An Introduction to Selective Forwarding Units

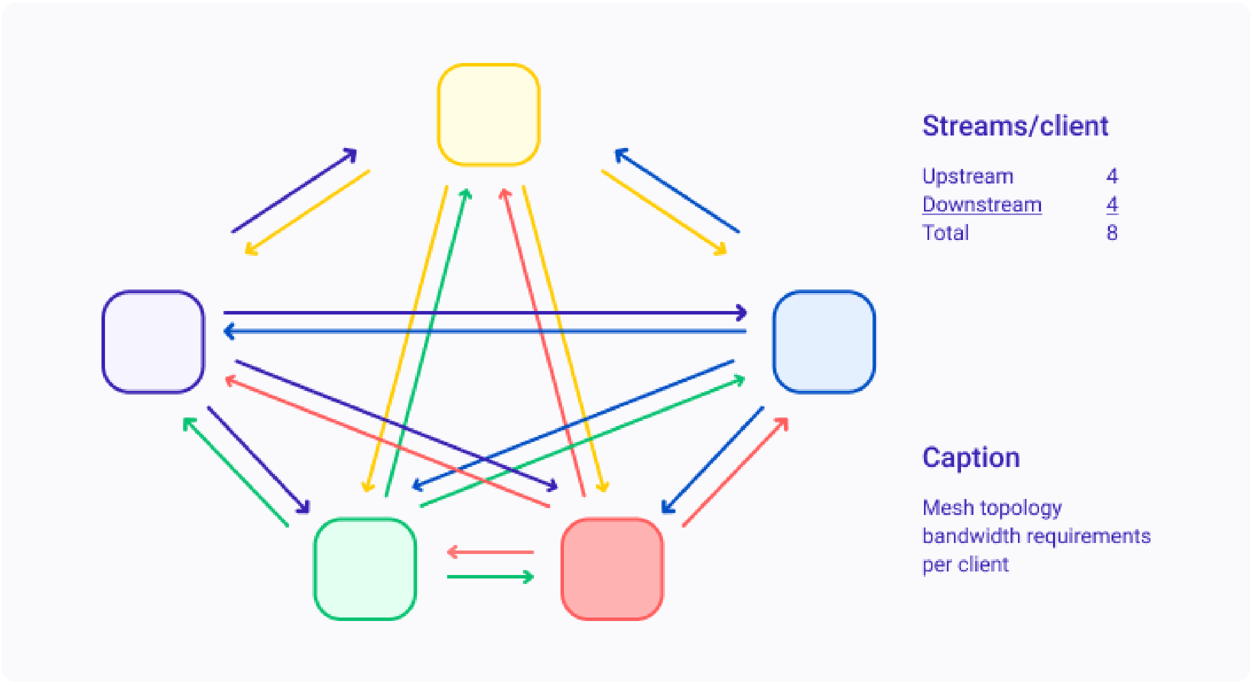
Πηγή: https://voximplant.com/blog/an-introduction-to-selective-forwarding-units

Adding peer-to-peer communications to an application is relatively straight-forward. Developers can leverage WebRTC APIs or a CPaaS service to quickly add real time voice and video to their web or mobile app. But, what if you want to hold a meeting with more than two people? How can you leverage powerful WebRTC APIs to build a multi party conferencing application?
Application developers have settled on the selective forwarding unit (SFU) as the preferred method of extending WebRTC to multi party conferencing. SFUs enable you to deploy WebRTC in efficient and scalable hub-and-spoke topologies with low latency and high quality of service. Recent simulcast enhancements are further improving this essential component for any conferencing service.

In this blog, we'll help you understand how an SFU fits into your conferencing application. We'll describe the key features and functions, review the range of DIY and CPaaS services available to help you add it to your application, and describe the SFU services available in the Voximplant CPaaS platform.

Multi party conferencing with WebRTC

Even though WebRTC was designed for peer to peer communications, it is relatively easy to build a multi party conferencing application with it. There are multiple approaches offering trade-offs in scalability, cost, quality and security. The simplest approach is to build a mesh topology in which every participant sends and receives media from every other participant. This maintains the end-to-end security inherent in WebRTC peer connections and offers the lowest latency and highest quality of service. However, a mesh topology quickly reaches scalability limitations. It consumes a lot of bandwidth and client processing power to manage all the media streams. In particular, the compute burden can be a significant limitation for mobile devices with limited battery power.

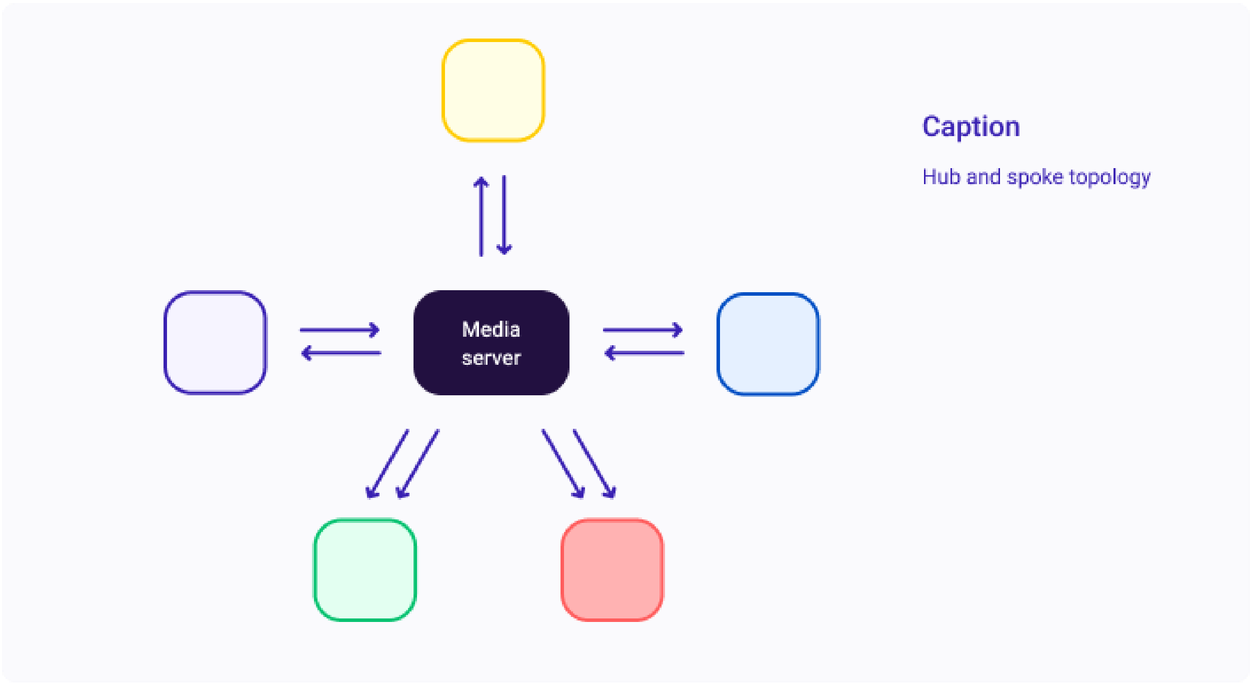


Upstream	4
Downstream	4
Total	8

Caption

Mesh topology bandwidth requirements per client

To increase scalability, you need to build a hub-and-spoke topology by inserting a media server into the network. A hub-and-spoke topology reduces the amount of network bandwidth and client CPU cycles required because the server takes care of replicating media streams for the clients. The amount of savings varies based on the type of media server used.



Caption

Hub and spoke topology

Hub-and-spoke topologies can increase latency because media must traverse a longer path from sender to receiver. For this reason, you should carefully consider the geographic placement of media servers relative to where clients are located.

In addition, the hub-and-spoke topology introduces an intermediary between clients that breaks the WebRTC end-to-end security feature. Encrypted media streams transmitted by clients are terminated by the media server, which originates a new encrypted stream for transmission to clients. Without the incorporation of additional end-to-end encryption (E2EE) techniques, a bad actor could potentially monitor the media as it passes through the media server. You should review regulations that apply to your industry and determine whether this is permissible.

Media servers for video conferencing

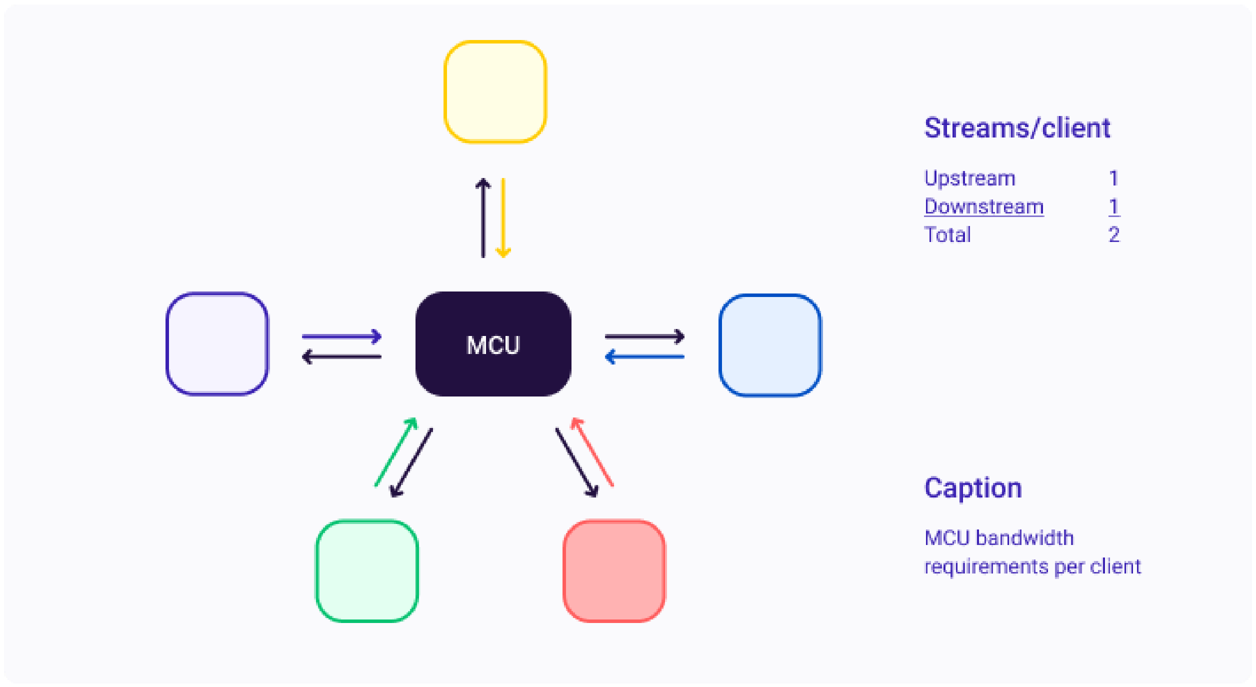
Two types of media servers can be used to implement the hub-and-spoke topology: The multipoint control unit (MCU); and the selective forwarding unit (SFU).

What is an MCU?

The MCU decodes each received media stream, rescales them, creates a new tiled stream featuring all participants, encodes and sends it to all clients. It may adapt each transmitted stream to the network conditions available for each client. For example, it may send 1080p video using VP8 to clients with good conditions, 720p to browsers with some bandwidth restrictions, and VGA using H.264 to mobile devices.

The MCU is an expensive and compute-intensive infrastructure element when used on high-bandwidth streams like video. Encoding at multiple resolutions places a heavy CPU burden upon it. The MCU scales higher than the mesh, but server CPU capacity can limit the maximum conference size. Another drawback: the MCU typically presents the same image to all participants; the participants have limited flexibility to change the tile arrangement on their screen without contorting that image.

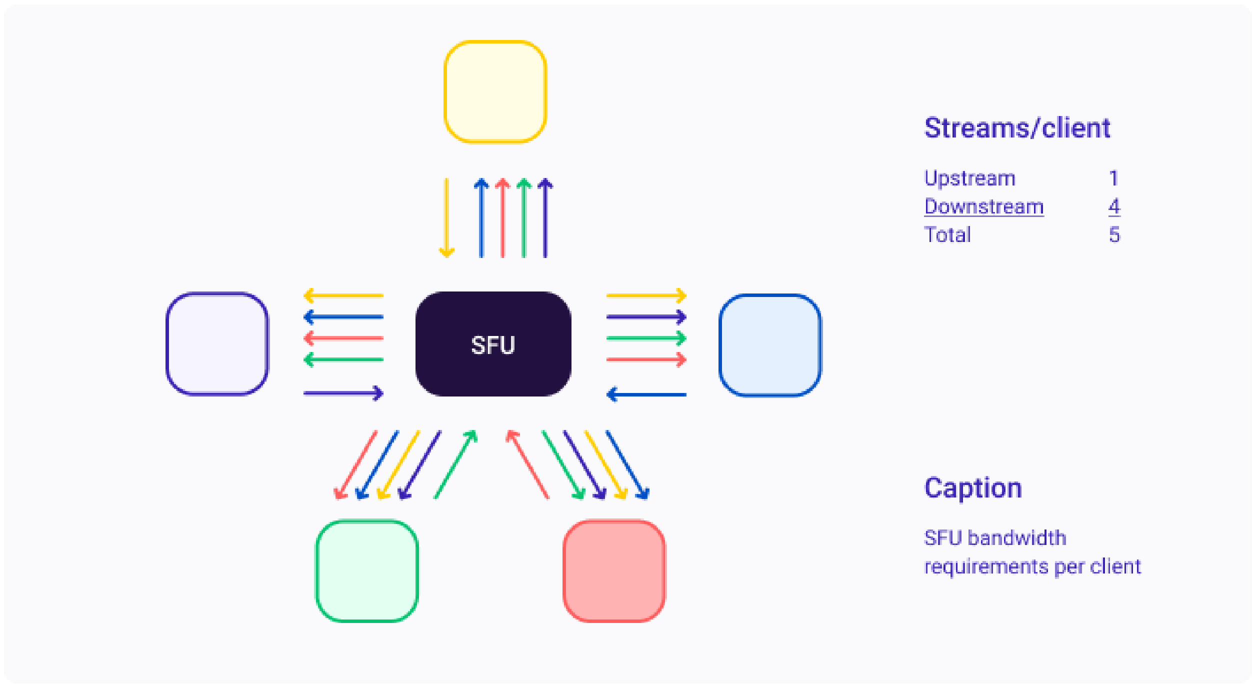
On the other hand, the MCU completely relieves clients of local processing and it is the most efficient of all the alternatives in its bandwidth utilization.



What is an SFU?

The SFU strikes a compromise between the mesh and MCU by limiting its manipulation of the media. The SFU receives media streams from each participant and merely forwards them to the other participants without changes. It does not perform any decoding and encoding of streams, which burdens the CPU of an MCU and adds latency. This makes the SFU more scalable and lower cost than an MCU, plus it delivers better quality of service.

SFU bandwidth efficiency is better than a mesh topology, but lower than an MCU. In contrast to the mesh, the SFU saves the upstream client link from carrying media addressed to all the other clients. But, the downstream client link must carry n-1 media streams, where n is the total number of clients participating in the conference (assuming the desire to view everyone at the same time).

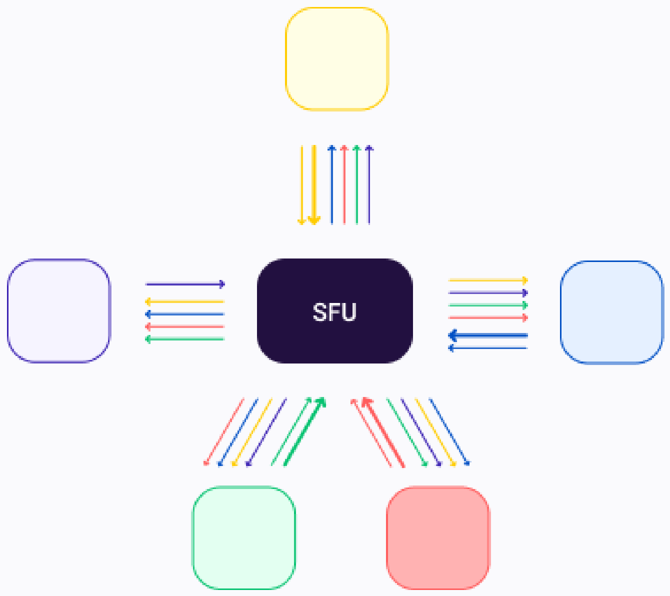


Because each client receives video from all other clients, it has flexibility to arrange the streams in any presentation preferred by the user. The user can choose to display the active speaker, tile view or any other arrangement. This makes the SFU more flexible than the MCU.

However, in an SFU architecture, the client with the least bandwidth dictates the video quality available to all clients. This is because video codecs dynamically adapt the amount of media data they transmit based on information the receiver provides about its available bandwidth. The codecs adjust resolution, quantization and frame rate so that packets aren't lost due to congestion. If one participant is sending “too much data”, all the other participants will throttle their sent bandwidth, reducing the video quality for all participants, not just the one with bandwidth issues.

Simulcast SFU

Simulcast is the latest advancement in SFU technology. It is designed to prevent a few clients with limited bandwidth resources from degrading the video quality available to all. Here is how it works: Using a common agreed codec, each client transmits its video stream in multiple quality levels. The SFU forwards the resolution preferred by each client, based on its available bandwidth.



Caption

Simulcast SFU with transmitstreams in two resolutions; purple client (mobile device) is connected by low bandwidth network.

Simulcast alleviates the lowest common bandwidth problem. Each client has access to the highest quality streams that its local network bandwidth can support. Quality limitations affect only the streams transmitted by clients connected to poorly performing networks.

As an example, consider 4 laptop clients connected to a simulcast SFU by a broadband network and one mobile device connected by 4G. The laptops transmit high and low quality streams, while the mobile device has only enough bandwidth to transmit a single, low quality stream. The laptops receive three high quality streams from the SFU, plus a low quality stream from the mobile device. The laptop users enjoy high quality video from their peers without being penalized by the low bandwidth connection used by the mobile device. The mobile device receives only low quality streams so as not to overwhelm its available bandwidth.

MCU vs. SFU vs. Mesh

While there are many tradeoffs to each approach to building a WebRTC conferencing service, SFUs have become the preferred method for developers. The comparison below illustrates how the SFU strikes a compelling balance among the alternatives.

	Simulcast SFU	**MCU**	**Peer**
Infrastructure cost	$	$$$$	-
Scalability	High	Medium	Low
Client bandwidth	Medium	Low	High
Client CPU load	Medium	Low	High
Server CPU load	Medium	High	N/A
Latency	Medium	High	Low
Presentation flexibility	High	None	High

SFU services: DIY or CPaaS?

There are multiple options available for deploying an SFU infrastructure. We'll leave a complete evaluation for another post. Instead, we'll describe the two primary options and some important considerations in selecting between them.

There are two ways to deploy an SFU infrastructure: You can do-it-yourself by leveraging one of the open source projects available; alternatively, leverage the infrastructure provided by a communications platform as a service (CPaaS) provider.

This is a classic build vs buy decision with factors such as cost, time to market, and SFU features weighing in the evaluation. From a cost perspective, you'll want to compare the cost and time required to build your own SFU, plus the cost to manage and maintain a distributed server infrastructure, against the cost to purchase SFU services from a CPaaS provider. Don't overlook maintenance costs because browser technology changes frequently, requiring regular testing and updates to your SFU.

Time to market and features can tilt the evaluation in favor of a CPaaS solution. The SDKs and APIs offered by most CPaaS make it quick and easy to integrate your application and a global infrastructure is already deployed. Because CPaaS service a broad base of applications and use cases, their SFU infrastructures are typically rich with features and offer excellent reliability. In addition, the CPaaS shoulders the burden of maintaining compatibility with changing browser technology.

Regardless of which path you follow, you'll want to consider geographic coverage because latency can reduce the quality of a conferencing service. You can minimize latency by locating SFUs as close to your users as possible. If you're building a global service, you should expect to position media servers across all densely populated regions. Most providers have servers in Europe, US east coast and west coasts, Brazil and a couple of Asian locations (e.g. Japan and India). In any case, it's best to monitor latency and adjust based on usage patterns.

Open source SFU implementations

There are many open source media servers. You'll want to examine the hosting platform and programming languages supported by each project. You should also consider the size of the user community and its activity level, in case you need some help along the way. Here is a summary of the two most popular, based on Github stars: Jitsi (3.1K) and Janus (4.6K). A more comprehensive list is available here.

The Jitsi project started in 2008 and produced the JitsiMeet video bridge for WebRTC in 2014. The technology powers many commercial web conferencing services, including 8x8, Comcast and Symphony. There is a vibrant user community and the organization touts a user community that exceeds 10 million.

The Janus project was begun in 2012 as a general purpose WebRTC server. It includes server plugins for an SFU. The company claims third party applications in a range of industries incorporate its technology, including online learning, co-working, broadcasting and contact centers. There is a vibrant user community of over 10 million, according to the organization.

	Jitsi Meet	Janus
Platform	Apache	Linux
Signaling	XMPP	Custom JSON

Extensibility	Specialized conference bridge, only	Plugins available for recording, streaming, SIP gateway

CPaaS SFU services

Most every CPaaS vendor offers video bridging among its core services and all are based on SFU technology. Here are some of the key characteristics to evaluate:

- **Maximum number of participants** - This can vary significantly between providers.
- **Platform support** - Does the CPaaS have SDKs for the languages and end user environments that your developers need and do they keep pace with changing browser technology?
- **Price** - This also can vary significantly between providers based on the number of streams and stream features.
- **Features and APIs** - Do they offer simulcast support and other important features?
- **Support** - Do they offer technical support for developers in case something goes wrong?

Here is a brief summary of the offerings from leading service providers to help get your evaluation started.

The SFU enables scalable WebRTC conferences

Adding multiparty conferencing capabilities to your WebRTC application is easy with a selective forwarding unit. These media servers are vital infrastructure that enable WebRTC to scale to large numbers of participants without consuming large amounts of network bandwidth or client CPU cycles.

Developers have flexibility to deploy proven SFU infrastructure solutions using open source software or CPaaS services.