**Zoom , whatsup , fb video calling system design**

**Requirement : how**  to build video conferencing system or video chatting application. Something very similar to zoom, Webex, whatsup or fb video call, google hangsout.

We will also look how to enhance this platform to build a live streaming system like want to live stream of a match. How do u do that to million of users.

**Let look at some functional and non functional requirements**

**FR**

A picture containing text, whiteboard

Description automatically generated

**So the**  very first FR this platform should support **one to one calling** feature wherein once person is making a video call to another person. The next it should support **group calling also.**  May be 5 or 10 people come together they should be able to see each other and able to **.**or an audio screen. Allow user to **record video call.**  Screen recording support in zoom, Webex but not in whatsup.

A whiteboard with writing on it

Description automatically generated with low confidence

The platform should be super fast. High availability

**Let look how can we design these system**

**Let**  try to understand some protocol on which internet is based

**Let look how does a client application talk to a server**

**TCP :**

Connection happen b/w client and server and packet send in order

UDP :

Packet may not send in order and loss of packet also can occur but it is fast.

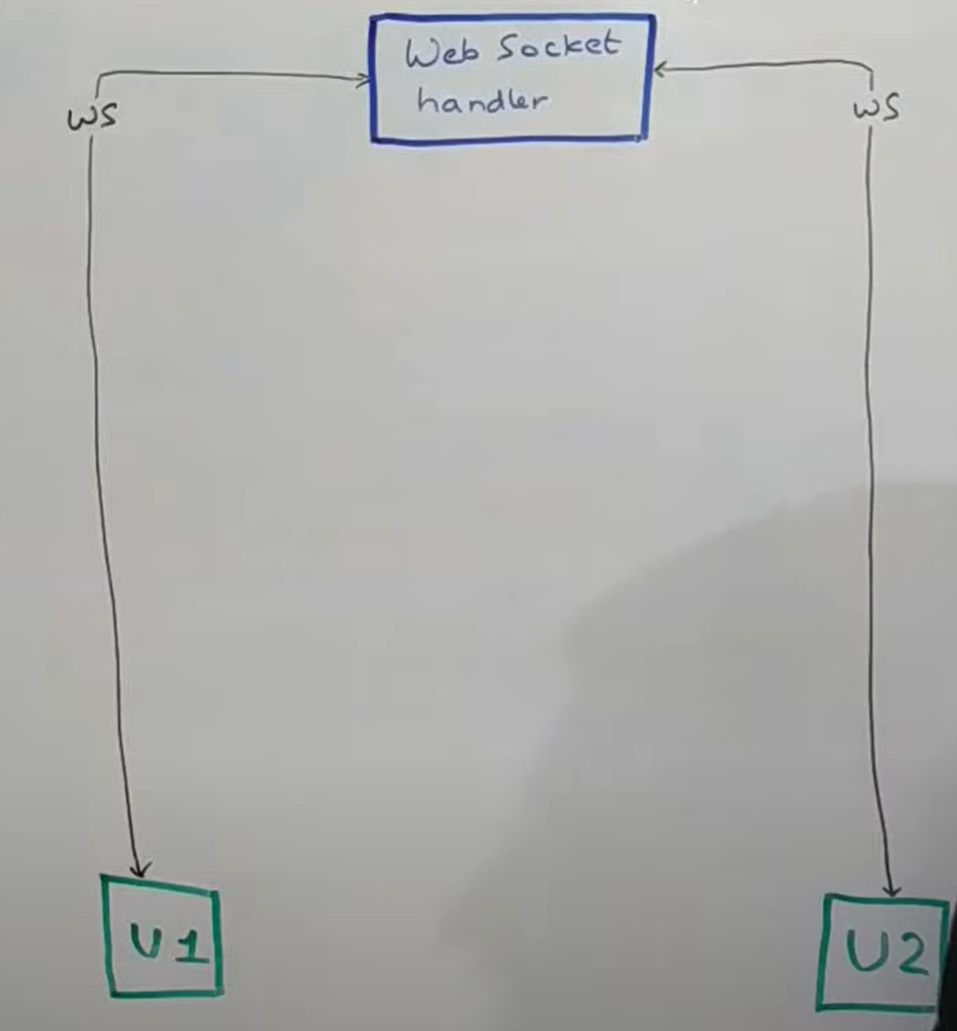
**Why TCP and UDP are used?**

The TCP and UDP transport protocols handle much of the data transferred over IP-based networks. TCP offers accurate delivery between two locations but requires more time and resources. UDP requires less overhead and lower latency but cannot guarantee that every datagram will be delivered.

**But in our case we will use UDP since couple of frame loss while video conferencing that fine.**

Now when we say UDP we are not using UDP for everything. **We will be using TCP** for all the communication b/w a client and a server which does not involve video transfer

**Lets quickly looks how whatsup chatting work and we will build on top of it.**



U1 want to send message to U2 . first it will send to web socket handler and lot of other component will involver and it will send to u2.

Web socket handler build on top of TCP for video calling hander we want to do it on top of UDP.

**Let say how we can establish connections b/w u1 and u2 instead of web socket handler for video calling.**

UDP fundamentally is connectionless protocol so there is no connection as such. Whenever somebody want to send a udp message , they essentially say that send this particular message to this particular ip address at this particular port . that’s what actually happens.

U1 understand how to send message to u2 in udp and vice versa.

**Now what does u1 need to send message to u2.**

It essentially need u2 public ip address and u2 need u1 public ip address and vice versa to send a message.

Diagram

Description automatically generated

Now there come something called as a **connector**

Connector job is to identify public ip address of both the user and inform them.

**U1 will call connector and connector will tell u1 about u1 public ip address**  and u2 will call connector and connector will tell u2 about u2 public ip address and then through web socket handler they will interchange each other public ip address with each other and then they start sending UDP message to each other. **Now why we do need a public ip address.**  So each time machine want to talk to another machine they need public ip address to talk to.

**So all the devices on ipv6 are not a problem but on ipv4 is a problem and we need to solve because majority of users will be on ipv4**

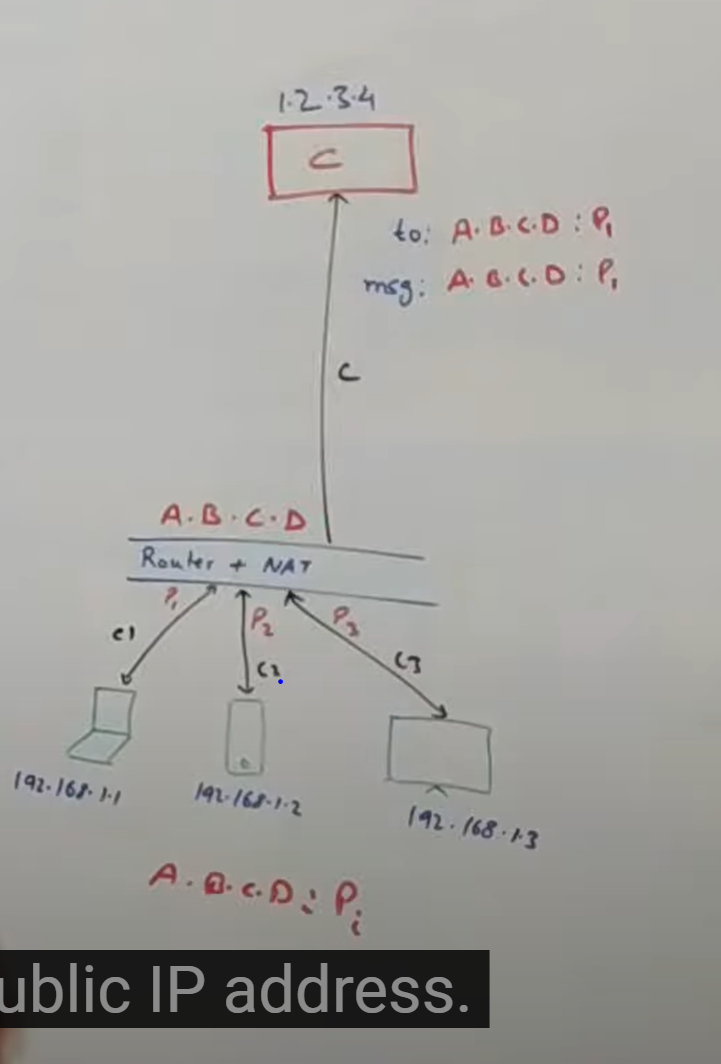
**What are the three main differences between IPv4 and IPv6?**

**IPv4 is a 32-Bit IP address, whereas IPv6 is a 128-Bit IP address**. IPv4 is a numeric addressing method, whereas IPv6 is an alphanumeric addressing method. IPv4 binary bits are separated by a dot(.), whereas IPv6 binary bits are separated by a colon(:). IPv4 offers 12 header fields, whereas IPv6 offers 8 header fields.

**And**  what happens with ipv4 – most of the devices would be connected to their home wifi router for example : **through a private ip address** so they wouldn’t have public ip address directly. Then their router would also not have public ip address . that again would have some private ip address. And at the end there would be ISP router which would essentially have public ip address. So we need to basically whatever ISP router u2 has that could become the public ip address of u2.

**Is ISP the same as Wi-Fi?**

**No, an internet service provider (ISP) and a WiFi provider are two different things**. The main difference is that an ISP provides access to the internet often through cable, digital subscriber line (DSL), fiber, or satellite connections.



**How each of the user identify their own public ip address in case of ipv4.**

Let say connector has ip address of 1.2.3.4 and these are three devices who want to understand their own public ip address. Let say laptop has ip address of 192.160.1.1 which is a private series of ip address. Lets say if a laptop wants to identify its public ip address then what exactly happens. This laptop then talk to ISP router via a lot of routers . it could be talking to home router then couple of more router in between. At the end it will talk to isp router. Let say this laptop establish connection with this **isp** router on port p1. Externally let say the public ip address of this router is A.B.C.D. these router(isp) will have firewall . they will have something called as a network address translator and a lot of things. **So this nat ( network address translator) convert this public ip address into this private ip address**.

Let see how that works

So this device would send a message to this connector asking for it public ip address. If u send a request to 1.2.3.4 that request will go here . this router will forward the request to 1.2.3.4. now when this connector get a request it will get to know that I have got a request from A.B.C.D from a particular port let say p1. If this was connected to port P1 of this particular router then connector will get to know that p1 was the port through which I got the request. So in the response it will send this kind of message . it will say to the same ip address A.B.C.D of this router on this port let say it is P1 in this case. It will send the same message saying your public ip address is **A.B.C.D:P1**  so this device will get to know my public ip address is A.B.C.D on port P1. So this is how device get to know their public address.

Diagram

Description automatically generated

Assuming both the parties (u1 and u2) have got their own public ip address

Public ip address of U1 : IP1:P1(port)

Public ip address of U2 : IP2:P2(port)

Once they got this information they will again talk to websocket handler . u2 will say my public ip address is this inform this to u1 and u1 will say that my public ip address is this inform it to u2. Now websocket handler will inform both the other parties that these are the public ip address . now you can go talk to each other.

**There are couple of other things that also happen here.**

**S**uppose u1 very high speed and u2 low then sending hd video not perform.

So both the parties will exchange a couple of attribute about their own connection their own configuration and then decide on a particular configuration for the video call. So both the parties will inform what kind of network bandwidth thay are on. They will also inform the kind of codec they support. And they will also inform about their resolution and stuff like that.

**Lets say u1 is on a mobile with lower number of pixel then the communication will probabl happen on lower number of pixel.**

**Now**  that handshake again happens through the same route via websocket handler . now once all of that is finalized then is when the actual video call begins. That could be a video call , audio call ,screen sharing. Now it is just a stream of packet which is being sent to either this ip address or this ip address.

**Now each device does two things**

It sends a payload and it receives a payload. So the logic of how to show something on the UI or the application would be taken care of by the client itself. So each of the application running on these devices would take care of how do I render the stream. Of input I m getting and how do I render stream of output that I m sending out so all of those would be taken care at that level.

Note :

**Some of the times firewall do not allow udp traffic**

A person standing in front of a whiteboard

Description automatically generated with medium confidence

U1 send message to call server and call server send to u2 and call server identify public ip of two device as we talk earlier . this whole process is called **web rtc**

**What is WebRTC used for?**

WebRTC (**Web Real-Time Communication**) is a technology that **enables Web applications and sites to capture and optionally stream audio and/or video media, as well as to exchange arbitrary data between browsers without requiring an intermediary**.

**Let see how this work when there are multiple people in call**

Small number of group : 5 user

Large number of group : more than 5 user

A picture containing text, whiteboard

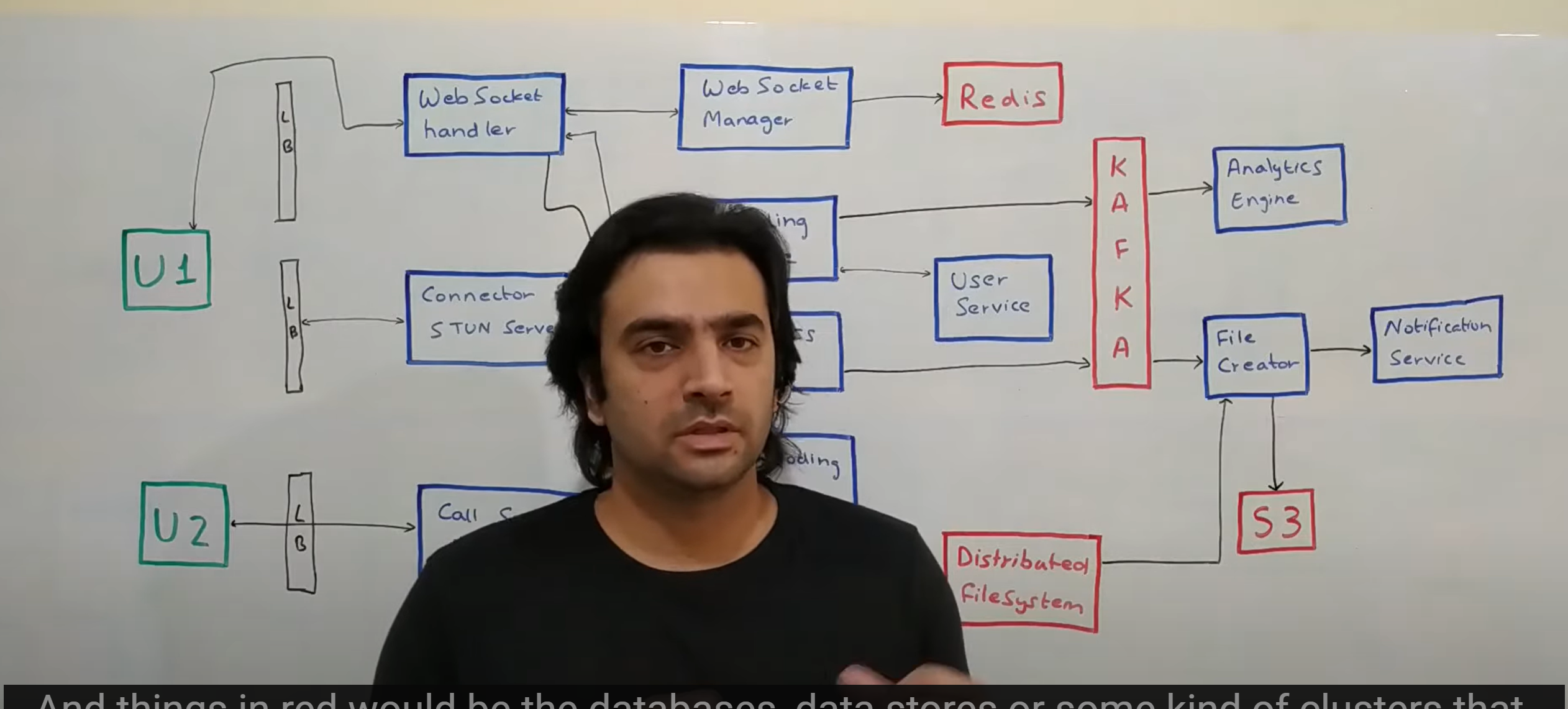
Description automatically generated

Peer to peer communication not possible since u1 need to send data to all other.

**U1 will just send this piece of information to call server . call server would then send it to all other three user.**  Each user is just sending once to caller server and receiving from call server. Now this call server will get a lot of loads in this case but atleast the user bandwidth utilization are still in control. We can increase the bandwidth of call server which is in our control but the user internet connectivity is not in our control.

**Now let look at overall architecture of the whole system**





**Now** if you look at any video conferencing platform they always have a chat system along with that. So the architecture diagram that we will see now is basically the extension of the architecture diagram that we look at in the whatsapp system design.

**Let go with some convention:**

Everything in green is user interface.

Things in black are the load balancers + reverse proxy + authentication authorization layer.

Things in blue are the web services or udp services that we have developed.

Things in red would be d/b , datastores or some kind of cluster that we use.

So the flow begins at a point when a particular user let say u1 wants to initiate a call with a user u2 . now all the users who are live in the system are connected to one of the backend components called as web socket handler very similar to whatsapp chat implementation. This **websocket handler**  keeps live connection with all the users who are active in the system. And there is something called **websocket manager**  which keeps track of which machine of websocket handler is talking to which user. There are 100 of web socket handler so that’s the reason web socket manager who orchestrate which machine is talking to which user. So in case there is a communication need to be establish b/w two users web socket handler can talk to web socket manager asking for which is the other web socket handler machine that is talking to user u2 something of that sort. Now websocket handler is built on top of **redis** . it basically uses redis as a data store.

Now let say **user u1** want to initiate call with **user u2. It**  makes a call to web socket handler which basically call something called as a **signalling service. Now**  signalling service is the one that initiate the conversation b/w two parties. It will also have some policies which check for certain condition. So let say if we have something like **fb** and you want to put restriction saying if you are not friend then you cannot make a video call then all of those thing would come within **signalling service.**  It will also use a component called **user service**  which is basically a repository of all the users in the system to power such functionalities. Now signalling service will basically talk t u2 and basically tell the web socket handler of u2 to ask u2 , if u2 want to accept the call or not. If u2 reject the call or if u2 is offline then it can be handled in certain ways. But if let say u2 accept the call then signalling server basically inform u1 that u2 has accepted your call. Now we can initiate the conversation.

The come something called a **connector**

Now both u1 and u2 talk to **connector component**.  **So this**  component is responsible for telling each of the parties about their publicly accessible ip address. Now through connector will get to know their public ip address. And again through this similar websocket handler via signalling server they would communicate these two things to each other. So what kind of codec, bandwidth all those thing share via websocket handler. Then regular peer to peer connection establish b/w u1 and u2 and they can start sending packet to each other and continue with the video call.

**Now the way video call actually work**

Is those video call would not be bigger chunk like five five second each. But in this scenario we will use much smaller chunk which is of a few millisecond kind of a thing.

Now let say it not able to establish peer to peer connection for whatever reasons may be public ip address not there then comes something called as **call server** . now each of them communicate to each other through this **turn server. U1**  send a message to turn server and this server send it to u2 and vice versa so this basically becomes a point which is orchestrating communication b/w both the parties. This is how your regular one to one connection or a regular one on one video call would work out. Now bandwidth is not constant so in the middle of call u2 ( low bandwidth due to some reason) it should be able to talk u1 again via the web socket handler saying now I don’t have enough bandwidth. Those thing could be changed in the middle of the call as well again via the same route of web socket handler and signalling service.

**Now each time there is such a change happening in the video call** that does impact the experience a little bit atleast. So whenever such thing happen signalling service would basically put out an event into **kafka**  saying bandwidth was changed . it cab be further used for some kind of processing to figure out how stable the networks are , in which geographies and based on that we can make some business decision. That is something we need to capture. Along with that these devices could send some analytics information along with everything else like may be how frequently the wifi is changing what kind of battery usage is there while being on a video call how much data is being consumed while on the video call. All of those details each of the devices would sent to websocket handler and websocket handler would send to **analytics service** which is basically purely a system that is taking input for analytics . **Analytics service then put the events into kafka** which we look at later on how can we process.

**For recording of conversation b/w two people** will come at **logger service.**  In case it is recording based system peer to peer communication would not happen. All the communication happen through this call server . **now when u1 send message to call server to send u2 call**  server not only send to u2 but it also send to something called as a **logger service. The**  logger service then basically aggregates all of those small small chunks of data and keep storing them into a **distributed file system or**  we can use s3 or we can dump into Hadoop. But the idea is that this logger service would be storing all of those small chunk in a permanent data store that is fault tolerant. Now it could be for a short period of time , till the time the call is active. Now once the call get **disconnected** b/w u1 and u2 via websocket handler , signalling service would get to know that the call has been finished. **Signalling service**  **would**  then put an events into kafka saying the call has been finished.

Then come something called as a **file creator. The**  idea of this whenever call is terminated this file creator would listen to that event. Each call would have a meeting id kind of thing. And file creator get to know about this meeting id has terminated. Similarly all the file chunk that were stored in the distributed file system would also be get stored against that meeting id. So this **file creator**  would basically read all of that data which is basically all the chunks of that whole conversation process them make a video out of it and store it in another location s3. Now once it has created a file , it can send a notification to the users who are involved in the conversation saying your meeting has been recorded here is the file , here is the link , go access it so that’s how this flow would end.

**Now lets look how does a group conversation happen**

Group conversation we were building in two modes. If there are **small number** of people like three or four people we could possibly build a peer to peer communication b/w all of them. But now each packet of information would be sent to all users. That is one way.

The other way is through **call server in**  case we have more number of users let say hundred of users then each device talking to hundred of others would be too much of trouble to manage. So then call server comes in. group conversation is slightly tricky because there we need to do something called as **transcoding . the main idea is**  let say there is a source of video and there are multiple users who are trying to access it . each user might be coming from a diff kind of bandwidth availability , different kind of codec that it can support , diff kind of file format that it can support so the idea of transcoding is to convert the original video into multiple such format in which you can support all the devices having a wide variety of bandwidth ranges

**Now hat happens in group video call**

**Let say u3**  **have low definition call** but u1 and u2 have definition call . now how do u do it.

So what happens is on the packet . let say u1 is talking something and that need to be send u2 and u3 so message would come back to this call server from u1 . call server knows that u2 is having high definition ability so it would send same package that it got from u1 which was in high definition directly to U2. It also knows that u3 need to have low definition video. So then it will call something called as **transcoding service** . now this transcoding service could physically from a h/w standpoint be running on the same machine to minimize the latency. But logically it’s a diff component. **So then call server talks to transcoding service which transcodes the high definition video into a low definition format and sends that to U3**.

Transcoding happen in one way : we can convert high definition video into low definition video but not vice versa. Call server would use cache based redis to keep all of that information like bandwidth of user.

Suppose u3 started at high definition bandwidth but in between it want low definition requirement so again that communication would happen via signalling service and call server would get to know that now u3 has moved to low definition mode so I need to start using the transcoding service so all of those thing would be handled by call server at runtime.

**Let look at the last component which is basically analytics**

So there is something called **analytic engine**  which will power all the analytics. This analytic engine would basically read all of those event and publish some kind of reports.

Now if u want to do very simple analytics like just figuring out how many number of calls happened in a day. So it a basic set of analytics then you can basically put all of that information in a time series database and plot some graph on top of it.

**Lets look at some intelligence that exists within the client**

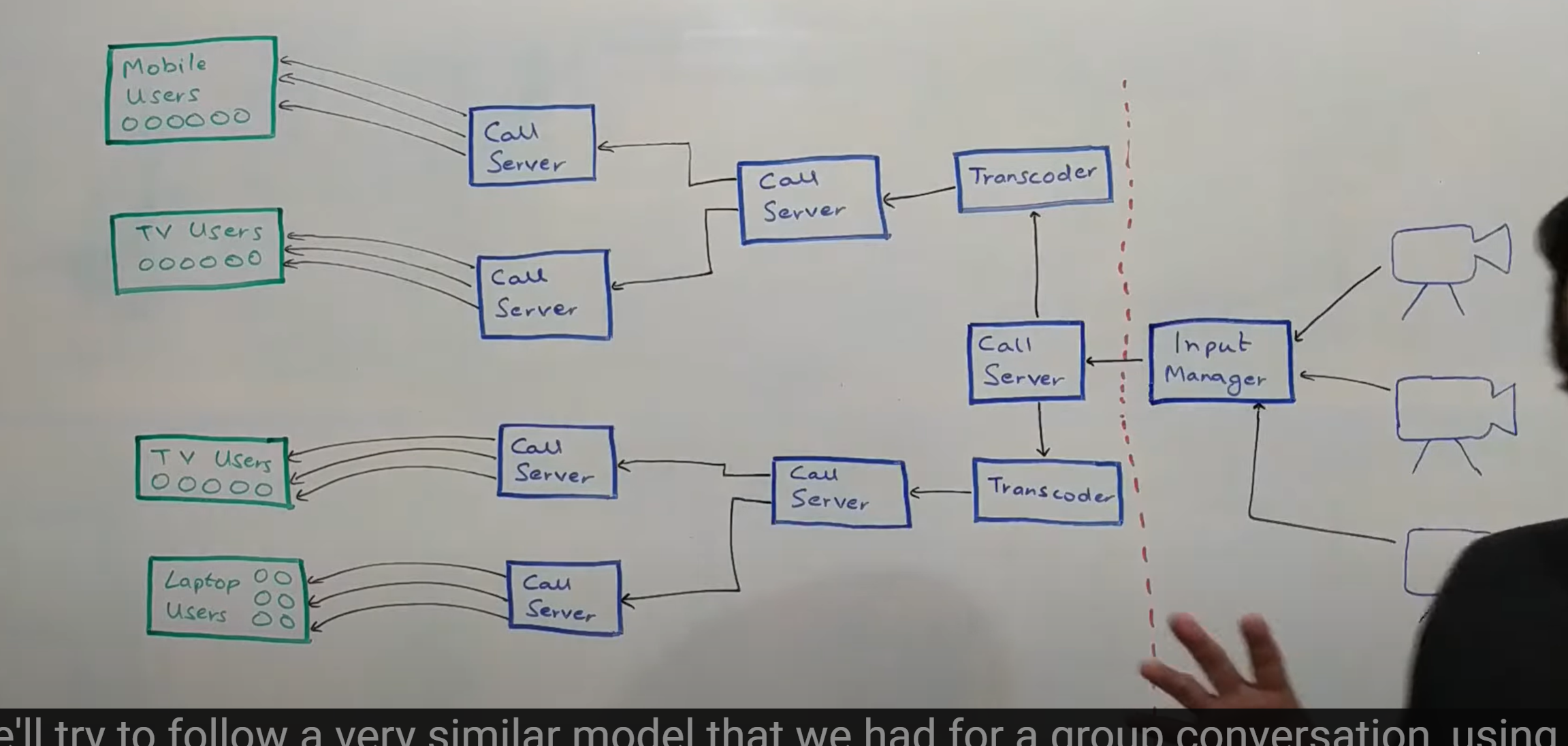
**U1**  is talking to u2 but due to some reason call is getting dropped off or let say public ip address of one user is changing. Now in that situation call will go via call manager instead of peer to peer communication b/w u1 and u2 . now how would that work

Let say these two communicate via web socket handler and signalling service and we need to now move connection to the call server. It may not happen exactly at same time. Both the client can move the communication at their own pace to the call server.

**Now same thing can happen in group conversation also**

If any issue occur in peer to per communication they will move to call server at their own pace.

**How do we this system to support a live broadcast of some massive event**



**Let**  just say there is big sporting event happening somewhere ex india vs pakistan match.

Let see how we can scale this system where million of user are trying to watch a livestream.

Any type of event there will be multiple camera and multiple audio devices taking input.

**Input manager** decides which camera and audio device out of 3 want to stream now.

It then basically send audio plus video combination into the system that need to be streamed.

Will follow a very similar model that we had for group conversation using call server. Now this **input manager** basically send the request to a call server saying this is the stream, now sends it across to whoever want to watch it. Now the thing is these kind of massive calls live stream is , there are lot of users on a wide variety of devices. Let say there are lot of users on mobile, tv and laptop.

So **call server** will basically forward the request to multiple **transcoders.**  All we need is multiple transcoders which is a function of how many output format do we want. So let say in this example we have three kind of user we will possibly have three kind of output formats and therefore three set of transcoders. So it will send that to all those transcoders which will then send the transocded output to a set of call server. Now what this call server gets is essentially the file format that would be sent out to one type of users. So in this there are three file formats so there will be three such call server an three such transcoders . I have made two just for representation purpose. Now let say that this call server get all the data that is to be sent to mobile devices. It would then send to multiple call server. Now this call server is not directly sending to user why? Because any sporting event would be seen by lot of people in a lot of geographies. So we need a cdn kind of a thing which we can first distribute in a lot of geographies and the from those geographies we can distribute out to multiple users. So this call server would send to multiple call server and all these server would be geographically distributed across the globe. And then individual call server will talk to one type of user. So one call server talk to mobile user so it will talk to 1000 user but only mobile server. If the call server close to the user that latency would be minimized.