Designing a Live Video Streaming System Like ESPN

By- Avash Mitra

Functional requirements

- There should be a maximum delay of 1 minute between the live event and the stream
- Our system should scale to a lot of users. (Heterogenous delivery)
- Our system should convert videos to different resolutions and codecs.
- Our system should be fault-tolerant.

Video transformation and ingestion

We are streaming the entire event live, so we cannot wait for the whole video to end to start converting it to different formats and codecs. To do that we can use a protocol called RTMP.

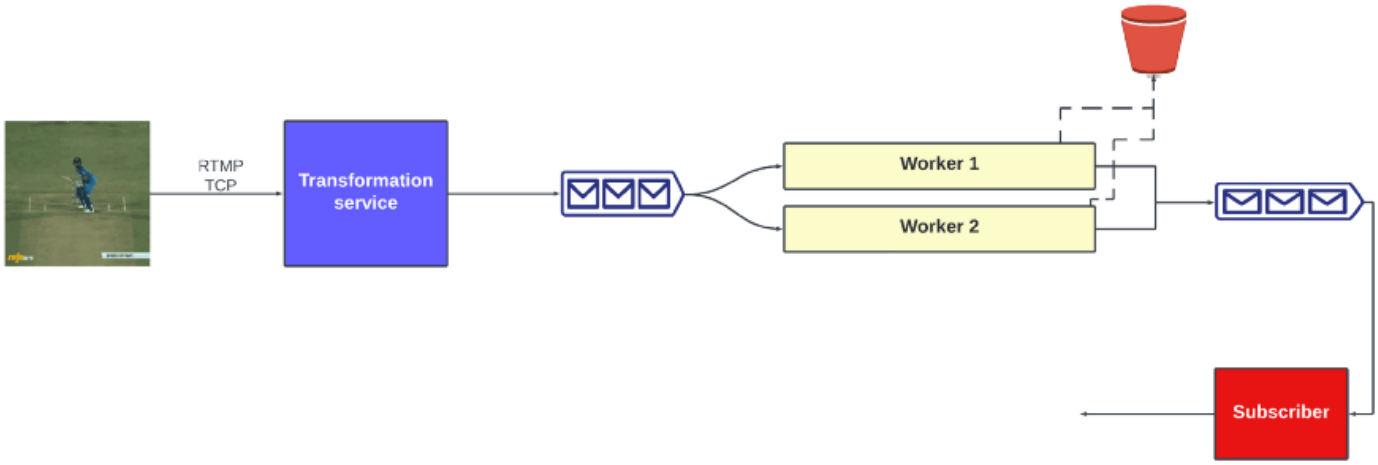
- RTMP stands for Real-Time Messaging Protocol.
- We are using RTMP because it is written over TCP so there is no data loss and is more reliable. We need TCP because we don't want to lose data packets during transportation. If we use something like UDP, it will be faster but the loss of data at every stage will lead to a poor-quality video.

Components required

- Transformation service RTMP gives us a video stream. Transformation service converts this video stream to different codecs and resolutions. It also has a job scheduler that takes the raw video stream as input and converts it into all resolutions and codecs. Several worker nodes carry out these tasks. When there is a new raw video stream transformation service pushes it to a message queue. Worker nodes subscribe to this message queue. They accept the input and once the video is converted, these nodes then push it to another message queue.
- Database We don't want to lose video data in case there is a disaster (We want fault tolerance). So we will use a database to store raw video data.

- Distributed File Service After worker nodes process the video, the result should also be stored in a file service for fault tolerance.
- Message queue

Architecture diagram



Transferring videos to end-users

We will use a Content Delivery Network to transfer videos to end-users. CDNs or Edge servers are geographically closer to the end-user.

Since our users will be connecting over HTTP so we can use protocols like HLS (HTTP Live Streaming) (for iPhone) or DASH (Dynamic Adaptive Streaming over HTTP) (for other operating systems).

But why should we use HLS/DASH?

HLS/DASH has an Adaptive Bit Rate. It means we will be adapting our bit rate depending on factors like

- Quality of video that can be served
- Network speed
- Resolutions and formats are supported by the client's device.

Since it is a streaming protocol it can do this judiciously with good usage of bandwidth and in real-time.

Note: HLS/DASH are lower quality connections compared to RTMP so we are making a tradeoff by choosing real-time over quality

To send videos to the CDN we can host servers around the world. We send the processed videos to the servers and then the servers send them to the CDN.

We will use RTMP to send videos to the servers.

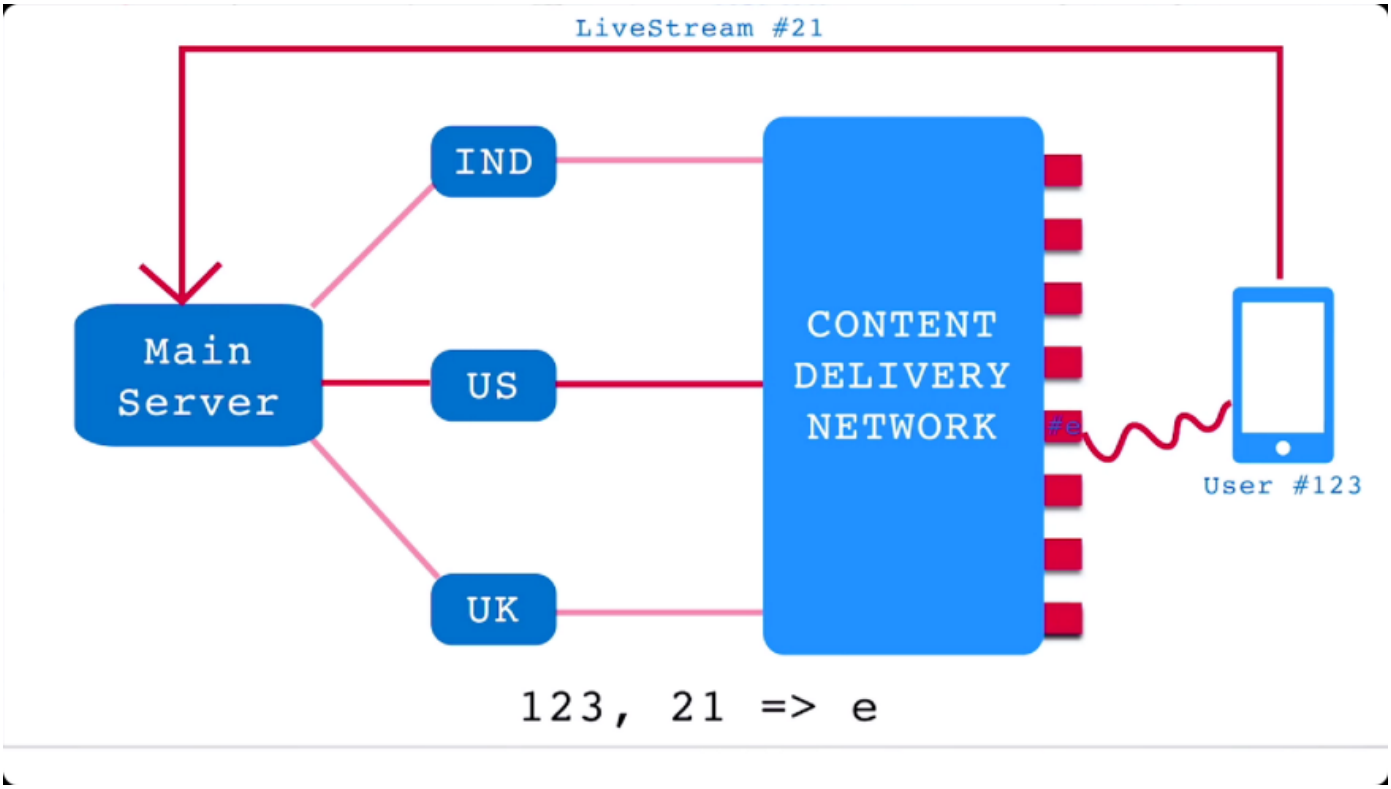
But, why are we not sending the processed video directly to the CDN?

The speed of propagation across content delivery networks may not be very high. So if we send the processed video to a CDN and they have servers in different locations the SLA guarantees might not be matching the Live streaming guarantee.

Whenever a client requests a video, one of the servers directs the client to an endpoint on the CDN. A client can use this endpoint to pull the video.

Caching

We will let the CDN take care of caching the videos. Although you can cache hot video segments in memory. However, to reduce the load on the server we can cache the mapping of Client <-> End-Point. Instead of begin rerouted every time, we get the end-point directly.



Fault Tolerance

To make our system more fault-tolerant we can use a load balancer. If one of the servers goes offline we can redirect the requests to other servers.

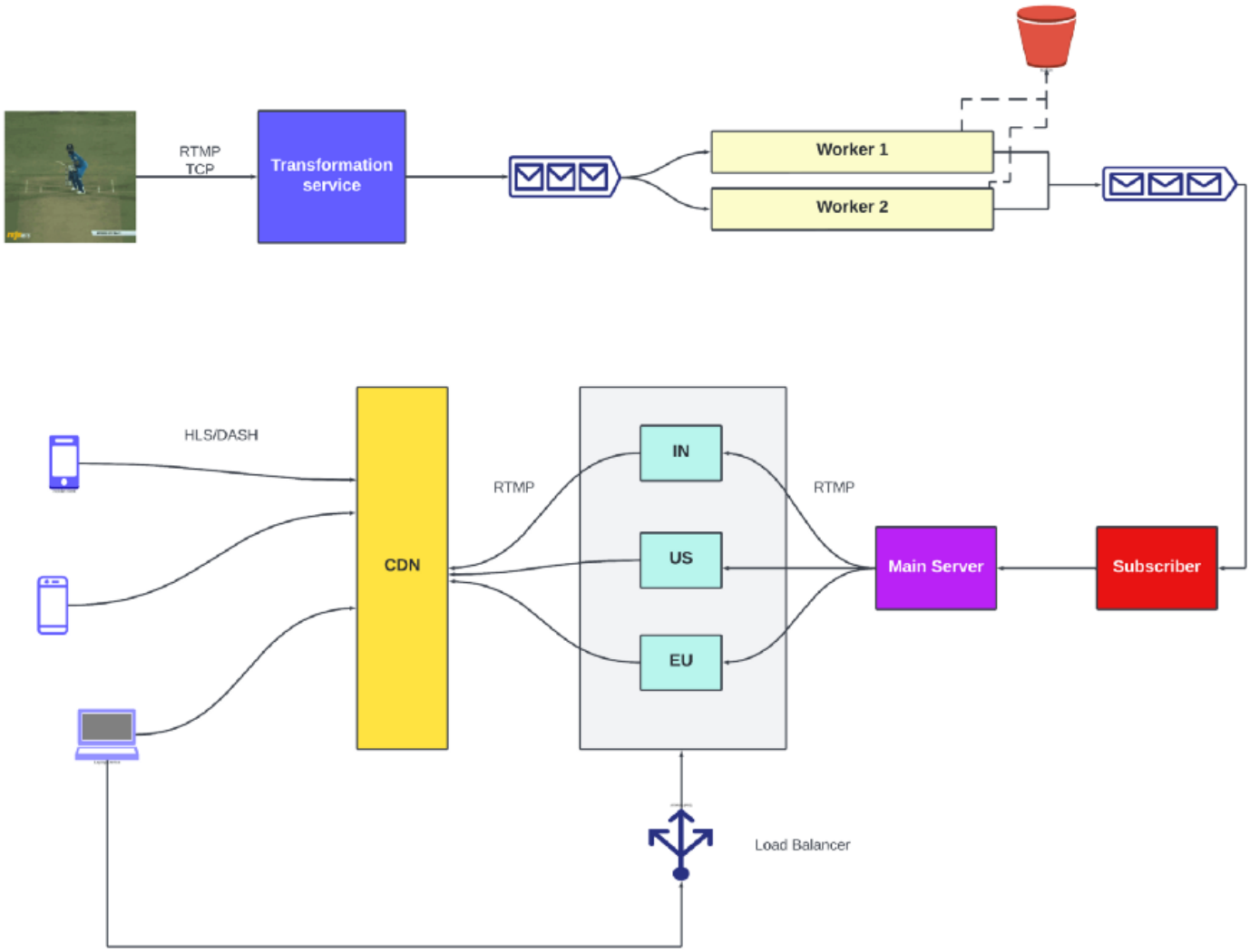
Trade-offs

- Using Web-RTC v/s using HLS/DASH for transferring videos HLS/DASH are HTTP based and work via TCP. They maintain orderly transfer. On the other hand, WebRTC is peer-to-peer-based and works via UDP. It might also send unordered chunks of data. Since we want to maintain the quality of video we will be using HLS/DASH.

Components Required

- Content Delivery Network
- Servers located in different locations

Architecture Diagram



Capacity Estimation

How many videos will need to be processed per live stream?

Assumptions

- 8k footage is being captured in the event.
- Resolutions we want to server: 1080p, 720p, 480p and 360p.
- Number of codecs: 4
- Duration of a cricket match: 10 hours
- Size of footage: 10GB

Size of 720p footage: 10/2 = 5GB Size of 480p footage: 10/4 = 2.5GB Size of 360p footage: 10/8 = 1.25GB

Total storage required for all resolutions: 18.75GB

Total storage for all resolutions and codecs: 18.75 * 4 = 75GB

How much data will be transferred to CDN in a single live stream?

Assumptions

- Number of users: 10,000,000
- Percentage of users having HD resolution: 50%
- Percentage of live stream users watch: 50%

Size of footage in standard resolution: 10/4 = 2.5GB

Therefore, Total Data Transfer = SUM(size_of_video_type * number_of_users_for_type) * average_watch_percentage

= (10 GB _ 50/100 _ 10^6 + 2.5 GB _ 50/100 _ 10^6) _ 50/100 = 3.125 _ 1000000GB = 3.125 PB

How much time would it take to send data from the event to a user's device?

Assumptions

- Amount of data consumed by user in a second = 10GB/10 hour = **300 kB/sec**
- Time is taken to transfer 300kB of data to the nearest CDN = **1sec**
- Time is taken to transfer 300kB of data to the user from CDN = **150ms**

Total travel time = 1 + 0.15 + 0.15 = **1.3sec**

Processing time assuming ffmpeg running at 2x of video speed = 1 / 2 = **0.5s**

Total latency = 1.3 + 0.5 =**1.8s.**