IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 31 July 2025, accepted 7 September 2025, date of publication 10 September 2025, date of current version 16 September 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3608475

RESEARCH ARTICLE

Tree-Based Architecture for Enabling Live Interaction in Massive Online Events

CARLOS ARRIAGA, ALEJANDRO POZO, AND ALVARO ALONSO
Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid, 28040 Madrid, Spain

Corresponding author: Carlos Arriaga (carlos.arriaga.prieto@upm.es)

This research was supported in part by the Spanish Agencia Estatal de Investigación under Grants FUN4DATE (PID2022-136684OB-C22) and SMARTY (PCI2024-153434), by the European Commission through the Chips Act Joint Undertaking project SMARTY (Grant 101140087) and by the Ministerio de Ciencia, Innovación y Universidades (Grant FPU21/03069).

ABSTRACT Massive online and hybrid events have become important during and after the COVID-19 pandemic. These events aim to replicate live experiences by maintaining interaction between the speakers and the audience. However, existing streaming and videoconferencing solutions fail to provide sufficient scalability and real-time interactions. Streaming technologies lack support for live interaction, whereas videoconferencing technologies scale enough. Events such as panel debates, conference presentations, and sessions with live interpretation require both the ability to support a large number of participants and real-time voice interactions. In this paper, we propose a new tree-based architecture that meets both requirements using videoconferencing technologies. This approach combines the scalability of streaming architectures with the low latency of videoconferencing technologies. The objective was to increase the maximum number of participants that videoconference providers can accept while maintaining live interactions. A prototype implementation of the proposed architecture was developed to test and validate it. Finally, this study provides valuable information for implementing and adapting the proposed architecture to various production environments.

INDEX TERMS Videoconference, architecture, WebRTC, scalability, multimedia.

I. INTRODUCTION

During the COVID-19 pandemic, videoconferencing systems experienced significant growth. Owing to mobility restrictions, most events were forced to shift from on-site to online formats, increasing the demand for multi-user videoconferencing systems. Even in the post-pandemic world, online and especially hybrid events remain relevant [1].

This increase in demand was addressed using cloud technologies, which adapt the available infrastructure to current demand, and distributed videoconferencing systems which improve scalability compared to both pure peer-to-peer (P2P) [2] and monolith server-based topologies. In pure P2P scenarios, each client must send a copy of its media to every other participant. The number of connections that each client can manage is limited by the upload bandwidth and CPU power. In contrast, monolithic server-based architectures require clients to send only a single media stream to a server. However, the server's capacity is constrained by the hardware on which it runs. Distributed videoconferencing systems enhance monolith server-based architectures by distributing computation across multiple machines. Furthermore, technologies such as HTML5 and WebRTC [3] provide web browsers with videoconferencing capabilities, allowing any user with a PC or mobile device and internet access to participate.

Despite the scalable distributed architecture, most video conference providers impose limitations on the maximum number of participants in a single session. Distributed architectures divide monolithic MCUs (Multipoint Control Unit) [4] or SFUs (Selective Forwarding Unit) into smaller components distributed across the available infrastructure.

In these architectures, audio and video are distributed by a broadcaster, a component that receives media flows (audio, video, or data) from a user and broadcasts them to the rest of the users. Using cloud computing technology, infrastructure can be scaled based on user demand. However, in scenarios with a large number of participants, a single broadcaster may exhaust the resources of the machine on which it is deployed. A broadcaster is an atomic unit that cannot split between multiple machines.

Scenarios that involve such a large number of participants and require real-time interaction include panel debates broadcast to large audiences, congress presentations with

The associate editor coordinating the review of this manuscript and approving it for publication was P. K. Gupta.

questions from the audience, sessions with live interpretation, etc. Live streaming platforms such as YouTube or Twitch can reach millions of viewers, but introduce delays that prevent real-time interaction. Some hybrid solutions connect a small interactive group and stream their feed to a larger, delayed audience. However, this still fails to resolve the scalability challenge in real-time sessions.

To address this gap, this study proposes a novel tree-based architecture to further increase the scalability of existing videoconferencing providers. The proposed architecture builds on a reference model that orchestrates the deployment of broadcasters across multiple machines [5], but does not allow a single broadcaster to scale over multiple machines. Our approach orchestrates the interconnection of broadcasters to cascade media to clients. This enables load distribution across multiple servers and overcomes the limitations of single machine deployment.

WebRTC, originally designed for low-latency communication under varying network conditions, is now used in a wide range of applications beyond traditional videoconferencing, including Virtual Reality (VR), telehealth, and live streaming [6]. The proposed architecture is application-agnostic, making it suitable for diverse applications.

The remainder of this paper is organized as follows. Section II describes the scenario and limitations of current videoconferencing solutions. Section III reviews related works on tree architectures in videoconferencing. Section IV presents the proposed solution for increasing the maximum number of simultaneous subscribers. Section V details the prototype implementation used for testing and Section VI discusses the results obtained. Finally, Section VII presents conclusions and potential directions for future work.

II. SCENARIO AND PROBLEM DESCRIPTION

In this section, the limitations presented by the distributed OneToMany (dOTM) architecture [5], [7] as well as the limitations imposed on online events, are reviewed.

A. USE CASE DEFINITION

As stated above, massive online events are limited by existing videoconferencing architectures, because they are not designed for this specific scenario. The purpose of this study is to design a new architecture that supports a higher number of participants while maintaining interactivity. The main focus and primary use cases are online events in which a few participants are actively involved while the majority are spectators. This contrasts with traditional videoconference sessions, in which all users typically participate. The defined scenario includes events such as congress presentations, panel discussions, large company events, and sales kickoffs.

However, to maintain interactivity and allow spectators to participate in the event, they must be able to convert seamlessly to an active participant. The event experience can be greatly improved by allowing the public to ask live questions or participate in discussion [8] as if they were attending in person.

TABLE 1. Maximum number of videoconference participants from different services.

Company	Maximum Participants
Zoom	500
Google Meet	500
Webex	1.000
Microsoft Teams	1.000

TABLE 2. Maximum number of participants for event services.

Company	Participants	Interaction
Zoom Events	50.000	Written
Webex Events	100.000	Written
Microsoft Town Halls	20.000	Written
Amazon IVS	10.000	Voice

All the commercial videoconference services have a maximum number of simultaneous participants, as shown in Table 1. All of these values refer to business or enterprise plans.

In addition to traditional videoconference rooms, most providers offer alternative services for hosting large events, as presented in Table 2. With these services, only a few preselected participants are speakers and almost all of them are spectators. In addition, multimedia data are broadcast using technologies that do not allow for live interaction and rely on a chat or written questions to achieve interaction. Only Amazon offers live streaming services[1] that allow participants to participate via voice. However, they only offer the infrastructure and no application is provided. This service will be discussed in further detail in Section III.

B. DOTM ARCHITECTURE AND LIMITATIONS

The base architecture analyzed was the dOTM. Previously, videoconferencing systems were monolithic MCU/SFU servers that distributed the media. MCUs decode every participant stream and create a single joint stream for each participant. Because the decoding and encoding process is computationally expensive, SFUs were designed to reduce the associated costs of deploying MCUs. SFUs receive the media from each client and forwards it to the other participants without processing it.

The dOTM model breaks down monolithic SFUs into a series of components that can be independently scaled according to the scenario. The core component is the broadcaster, also known as One-to-Many (OTM), a process that distributes media based on the publisher-subscriber pattern. It receives a media stream from a publisher and distributes it to all the clients who subscribe to it. A single OTM is deployed per publisher, so the publisher sends one stream to the SFU/MCU, reducing its bandwidth consumption.

The dOTM architecture is implemented by most current SFUs/MCUs. OTMs have been implemented under the

[1] https://ivs.rocks/

following names: Licode OneToMany,[2] Jitsi Videobridge,[3] Kurento DispatcherOneToMany,[4] MediaSoup Router,[5] etc.

Videoconference rooms (logical aggregations of streams from the same session) are divided into a series of OTMs as shown in Fig. 1. In this example, the tree participants act as publishers, and each one sends its media to an OTM. All participants subscribe to each other, so they receive all other media flows. These OTMs can be deployed on different machines or on the same one. However, this model is based on the assumption that a single OTM does not saturate the CPU of a single machine, because each publisher transmits media to only one OTM.

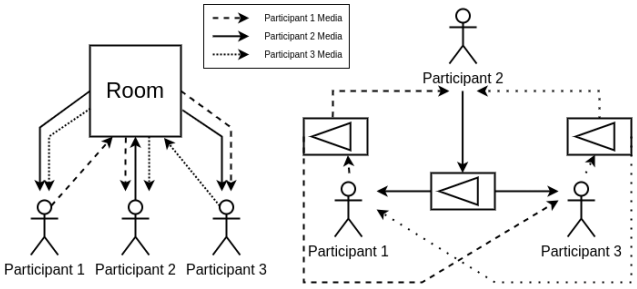


FIGURE 1. Room breakdown in OTMs.

The dOTM architecture presented in Fig. 2 is composed of the following components:

- OTM: The broadcaster that receives the publisher's media flow and distributes it to the other users. OTMs are the smallest indivisible units responsible for forwarding media.
- Agent: The component responsible for deploying and managing OTMs. Each machine used to deploy OTMs has an Agent that starts the corresponding processes and monitors both the OTMs and the machine.
- Controller: The component that manages rooms and exchanges messages between OTMs and clients. Most control logic is located in this component.
- All of these components intercommunicate with each other using a communication channel, such as a message queue.

The shortcomings of this architecture for massive online events are:

- As stated above, this architecture was not designed for massive online events. OTMs were designed as small independent units deployed on separate machines. In this scenario, each OTM's capacity becomes the limiting factor, as each publisher is assigned to a single OTM.
- The Controller can potentially be a bottleneck, even though the control layer handles a significantly lighter workload than the OTMs. As the number of participants increases, more messages must be exchanged between components, thereby increasing the Controller's load.
- Agents are not a concern for massive events. In fact, their workload is reduced because they have to deploy fewer but larger OTMs. Instead of hosting multiple OTMs per machine, each machine hosts an OTM that uses all available resources due to the higher number of subscribers.

[2]https://github.com/lynckia/licode/blob/master/erizo/src/erizo/OneToManyProcessor.cpp

[3]https://jitsi.org/jitsi-videobridge/

[4]https://doc-kurento.readthedocs.io/en/latest/_static/client-jsdoc/module-elements.DispatcherOneToMany.html

[5]https://mediasoup.org/documentation/v3/mediasoup/api/#Router

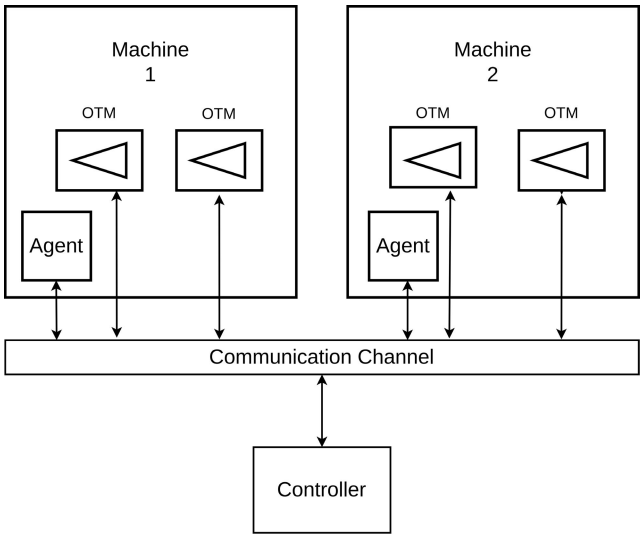


FIGURE 2. dOTM architecture.

The scenario described in Section II-A is a hybrid between streaming and videoconferencing, where the user scale resembles that of streaming, but the QoS requirements align with videoconferencing. Existing live streaming and videoconferencing solutions are not suited for massive online events; the former is due to the high delay, and the latter is due to the subscriber limitation. Protocols such as HLS [9] or MMPEG-DASH [10] are used in live streaming scenarios, but transcoding the incoming stream increases both delay and CPU usage. This delay complicates the interaction between participants. Therefore, videoconferencing technologies must be used in conjunction with live streaming architectures.

The effects of delay in videoconferencing and traditional telephone conversations have been studied extensively. To preserve interactivity, the maximum delay between the publisher and the subscriber must remain below a certain threshold. ITU [11] provides recommendations for total delay in voice communication, presented in Table 3. Other studies have confirmed a noticeable decline in interactivity and conversation quality when delays exceed 300 ms [12]. For this reason, the interactivity threshold is set at 300 ms.

In Section V, a tree based architecture is proposed to address the scalability limitations of OTMs, a tree-based infrastructure is proposed in Section IV. In this proposal, OTMs are capable of receiving and distributing media from other OTMs, thereby forming a cascading tree. Media streams are relayed along the OTM tree, thus overcoming the limitation of having only one OTM per publisher.

TABLE 3. ITU maximum delay recommendation.

Delay(ms)	User satisfaction
0-200	Very satisfied
200-280	Satisfied
280-390	Some users dissatisfied
390-520	Many users dissatisfied
>520	Nearly all users dissatisfied

III. RELATED WORK

Several videoconferencing providers have proposed architectures that resemble the tree-based model introduced in this study, however each has notable limitations.

A comparable tree architecture was implemented by Janus.[6] However, instead of relying on standard protocols such as WebRTC to interconnect OTMs, it uses a proprietary protocol to forward RTP (Real Time Protocol) packets. RTP forwarding is achieved using the Octopus protocol, a wrapper that adds an 8-byte header containing additional information such as conference and endpoint identifiers. Although this approach enables basic cascading, the system lacks adaptive bitrate control. Since pure RTP does not include bandwidth adaptation mechanisms, this approach is unsuitable for networks without guaranteed Quality of Service. Additionally, Janus does not provide any orchestration mechanism to automatically balance subscribers between publishers.

The Matrix[7] organization proposes a different model. Instead of a tree, it forms a mesh of interconnected SFUs, resembling the original P2P WebRTC topology. Media streams are exchanged between SFUs, and clients connect to the closest available endpoint. This scenario has the same limitations as the P2P one. In deployments with hundreds of machines, the number of connections increases exponentially. With a tree architecture, not all machines have to connect with each other, as each tree is independent. However, this model is of special interest when creating a geographically distributed network since every client connects to the closest endpoint, reducing latency.

Live streaming platforms also implement hierarchical media distribution mechanisms to scale to millions of simultaneous users. For instance, Twitch, the largest livestreaming platform, employs a directed graph hierarchy called a Replication Tree[8] to distribute media. However, Twitch reports a latency of approximately 1.5 seconds, even in low-latency mode, well above the interactivity threshold required for real-time communication.

Similarly, Amazon Web Services (AWS) provides a live streaming solution built on a Content Delivery Network (CDN). The stream cascaded from the streamer region to the rest of Amazon's Points of Presence (POP).[9] These connections utilize private infrastructure to minimize delay. Amazon reports that their infrastructure is capable of delivering media with a delay of less than 300 ms.[10] This solution is based on a proprietary PaaS that combines videoconference and streaming technologies. This service is similar to the one that is intended to be developed. However, the aim of this study is to provide an architecture and implementation based on standard protocols and open-source technologies. In addition, viewers do not have a return voice channel to interact with the publisher.

As can be observed, the cascading architecture used by the streaming services is capable of sending video in almost real time to millions of subscribers. Therefore, the proposed architecture is a combination of streaming architecture and videoconferencing technologies. Amazon is capable of achieving the sub 300 ms delay by using its private infrastructure and direct connections between POPs.

After reviewing existing videoconferencing solutions and streaming technologies, we have found that there is no existing solution to scale videoconferencing applications. Janus protocol is not based on standard protocols and does not provide an orchestration mechanism to connect users. The Matrix provides basic cascading by interconnecting SFUs and also does not provide and orchestration mechanism. Finally, AWS and Twitch are proprietary solutions based on proprietary implementations. For this reason, this article aims to provide both an scalable architecture based on open-source protocols and orchestration mechanisms to distribute users.

The technology selected for the implementation of the architecture is WebRTC. Over the last few years WebRTC has not only been used for traditional videoconference applications. The protocol was designed to maintain low delays under variable network conditions, so it has been used for Virtual Reality (VR) data transmission, telehealth, livestreaming, etc.

Because all the information in these scenarios is transmitted over WebRTC connections, they can also benefit from the proposed architecture. Importantly, the receiving client does not need to be a traditional videoconferencing application,it can be any software program capable of establishing a WebRTC connection.

IV. ARCHITECTURE PROPOSAL

A new architecture based on a tree structure is proposed to overcome the scalability limitations of the dOTMs model. This section introduces the architecture's components, algorithms, and bandwidth- control mechanisms.

A. REQUISITES

The proposed architecture must satisfy the following requirements:

- Horizontal scalability: It must support a larger number of subscribers per publisher than the dOTM model.
- Low latency: End-to-end delay must remain below the interactivity threshold.

[6]https://www.meetecho.com/blog/sfu-cascading/

[7]https://matrix.org/

[8]https://blog.twitch.tv/en/2023/09/28/twitch-state-of-engineering-2023/

[9]https://docs.aws.amazon.com/ivs/latest/LowLatencyUserGuide/security-resilience.html

[10]https://docs.aws.amazon.com/ivs/latest/LowLatencyUserGuide/what-is.html

- Video quality preservation: Subscribers must receive media with the same quality as in the dOTM scenario.
- Transparency: The changes introduced must be transparent to the subscriber application.
- Efficient resource usage: The architecture should optimize the use of system resources.
- Standards compliance: Standard protocols must be used to ensure interoperability.
- Rapid deployment: Components must be instantiated quickly to enable live interaction.

All of these criteria will be evaluated as part of the implementation assessment.

B. TREE ARCHITECTURE DEFINITION

This section defines the core components of the architecture and the connections between them.

A tree structure [13] is an acyclic graph in which two vertices are connected by exactly one path. Rooted trees (trees with a designated root) have been used as a data structure in computer science to hierarchically represent data.

In the proposed architecture, the OTM responsible for forwarding audio and video data from the publisher was replaced by a rooted tree of OTMs. The publisher serves as the root, and its media is cascaded through successive layers of OTMs.

The tree consists of OTMs connected in a hierarchical structure to transmit the publisher's media. In accordance with the dOTM architecture, each OTM can be deployed on the same machine or on separate machines. The tree, like a videoconference room, forms a logical grouping of OTMs.

The basic unit of the tree is the node, which is a combination of components that extract the data from an OTM and forward it to the clients or to a node where more clients can connect. A node is composed of the following:

- An internal subscriber that connects to an upstream OTM to receive media.
- An internal publisher that contains the OTM to which other nodes or clients connect.
- A connection module which transfers media between components.

The internal structure and data flow of a node are illustrated in Fig.3

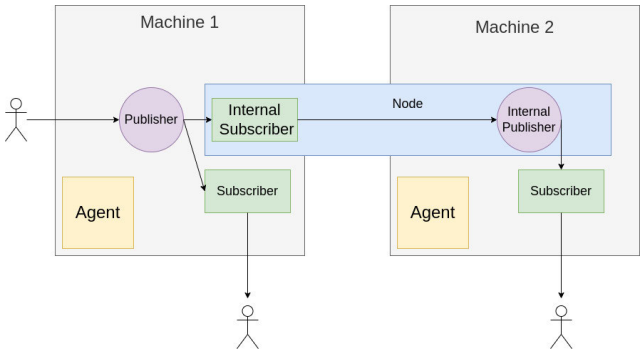


FIGURE 3. OTM interconnection.

The nodes are organized into levels, where each level consists of nodes equidistant from the root. The root node, where the publisher is connected, resides at level 0. Clients should be connected to the nodes from the lowest available level, as the end-to-end delay will be lower because of the lower number of jumps to the original source. The connection between nodes must be able to both transmit the data and the required signaling messages to establish the multimedia flow.

Despite their hierarchical structure, OTMs remain the fundamental unit of deployment, which is consistent with the dOTM philosophy. OTMs are self-contained and can be independently deployed.

C. TREE CONTROLLER

The logic for managing trees is encapsulated in the TreeController, a submodule of the main Controller. While the Controller continues to handle node creation, Agent selection, and client authentication, the TreeController manages tree-specific operations.

The TreeController maintains the following state information:

- Tree layout: Structure of the tree, and connection between nodes.
- Node status: Each node can be available or unavailable, depending on whether it can currently accept incoming connections. A node marked as unavailable can become available if the conditions allow for it.
- Client assignment: Tuple of client and node assigned, to forward the corresponding requests to the correct node.

This information enables the execution of the following algorithms, which are essential to efficiently manage the tree and incoming connections.

- Node selection: Allocates clients to the optimal available node.
- Node completeness: Identifies when a node has reached its capacity to prevent overload.
- Parent node selection: Chooses a parent node for a newly created node, aiming to minimize delay while ensuring connection diversity and fault tolerance.

In massive videoconferencing scenarios, OTMs must handle significantly more subscribers than in traditional settings. For this reason, providers should review their Agent assignment algorithms for OTMs. While in conventional conferences multiple OTMs may run on a single machine, here a single OTM should ideally utilize the full capacity of one machine. Deploying multiple OTMs on the same machine does not increase capacity, as they still compete for the same hardware resources.

D. CAPACITY OF THE TREE

Although the tree architecture can theoretically scale indefinitely, in practice it is constrained by the maximum acceptable end-to-end delay. This imposes a limit on the maximum number of levels in the tree. Clients connected to higher levels experience more hops and thus more delay.

The maximum number of subscribers M_{subs} the tree can support also depends on the fanout of each node, this is how many clients or other nodes it can serve. Since connecting to either a client or another node requires the same resources from the OTM's perspective, fanout can be used to either connect clients or other nodes. Depending on these values, the structure and the maximum number of participants will vary.

To calculate the capacity of the tree, the maximum acceptable delay D_{max} must first be set. This delay can vary depending on the application and should be approximately 300ms for videoconference. The delay of the clients connected to the last level of the tree (D_N) must be less than D_{max} as stated in (1).

$$D_N < D_{max} \tag{1}$$

D_N is calculated using (2):

$$D_N = T_{pub} + T_{sub} + \sum_{i=0}^{n} Pi + \sum_{i=0}^{n-1} Ti. \tag{2}$$

T_{pub} is the transmission time from the publisher to the tree root, T_{sub} is the transmission time from the last node to the client, T_i is the transmission time for each level, and P_i is the processing time for each node.

Once the maximum number of levels has been determined, the maximum capacity of the tree can be calculated.

The fanout of a node (F) can be used to connect with clients or other nodes. The sum of the maximum number of nodes subscribed (Ns) and the maximum number of clients subscribed (Cs) must be equal to F.

Both Nn and Cs should be configured by level. For example, setting $Nn > 0$ for the last level limits the number of subscribers, as no nodes will connect to that level. Assuming that all nodes have the same F because they are deployed on equivalent hardware, we can calculate the number of subscribers of a tree with n levels using (3) and (4):

$$F = Cs_i + Ns_i \forall i \in [0, n-1] \tag{3}$$

$$M_{Subs} = \sum_{i=1}^{n} (\prod_{j=0}^{i-1} Ns_j) \cdot Cs_i \tag{4}$$

To maximize the number of subscribers, the following values of Csi and Nsi are:

$$Cs_i = 0 \forall i \in [0, n-2] \tag{5}$$

$$Ns_i = 0 \forall i = n-1 \tag{6}$$

Under these conditions, clients are only connected to the nodes of the last level, and the fanout of the previous nodes is used to increase the tree width. Combining all equations, the maximum number of subscribers is given for a tree with n levels and nodes with F fanout.

$$M_{Subs} = F^{(n-1)} \tag{7}$$

There may be scenarios in which this distribution of subscribers is not optimal. There might be instances where, for example, a node of the first level of the tree is reserved to other speakers of the conference so that the active participants have the lowest possible delay.

Because most of the parameters listed depend on each implementation and deployment, in this section only theoretical calculations are presented to serve as a basis for each implementation.

Both the fanout and processing time of the nodes can be improved with better hardware. However, transmission delay is largely dependent on network routing, congestion, and geographic distance. For this reason, it is of special interest to deploy nodes as close to the end users as possible. Integration with Edge Computing resources and Software Defined Networks (SDN) [14], [15] can improve latency and reduce congestion in the core network.

E. SINGLE POINT OF FAILURE

In this architecture, the root node of each tree represents a single point of failure (SPoF) for that stream. This is inherited from the dOTM architecture, where in case of an OTM failure, no clients would be able to receive that stream. However, in this use case, publishers are expected to be critical users, such as conference speakers rather than just another participant. For this reason, this single point of failure is particularly important.

An intermediate solution between a pure P2P network and the dOTM model is to create n parallel trees. The publisher sends n copies of its media stream, each used to create a separate tree. Clients are distributed among these trees to mitigate the SPoF. In the event of an OTM failure, only 1/n of the clients are disconnected.

However, this solution is not suitable for our architecture because it shifts the SPoF from the OTM to the publisher's device Additionally, the publisher would consume n times more upload bandwidth, as it must send multiple copies of the stream. Although the publisher's device and its network connection also represent SPoFs, these are considered out of scope. Avoiding all SPoFs would require the publisher to use multiple devices connected to different networks, each sending media to a different tree.

V. IMPLEMENTATION

To validate the architecture, a prototype implementation has been developed on Licode,[11] an open-source MCU that provides full WebRTC compatibility and implements the discussed dOTM architecture. The prototype includes proposed versions of the node creation, parent node selection, node completeness, and agent assignment algorithms.

In the following sections, the details of the reference implementation are presented.

A. NODE INTERCONNECTION

Nodes exchange media by establishing a WebRTC connection. The ICE protocol (Interactive Connectivity Establishment) [16] is used to negotiate the connection parameters. Both endpoints have a full ICE implementation, exchanging

[11] https://lynckia.com/licode/

Session Description Protocol (SDP) messages to relay ICE candidates and capabilities. ICE-trickle is used to reduce the time required to establish the connection.

Multimedia data are transmitted through PeerConnection, a P2P WebRTC connection between the local and a remote peer using the full WebRTC stack. ICE messages are exchanged through the communication channel. The TreeController stores the parameters needed to exchange SDPs between peers. Licode uses the RabbitMQ[12] message broker as a communication channel. Licode is configured to only forward media, acting as an SFU, to better evaluate the cascading of the media.

B. AGENT ASSIGNATION ALGORITHM

Because OTMs are expected to become CPU-bound, new OTMs are deployed on empty machines, so OTMs maximize resource usage. This approach reduces the number of tree nodes, which results in a reduction in tree levels.

Algorithm 1 Agent Assignation Algorithm

```
agents = [agent1, agent2, ..., agentN]
for agent in agents do
    if agent.otms.lenght == 0 then
        return agent
    end if
end for
agent = startAndReturnAgent()
return agent;
```

C. NODE SELECTION AND NODE COMPLETENESS ALGORITHMS

1) CPU BASED

Following the work presented in [17], where CPU-based metrics are used in cloud scheduling, the initial node completeness algorithm was based on CPU usage.

Agents periodically send CPU usage reports. If a threshold is exceeded, all nodes on that agent are marked as unavailable for new connections, although they can still serve as parents for existing nodes.

The TreeController maintains a FIFO queue of subscription requests. It attempts to assign each request to the first available node at the lowest available tree level. If no such node is available, a new node is requested and created. While no nodes are available, the queue stops processing requests but continues to accept them.

However, this algorithm was discarded after the initial tests. The main issue found during the testing was that assigning a client to a node did not immediately increase the CPU usage of the node. Instead, there is a delay during connection negotiation, followed by a CPU spike that later stabilizes. This spike occurs during the creation of the connection, just after the negotiation phase.

[12]https://www.rabbitmq.com/

Algorithm 2 Client Balancing Algorithm

```
levels = [[node1, node2, ..., nodeN], ..., [node1, node2, ..., nodeN]]
for nodes in levels do
    for node in nodes do
        if node.isFull == false then
            return node
        end if
    end for
end for
node = createAndReturnNode()
return node;
```

When users connect in rapid succession, these spikes can overlap and saturate the node. The prediction of CPU impact and timing proved to be hardware dependent and too situational for reliable use, leading to the rejection of this method after testing.

2) TOKEN BUCKET

The second implementation uses a token bucket model to control user connection rates and avoid CPU overload. The token bucket algorithm was selected over other rate-limiting algorithms such as the leaky bucket or fixed window rate due to its capacity to absorb traffic bursts due to token accumulation. At the start of a conference, when the nodes are empty, this algorithm can conform the burst while preventing CPU saturation. In addition, connections that do not receive a token are placed in a queue instead of being discarded to avoid modifying the client and forcing it to submit multiple petitions.

Multiple OTMs are run in parallel, each with its own token bucket and client capacity limit. When a user requests a connection, they are assigned to an available OTM via round-robin. Clients waiting for a token still count toward that OTM's capacity to avoid queuing on overloaded nodes. After receiving a token, the client negotiates a connection with the assigned node.

When an OTM reaches the maximum number of subscribers, a new OTM is added to the tree to maintain the number of available nodes.

D. PARENT NODE SELECTION ALGORITHM

Each node has a maximum number of child nodes. New nodes are assigned to a parent in the previous level that is available using a round-robin strategy. Once all nodes at a given level have reached their child capacity, new nodes are added to the next level.

This method minimizes the number of hops between the publisher and the subscribers, thereby reducing the additional delay caused by cascading. It also ensures that users are evenly distributed among nodes. We selected the round-robin algorithm for parent selection because we found that CPU-based algorithms do not perform well due to the high variability of CPU usage.

E. NODE CREATION PROCESS

This section details the creation of the subcomponents of a node: the internal subscriber, internal publisher and the connection.

The creation of a node is a tree-step process:

- First, a parent node is selected using the parent selection algorithm.
- Then, an agent for the child node is selected.
- Finally, an internal subscriber is created in the parent, and an internal publisher is created in the child with the corresponding connection.

This process is illustrated in Fig. 4

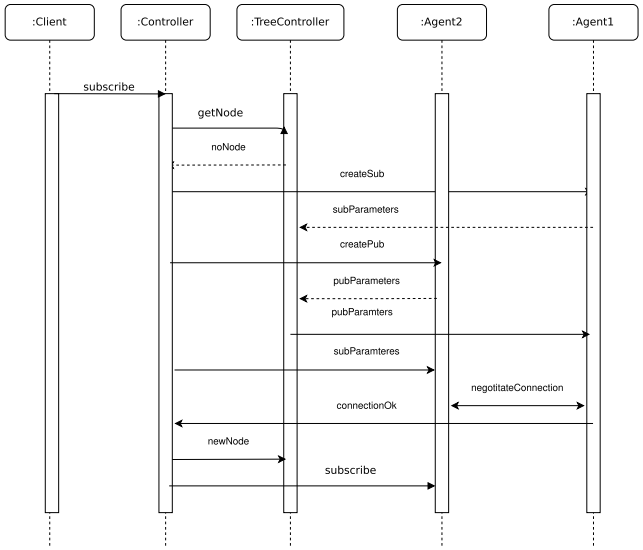


FIGURE 4. Node creation process.

The TreeController receives a series of parameters from both the publisher and the subscriber and stores them to forward ICE messages between them using the message queue.

This process is transparent to the client application. They request a subscription to the original stream, which is then routed to the corresponding OTM with the necessary changes in the parameters made by the Controller.

F. NODE FAILURE CONSIDERATIONS

A node is considered failed if its connection to its parent node fails or if multiple subscribers report a disconnection. In this case, all child nodes and subscribers are disconnected and the node is removed from the tree.

As WebRTC connections are P2P connections, clients must negotiate a new connection with a new endpoint. To achieve this, the client application must request a new subscription which will be handled by the node selection algorithm.

G. SECURITY CONSIDERATIONS

WebRTC was designed as a P2P protocol; therefore, the security architecture [18] and threat analysis [19] focused on this scenario. End-to-end (E2E) encryption is guaranteed by default in the pure P2P scenario.

However, this solution becomes a hop-by-hop (HBH) encryption when clients connect to an SFU. The SFUs break the encryption with the negotiated key, and then encrypt it again with the key exchanged with the subscriber. A solution to maintain E2E encryption is to create a ''double encryption'' as proposed by Janus [20] in their Privacy Enhanced RTP Conferencing architecture (PERC). The media is encrypted with a key that the SFU does not have access to. This solution limits some functionalities that require access to unencrypted media, such as recording, simulcast, or Scalable Video Coding (SVC).

In the tree architecture, HBH encryption is maintained if default WebRTC mechanisms are used. E2E encryption can also be achieved using the double encryption mechanism, as the key negotiation process is carried out outside the SFU / MCU. Using E2E encryption within the tree architecture also limits functionalities such as recording, simulcast, or using SVC as no node in the tree can access the raw media.

The WebRTC standard requires all connections to be performed over the Secure Real-Time Transport Protocol (SRTP), an extension of the RTP that encrypts the media. In a tree scenario, this requires encrypting and decrypting the media at every step with the corresponding overhead. To reduce CPU load, encryption can be disabled between nodes if they are all located inside a secure network. Removing this encryption implies that an attacker that has access to network that interconnects the nodes would be able to access the raw media stream, without needing access to the machines where the nodes are hosted. In distributed scenarios, where nodes connect over public networks, encryption between nodes is required by the standard.

In our experiments, encryption between nodes was disabled due to the use of a private network. Encryption between the publisher and the SFU and between the SFU and the subscriber are always be enabled.

H. BANDWIDTH CONTROL CONSIDERATIONS

Videoconference providers use algorithms such as Scalable Video Coding (SVC) [21] or Simulcast [22] to adapt the bandwidth used by clients.

SVC splits the video data into spatial and temporal layers. However, this does not work well in tree architectures: once a node receives only a subset of layers, all its children are limited to those layers.

For this reason, Simulcast was selected over SVC for bandwidth adaptation. Simulcast embeds multiple spatial and temporal layers into a single stream that can be extracted depending on the client's bandwidth. Because of this, nodes in the tree can forward the full stream with all spatial and temporal layers, and the last node can forward only the necessary layer to the clients.

VI. ARCHITECTURE VALIDATION AND RESULTS

This section presents the test performed on the reference implementation and key findings.

A. TEST SCENARIO

To validate the proposed method, two experiments were conducted in the described scenario. First, the current Licode implementation (dOTM architecture) is used to corroborate the limitations of the architecture. Subsequently, the prototype implementation of the tree architecture is tested to assess its viability across various performance parameters.

The testbed consisted of six machines, and the hardware is presented in Table 4. One machine deploys a Controller, the communication channel, the software used to monitor resources, and the logic related with client authentication. Other four machines deploy Agents, whereas the remaining one is used for generating client's connections.

B. CLIENT SIMULATION AND MONITORING

Licode includes a module called Ackuaria,[13] which collects events such as node creation, client subscriptions, Agent CPU reports, etc., and stores these data in a MongoDB[14] instance.

To simulate client connections, Selenium[15] was used to launch headless (without GUI) instances of the Chrome browser. These browsers are executed by machines different from those hosting the Agents, therefore, the CPU usage statistics are not altered.

C. DOTM ARCHITECTURE RESULTS

In the dOTM architecture, when the publisher connects, an OTM is created on one of the available machines, and all subscribers are added to it. The CPU usage over time of the machine where the node is created is presented in Fig. 5.

In this experiment, 500 clients were connected to the application over a period of 100 seconds. With the available hardware, 240 clients were able to connect to the OTM, and of these, only 190 received any media data before CPU saturation. At approximately 190 subscribers, the CPU reached 100% utilization, and the OTM's performance begins to degrade.

After a few seconds, the OTM crashed and all clients were disconnected. Despite the availability of the other three machines, Licode does not use them. This is because the OTM is the smallest indivisible unit of the architecture; and therefore Licode cannot distribute the load. The remaining machines remained idle because no OTM was assigned to them.

D. TREE ARCHITECTURE RESULTS

The objective of this experiment was to validate the proposed architecture by increasing the maximum number of subscribers for a single publisher. As there are four available Agents, the TreeController will deploy a node on each machine. The first node connects with the publisher, and the three reaming ones connect to the first node, forming the tree.

[13]https://github.com/lynckia/ackuaria
[14]https://www.mongodb.com/
[15]https://www.selenium.dev/

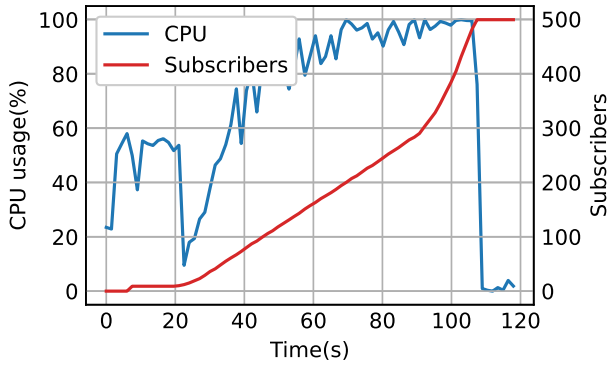


FIGURE 5. Failed OTM due to machine CPU overload.

The first test evaluated the CPU-based algorithm for node assignment. The same experiment was repeated using the tree architecture implemented in Licode. As the implementation is transparent to the application, no changes were necessary to the client simulator.

With this new architecture, clients are distributed among the different nodes, allowing them all to connect. The nodes were created in less than 500 ms as shown in Fig. 6 so they can be rapidly deployed on demand.

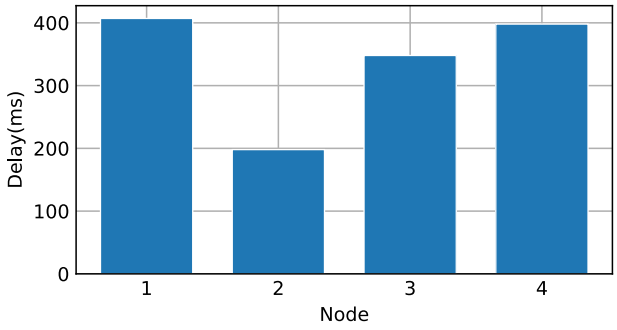


FIGURE 6. Node creation delay.

However, as previously noted, the use of a CPU-based algorithm to control node availability leads to node saturation. This is because the algorithm does not mark the nodes as unavailable until the connection is fully established. In this setup, nodes are marked as unavailable when CPU usage exceeds 60%. Fig. 7 shows the CPU usage of the Agent where node 1 is connected and the number of clients connected to that node. The CPU usage increased significantly after accepting new clients and reached 100% usage even with this security margin. Additionally, during the experiment, clients arrived at a constant rate; in the event of a sudden burst, all clients could potentially be assigned to a single node.

To address this issue, a token bucket-based algorithm was introduced to control the client admission into nodes. To stress this solution, all clients will request the subscription at the same time. This simulates the start of a conference or a user wanting to ask a question that must be listened to by the rest of clients. However, due to hardware limitations

TABLE 4. Hardware specifications.

Role	Nº Machines	CPU	RAM	DISK	OS
Controller	1	Intel Xeon CPU X3320 @ 2.50GHz	8GB	SSD 250GB	Ubuntu 20.04 Server
Agent	4	Intel Xeon CPU X3320 @ 2.50GHz	8GB	SSD 250GB	Ubuntu 20.04 Server
Clients	1	Intel® Core™ i9-12900 × 24	24 GB	SSD 2TB	Ubuntu 20.04.6 LTS

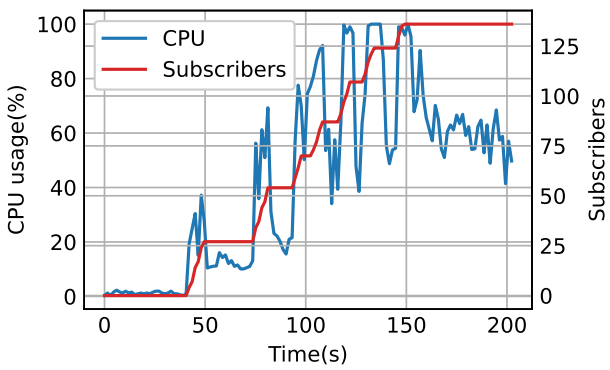


FIGURE 7. CPU spikes caused by establishing connections.

only 310 clients could be simulated simultaneously (creating the WebRTC connection with the application requires more resources than maintaining it).

Once the publisher connects, three nodes are created, one in each available machine. The token bucket of each node conforms the clients connection at a rate that the nodes can support without overloading. Fig. 8 shows the differences in CPU usage using the token bucket algorithm. The CPU spikes are mitigated using the token bucket, and the clients are evenly split among nodes. CPU usage reached a maximum of 60% in the nodes.

However, with the available hardware, the time spent reached 50 seconds, because clients must wait for a token to connect, instead of being directly accepted. This time can be reduced by pre-provisioning more nodes or by using higher-end machines with an increased rate of token generation.

Once all clients are connected, the experiment ends and all clients are disconnected.

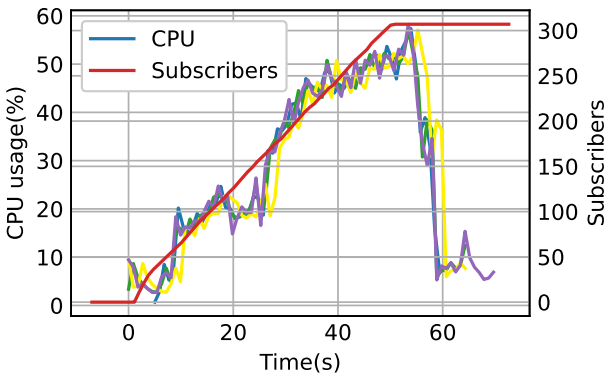


FIGURE 8. Token bucket conforming traffic.

The delay reported for the connection between nodes is shown in Fig. 9. Because the Agents are connected through the same switch, transmission delays are less than 10 ms and allow for the creation of multi-level trees to be created. As stated in 2 the transmission delay is only a part of the total end-to-end delay and the processing time required by the nodes should also be considered when deploying the infrastructure. In a real-world scenario, the nodes can be geographically distributed to reduce latency instead of all being located in the same datacenter. However, previous experiments show that it is not a trivial task and can have a negative effect on user experience [23] depending on the conference setup.

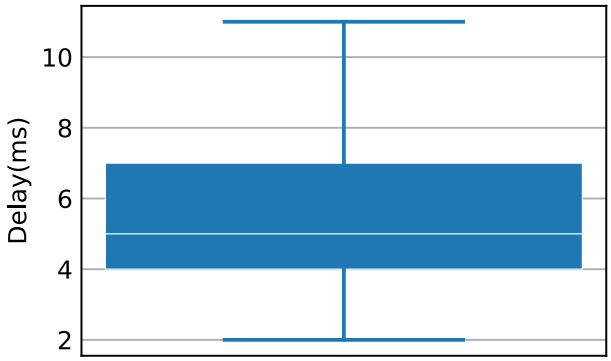


FIGURE 9. Node transmission delay.

VII. CONCLUSION AND FUTURE WORK

In this study, a tree-based architecture is proposed to improve the scalability of the dOTM architecture. Its objective is to increase the maximum number of subscribers per publisher while maintaining the quality and end-to-end delay. As the architecture is transparent to the client application, this solution can also be used to broadcast data for other applications such as VR. Finally, we implemented and tested a reference architecture to validate the proposal and identify possible pitfalls when implementing a production-ready or enterprise-grade system.

Despite the limited test-bed due to hardware limitations, based on the results obtained in Section VI, it can be concluded that the presented architecture is viable and scales over the limits imposed by the current distributed OTM solution. The architecture is able distribute a single publisher between multiple machines without overloading them. Experiments were performed for hundreds of users, but they can be scaled up to thousands of participants using both high end machines and a higher number of them.

For future research work, the areas that require further work are as follows:

- The main factor limiting the scalability of the tree architecture is the maximum tolerated delay. To reduce the latency and create trees with more levels, the deployment of the infrastructure in Edge Computing resources should be studied. However, studies have shown that this is not a trivial task and can have a negative effect on user experience [23]. In that study, even though the path that the media followed between clients was optimized, it did not result in a reduction in RTT between clients.
- The deployment of the architecture over cloud infrastructure should be researched as videoconferencing applications are not easily integrated over the commonly used orchestrators such as Kubernetes. There are projects such as STUNner that solve most common limitations such as Network Address Translation (NAT) traversal and scaling down nodes without terminating existing connections, so the implementation of this architecture over these projects should be studied.
- Finally, further research regarding the parent node selection algorithm and the node assignation algorithm should be performed to develop better algorithms based on parameters such as node location, CPU usage, user location, etc.

REFERENCES

[1] G. Richards, "Physical and digital events: Virtually the same? An examination of the digital pivot in events," Tilburg Univ., The Netherlands, Tech. Rep., May 2022.

[2] M. H. Willebeek-Lemair and Z.-Y. Shae, "Centralized versus distributed schemes for videoconferencing," in *Proc. 5th IEEE Comput. Soc. Workshop Future Trends Distrib. Comput. Syst.*, Apr. 1995, pp. 85–93.

[3] J.-I. Bruaroey, H. Boström, C. Jennings, and F. Castelli. (Mar. 2023). *WebRTC: Real-Time Communication in Browsers*. [Online]. Available: https://www.w3.org/TR/2023/REC-webrtc-20230306/

[4] M. H. Willebeek-Lemair, D. D. Kandlur, and Z.-Y. Shae, "On multipoint control units for videoconferencing," in *Proc. 19th Conf. Local Comput. Netw.*, 1994, pp. 356–364.

[5] P. Rodríguez, Á. Alonso, J. Salvachúa, and J. Cerviño, "Materialising a new architecture for a distributed MCU in the cloud," *Comput. Standards Interfaces*, vol. 44, pp. 234–242, Feb. 2016.

[6] H. Mahmoud and R. Abozariba, "A systematic review on WebRTC for potential applications and challenges beyond audio video streaming," *Multimedia Tools Appl.*, vol. 84, no. 6, pp. 2909–2946, Nov. 2024, doi: 10.1007/s11042-024-20448-9.

[7] P. Rodríguez, A. Alonso, J. Salvachúa, and J. Cerviño, "DOTM: A mechanism for distributing centralized multi—Party video conferencing in the cloud," in *Proc. Int. Conf. Future Internet Things Cloud*, Apr. 2014, pp. 61–67.

[8] H. Kharouf, R. Biscaia, A. Garcia-Perez, and E. Hickman, "Understanding online event experience: The importance of communication, engagement and interaction," *J. Bus. Res.*, vol. 121, pp. 735–746, Dec. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0148296319308033

[9] R. Pantos and W. May, *HTTP Live Streaming*, document RFC 8216, Aug. 2017.

[10] *Dynamic Adaptive Streaming Over HTTP (DASH) Part 1: Media Presentation Description and Segment Formats*, ISO Central Secretary, Geneva, Switzerland, Aug. 2022.

[11] I. T. Union. (May 2003). *G.114: One-Way Transmission Time*. [Online]. Available: https://www.itu.int/rec/T-REC-G.114

[12] F. Hammer, P. Reichl, and A. Raake, "Elements of interactivity in telephone conversations," in *Proc. 8th Int. Conf. Spoken Language Process. (ICSLP)*, Jeju Island, South Korea, Oct. 2004.

[13] J. Morris. (2023). *Combinatorics: Enumeration, Graph Theory, Design Theory*. [Online]. Available: https://openlibrary-repo.ecampusontario.ca/jspui/handle/123456789/481

[14] C. A. Hasrouty, M. L. Lamali, D. Magoni, and J. Murphy, "SDN-enabled adaptation of videoconference streams to network dynamics," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, vol. 2, Jan. 2017, pp. 1–6.

[15] A. Wang, Z. Zha, Y. Guo, and S. Chen, "Software-defined networking enhanced edge computing: A network-centric survey," *Proc. IEEE*, vol. 107, no. 8, pp. 1500–1519, Aug. 2019.

[16] J. Rosenberg, *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*, document RFC 5245, Apr. 2010.

[17] Á. Alonso, I. Aguado, J. Salvacúa, and P. Rodríguez, "A methodology for designing and evaluating cloud scheduling strategies in distributed videoconferencing systems," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2282–2292, Oct. 2017.

[18] E. Rescorla, *WebRTC Security Architecture*, document RFC 8827, Jan. 2021.

[19] E. Rescorla, *Security Considerations for WebRTC*, document RFC 8826, Jan. 2021.

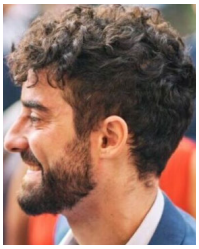
[20] A. Amirante, T. Castaldi, A. Gouaillard, L. Miniero, S. G. Murillo, and S. P. Romano, "Bringing privacy to the Janus WebRTC server: The PERC way," in *Proc. Princ., Syst. Appl. IP Telecommun. (IPTComm)*, Sep. 2017, pp. 1–8.

[21] B. Aboba. (Nov. 2023). *Scalable Video Coding (SVC) Extension for WebRTC*. [Online]. Available: https://www.w3.org/TR/2023/WD-webrtc-svc-20231115/

[22] B. Burman, M. Westerlund, S. Nandakumar, and M. Zanaty, *Using Simulcast in Session Description Protocol (SDP) and RTP Sessions*, document RFC 8853, Jan. 2021.

[23] B. Grozev, G. Politis, E. Ivov, and T. Noel, "Considerations for deploying a geographically distributed video conferencing system," in *Proc. IEEE 8th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2018, pp. 357–361.

CARLOS ARRIAGA received the master's degree in engineering from the Universidad Politécnica de Madrid, Madrid, Spain, where he is currently pursuing the Ph.D. degree. His research interests include videoconference and artificial intelligence.



ALEJANDRO POZO received the Ph.D. degree in engineering from the Universidad Politécnica de Madrid, Madrid, Spain. He is currently an Assistant Professor with the Universidad Politécnica de Madrid. His research interests include artificial intelligence, data engineering, and cybersecurity.

ALVARO ALONSO received the Ph.D. degree in engineering from the Universidad Politécnica de Madrid, Madrid, Spain. He is currently a Professor with the Universidad Politécnica de Madrid. His research interests include multi-conferencing systems in cloud computing and artificial intelligence.