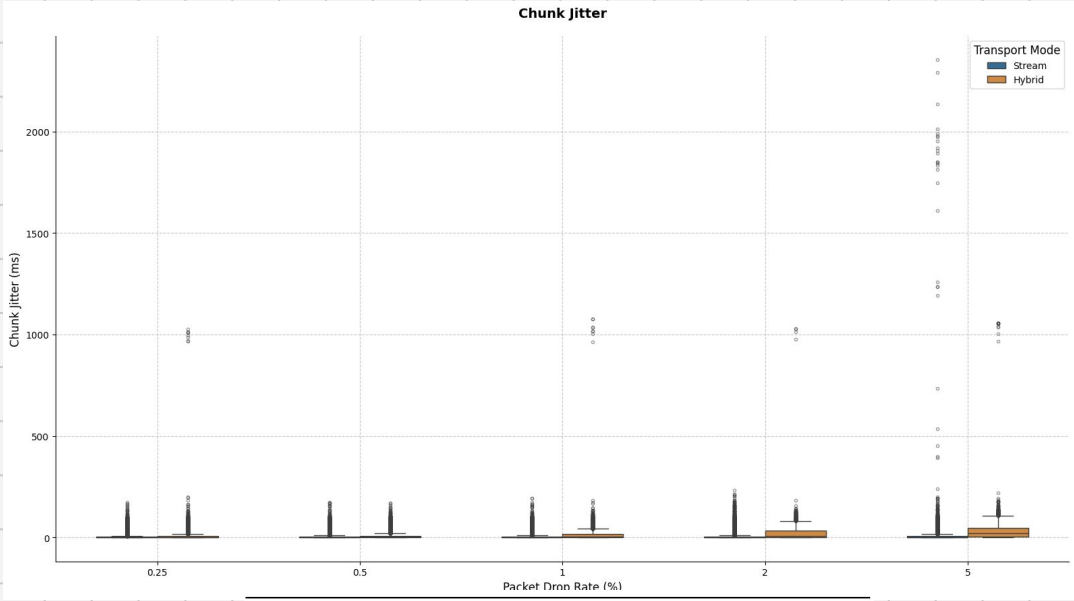


Drop rate	Mean		Median		Standard Deviation	
	Stream	Hybrid	Stream	Hybrid	Stream	Hybrid
0.25%	32.36	42.34	29.29	34.74	19.89	31.18
0.50%	29.26	41.46	26.58	33.82	18.38	31.54
1%	26.86	40.07	24.36	32	18.65	31.18
2%	20.51	36.23	18.59	28.01	16.86	29.24
5%	13.17	35.61	7.33	27.06	15.47	31.38

Evaluation Result

Hybrid (Jitter)

- Hybrid mode constantly shows higher jitter across all packet drop rates
- Trade-off between **better bandwidth utilization** vs **higher jitter**

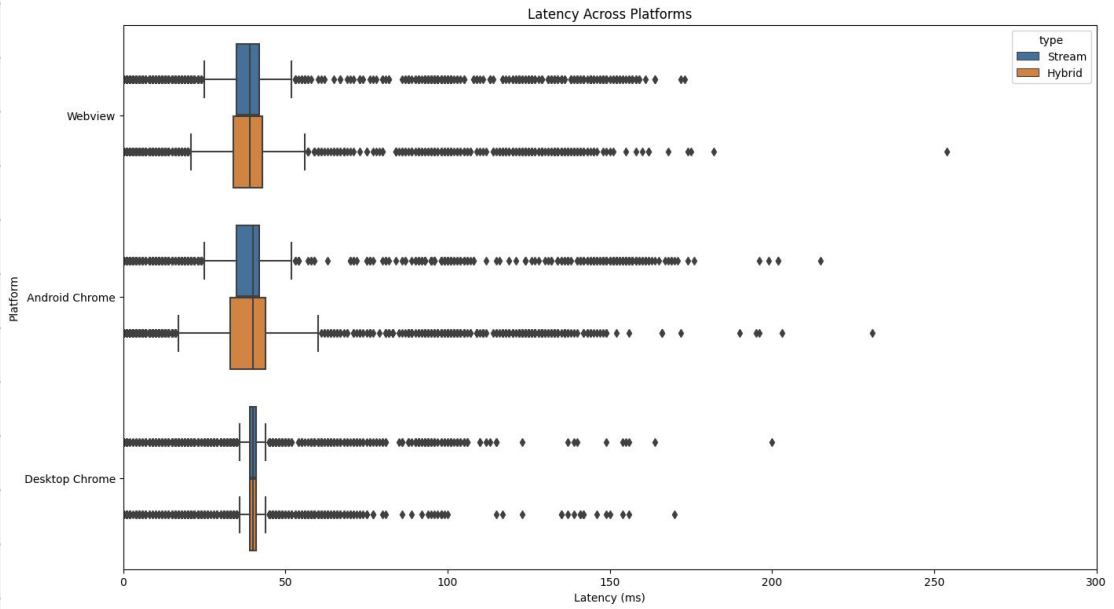


Drop rate	Mean		Median		Standard Deviation	
	Stream	Hybrid	Stream	Hybrid	Stream	Hybrid
0.25%	7.42	10.44	2	2	15.83	26.1
0.50%	7.8	11.12	2	3	16.29	19.12
1%	8.1	14.67	2	4	16.77	31.2
2%	8.43	20.5	2	8	18.08	32.31
5%	10.95	32.29	2	22	53.26	53.23

Evaluation Result

Android (Latency Across Platform)

- Based on the testing, we can see every platform has identical average latency, around **40 ms**.
- Desktop Chrome has the narrowest inliers and whiskers range for both mode.

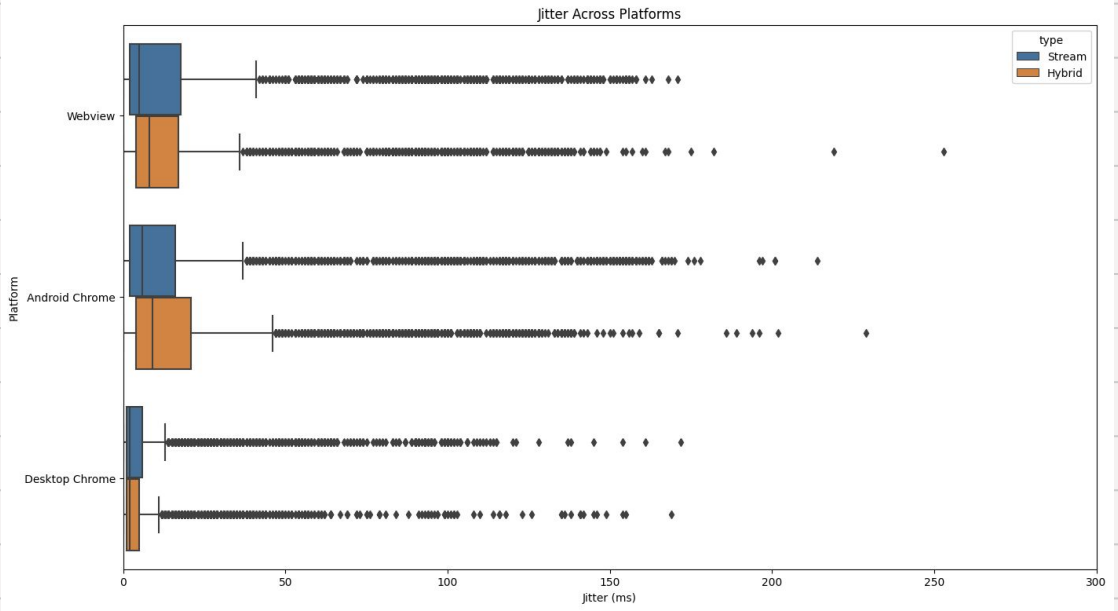


Platform	Average Latency (ms)	
	Stream	Hybrid
Webview	39.93	40
Android Chrome	39.96	39.95
Desktop Chrome	39.99	39.98

Evaluation Result

Android (Jitter Across Platform)

- Based on the testing, we can see that Desktop Chrome has the narrowest inliers range and lowest average jitter, **approximately 3 times lower** compared to other platfrom
- This indicates Desktop Chrome has the best jitter consistency and latency stability
- Both Webview and Chrome Android have roughly the same consistency and stability level.

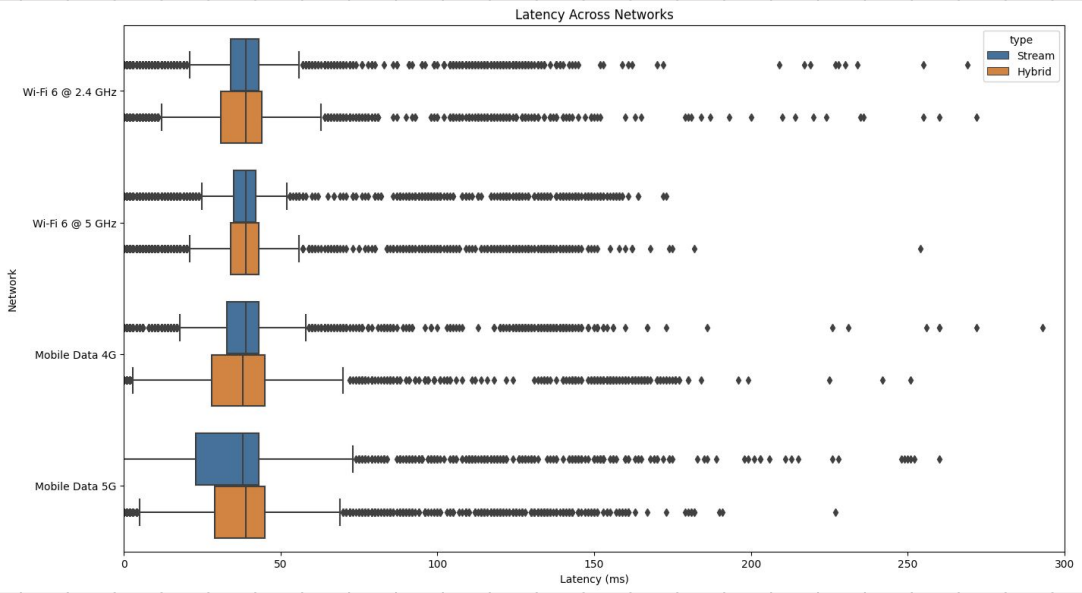


Platform	Average Jitter (ms)	
	Stream	Hybrid
Webview	20.39	21.1
Android Chrome	21.65	21.54
Desktop Chrome	9.47	7.69

Evaluation Result

Android (Latency Across Network)

- Based on the testing, we can see every network has identical average latency, around **40 ms**.
- Wi-Fi 5 GHz has the narrowest inliers and whiskers range, followed by Wifi 2.4 GHz, Mobile Data 4G and 5G

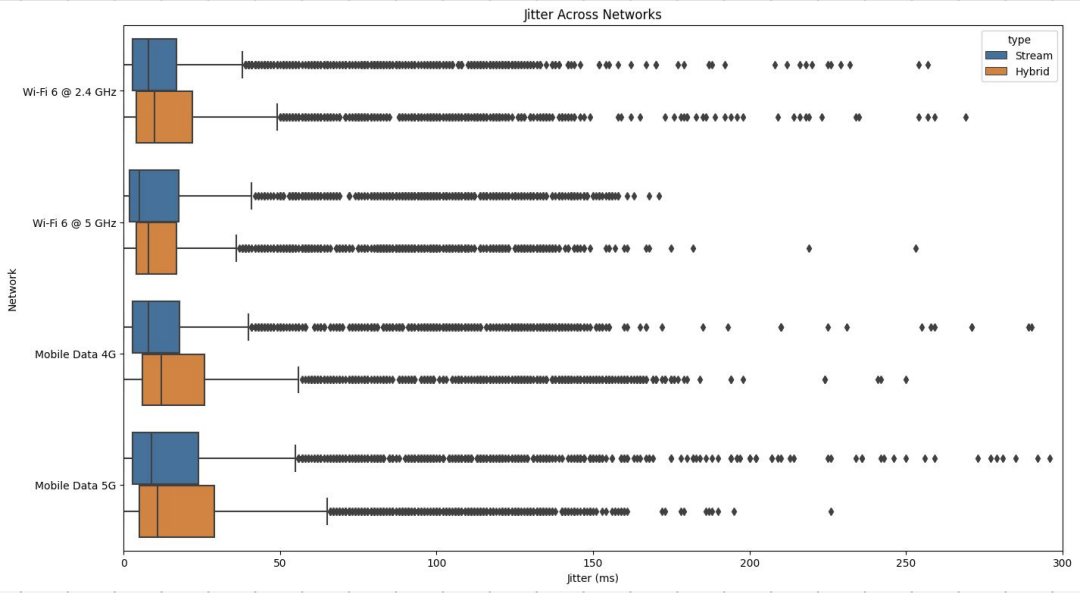


Network Type	Average Latency (ms)	
	Stream	Hybrid
Wifi 2.4 GHz	39.95	39.95
Wifi 5 GHz	39.93	40
Mobile Data 4G	39.98	40.11
Mobile Data 5G	39.98	40.36

Evaluation Result

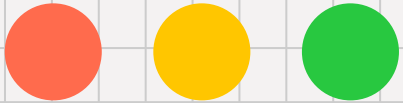
Android (Jitter Across Network)

- Based on the testing, we can see both Wi-Fi network has smaller jitter compared to both mobile data network.
- This indicates Home Wi-fi network offers better latency and jitter stability
- Both Mobile Data 4G and 5G have relatively higher jitter, meaning unstable



Network Type	Average Jitter (ms)	
	Stream	Hybrid
Wifi 2.4 GHz	19.55	22.56
Wifi 5 GHz	20.39	21.1
Mobile Data 4G	21.06	26.69
Mobile Data 5G	33.1	25.94

Conclusion



Conclusion

Conclusion

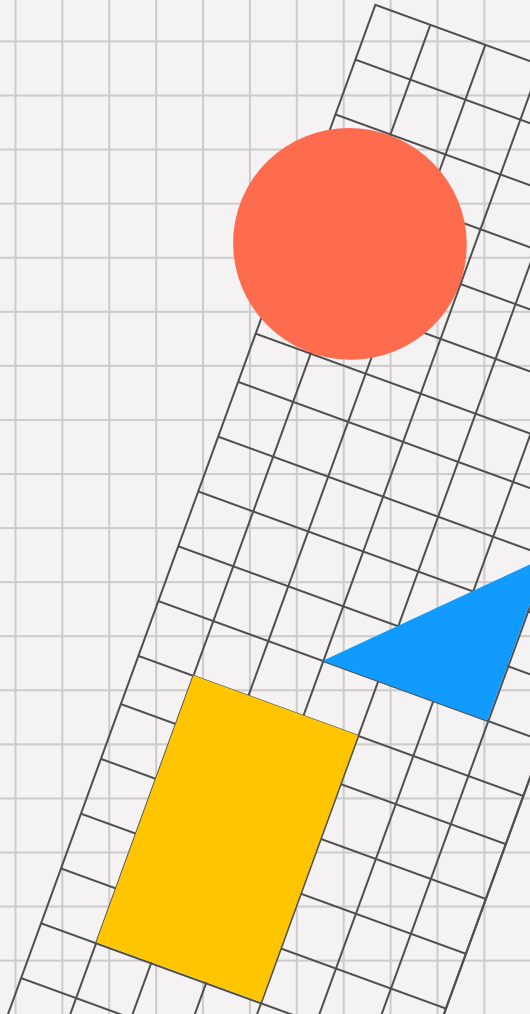
- We have successfully implemented live stream feature with good results, around 2.3s end-to-end latency
- New implementation of hybrid mode mitigates large latency spike and faster download speed compared to stream. However has higher jitter.
- Android has worse jitter compared to Desktop, but is acceptable.
- Wi-Fi shows to be more stable compared to mobile, but difference is not too big.

Limitations

- Mentioned in quic-go's documentation, it is not suitable for heavy throughput of datagrams due to weaker algorithm compared to stream
- Current implementation of fragmentation is too naive, where it simply splits the raw bytes, potentially splitting the NAL units
- Current FFmpeg script writes segments every 2 seconds, therefore it can't do real-time transport

Suggestions

- Implement fragmentation in single-file instead of multiple chunks of fragmented mp4
- Alternatively, video delivery can be done by sending the raw H264 stream directly, instead of the mp4 container. Much like WebRTC



Thank You

Terima Kasih

감사합니다

ありがとうございます

谢谢

شكراً

ขอบคุณ

धन्यवाद

□□□□□□□□□□