November 2017

WebRTC for Live Media and Broadcast

Second screen and CDN traffic optimization

Author: Jesús Oliva
Founder & Media Lead Architect

epic labs

It is not a surprise if we say browsers are evolving so quickly and they have become one of the most important apps of our computers. From the time they were limited rendering engines with basic scripting support, the few needed to render a few snowflakes in your web during Christmas period, a lot has changed. I remember how amazed I was when I saw Google Wave for the first time, or later on when Google Docs was released. It was like having all my office tools within the browser!

As time goes by more applications are getting into our browsers. Their rendering capabilities are growing and scripting engines are becoming so powerful and flexible that they are even used as the core of backend technologies (Chrome V8 and Node.JS). There are more and more new APIs that enable us to implement new and enriched experiences: canvas, WebGL, HTML5 video and audio,... Did you imagine 15 years ago that someday you could run Quake III within your browser? Me not.

This evolution is not only affecting how we can render things in the screen or the performance and efficiency of javascript engines. It is also coming with very powerful APIs and technologies that bring us the possibility of implementing products that challenge the old concept of web apps based on HTTP or, in general, any application that requires any kind of interactivity across any number of users and then needs low latency communications.

What is WebRTC

WebRTC is a set of technologies and protocols that allows real time communications (video, audio and/or data) between two or more peers. It doesn't use HTTP, it doesn't require a server. And the most important thing: it is integrated in most of modern browsers. This is, Chrome, Firefox and latest versions of Microsoft Edge and Safari.

This means that with a very simple API exposed by your browser and accessible from Javascript you can start implementing applications that require real time communication features. Just think of the possibilities of having a low latency bi-directional communication channel that allows direct and secure connections with other users at your disposal. Some of those applications could be video conferencing systems, applications in which two or more users can send data to each other without using an intermediary server (P2P mesh network) or, in general, any application that requires any kind of interactivity across any number of users and then needs low latency communications.

Copyright Epic Labs 2017. ©

Going Beyond WebRTC Limitations

Of course, nothing comes without a price. For us, developers, the possibility of having real time communication in the palm of our hand is great but it also comes with some responsibilities. The responsibility of making it scale. As an example, imagine we are building a web application that uses webRTC to implement an application in which one user (the sender) publishes video content that can be watched by any number of watchers. If there are 100 users watching the stream, the sender will encode her video just one time, but will need to send it independently to each one of the watchers. Doing some numbers, if the sender is encoding her video with a bitrate of 1Mbps, she will need 100 Mbps of sustained bandwidth for making this work. It sounds like too much for not a huge number of users, right?

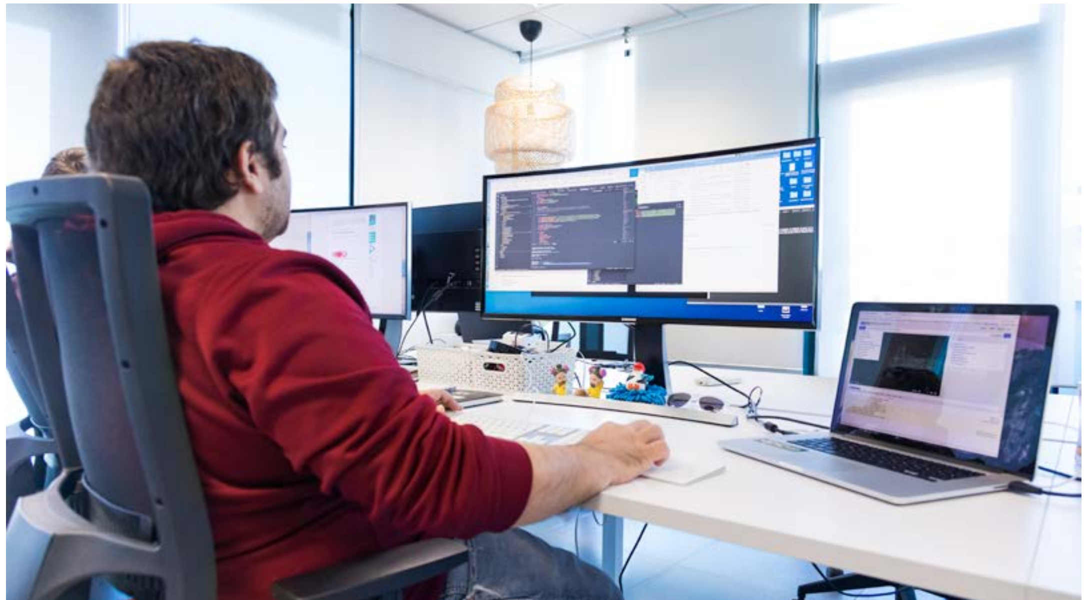
But that's not all. In video/audio communications, WebRTC automatically adapts video quality, taking into account the conditions of the network connection. This is done with the target of ensuring video/audio is transmitted with low latency (200-500 ms), key aspect of WebRTC. This sounds great. You get this functionality from the browser without implementing your own bitrate or quality adaptation algorithms. But how good the bitrate adaptations works when you have an enough big number of watchers? Coming back to our example, the way it works is, if one of our 100 watchers has network issues or bandwidth limitations, the sender will encode her content with such a reduced quality that can guarantee low latency for this limited bandwidth watcher. In other words, a network issue in one of our 100 watchers will affect the video quality received by all of them. Not so good.

Do these limitations mean that we cannot use WebRTC in production environments in which we have an enough big number of users? No, not at all, we just need to keep in mind its "limitations" and design solutions that overcome them. How easy is to develop a web server using NodeJS? So easy. How easy is to develop a web server using NodeJS that can scale to attend 1 million of users per second? So complicated that everyone can't do it. The same applies to WebRTC but with the extra complexity of scaling real time communication based applications.

There are ways to overcome WebRTC scalability limitations to enable more and more use cases. Including the ones thought for television consumption environments in which you can use WebRTC to provide interactive or low latency streaming services to millions of users. You just need the right expertise. One option is to design a solution that moves the complexity of the scalability to a Media Platform on the cloud which acts as an intermediary in the communications. In this kind of solution, any sender will do just one transmission, the one to the Media Platform, and it is the responsibility of the system handling all delivery and routing process across the different connections/watchers. In addition,

Copyright Epic Labs 2017. ©

different capabilities can be built through a cloud service provider, lifting machines as needed to ensure scalability, if necessary. With this approach, different business logics can be assembled for a wide variety of use cases that go beyond videoconferencing.



What can be expected in the Media Broadcast environment?

In today's television market, whether we are talking about traditional media or those in the Web TV and OTT environment, this technology allows us to open a channel with users to send and receive information in real time. This can be particularly useful in second-screen applications associated with live sport events, for example. In motor sports, it can be used to generate data feeds that deliver real time information about the race, or it can also be used to stream cameras and real time video feeds that are not available in the main broadcast.
WebRTC's ability to exchange both video and data at low latency enables interactive video experiences. Conversations (small video conferences) can be held while the event is taking place and data can be accessed.

For broadcasters, ideally, and with the goal of keeping latency as lowest as possible, the SDI signal should be converted to a WebRTC stream as early as possible. Although there is no product to perform this particular repackaging/ transcoding operation, it can be done knowing well the specifics about SDI, and WebRTC encoding and muxing characteristics. At the end of the day WebRTC is not re-inventing new protocols, it is leveraging the use of existent and widely

Copyright Epic Labs 2017. ©

used technologies (SDP and SRTP, besides others) and there are ways to achieve complete interoperability between it and protocols used in broadcast environment.

Additionally, an advantage of WebRTC is its protection schema. Every packet sent is encrypted to guarantee the security and integrity of the information. This is one of the fundamental pillars of WebRTC's definition since its inception by Google and later on by the W3C, which is responsible for defining the standard. The integrity of communications is therefore guaranteed regardless of its application or the content that travels through them.

The biggest problem that a TV station or a TV Platform faces when it comes to working with this technology is scalability. It is not easy scaling applications that deliver real time content or data. As explained, WebRTC, when used for video/audio transmission, does whatever it takes to keep latency low, and that can also be a drawback because it may involve degradation of video quality. If premium live content is being consumed, the screen cannot go black because of insufficient bandwidth. The key is to find the right balance between quality and availability of the service, that depends on the kind of application we are building and on how the QoE affects its users. It is not the same a service that has to support real-time video with high quality on a mobile device, or a second screen or complementary one that will support a layer of interactivity in real time at the same time that a main broadcast occurs.

Copyright Epic Labs 2017. ©

Use Cases for live content

1 – Second screen Applications

WebRTC allows second screen experiences for live premium content, enabling alternative video sources and interactivity. This field admits a lot of creativity. In the second screen it would be possible to include alternative cameras that cannot be accessed in the main broadcast and all types of data feeds. For example, in a racing car competition all kinds of statistics, times, maps, biographies, etc. can be displayed.

In addition, WebRTC allows communications between users through the same channels. We can talk about a simple chat or videoconferencing between several users in real time while you are watching the event and commenting on the application data on the second screen. This is something that works in this technology natively, without the need for a server for conversations between users to take place.

2 – P2P complement to a CDN

WebRTC's capabilities as a peer-to-peer technology enable very interesting use cases in the CDNs environment, even though they may appear to be antagonistic technologies. One of them is the ability to establish a serverless P2P network that helps the dissemination of content through WebRTC data channels. This makes each user can operate as a small CDN, sharing content that is being downloaded with other users, without the need for any type of server in between. This generates several benefits, the first one is the ability to reduce traffic from the CDN as an efficient cost reduction strategy.

The second, as an enabler of such high bitrates that are difficult to reach using a CDN. Both cases have many applications but one of the most exciting is their use in the broadcast of large live events, such as a Champions League Final, where many people will be watching the same content simultaneously and in proximity to each other. This type of application is perfect for regionalization, feeding through P2P the content coming from the CDN and guaranteeing its quality at all levels. CDNs are perfect for taking content anywhere, but they have problems scaling very steeply in very specific areas, and that's where WebRTC and its P2P capabilities can ease the load on servers that are deployed around the world and ensure content delivery anywhere. Although these types of architectures are not simple, at Epic Labs we have the knowledge to deploy them and make WebRTC a natural ally of traditional television, OTT TVs and CDNs.

Copyright Epic Labs 2017. ©

epic labs

Jesús Oliva
Founder & Media Lead Architect

- ✔ Software Architect
- ✔ Backend Engineer
- ✔ Data Engineer

Biography

A vocation for innovation.
Jesus is Founder & Media Lead Architect of Epic Labs, a Software Innovation Center specialized in media, video and streaming. An engineer at heart, he is an early software engineer with passion on distributed systems, video processing and encoding algorithms, media protocols and WebRTC. He currently leads the Epic Labs Team driving the reference client implementation of DASH IF, Dash.js.

Before Epic Labs, Jesús worked at Akamai doing Media Products engineering where he received several performance and innovation awards.

Copyright Epic Labs 2017. ©

epiclabs.io

epic labs

Calle Salvatierra, 4
28034 Madrid
+34 666 069 988
info@epiclabs.io