AsTree: An Audio Subscription Architecture Enabling Massive-Scale Multi-Party Conferencing

Tong Meng, Wenfeng Li, Chao Yuan, Changqing Yan, and Le Zhang, *ByteDance Inc.*

https://www.usenix.org/conference/nsdi25/presentation/meng

This paper is included in the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation.

April 28–30, 2025 • Philadelphia, PA, USA

978-1-939133-46-5

Open access to the Proceedings of the 22nd USENIX Symposium on Networked Systems Design and Implementation is sponsored by

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

AsTree: An Audio Subscription Architecture Enabling Massive-Scale Multi-Party Conferencing

Operational Systems Track

Tong Meng Wenfeng Li Chao Yuan Changqing Yan Le Zhang
ByteDance Inc.

Abstract

While operating a multi-party video conferencing system (Lark) globally, we find that audio subscription alone may pose considerable challenges to the network, especially when scaling towards massive scales. Traditional strategy of subscribing to all remote participants suffers from issues such as signaling storm, excessive bandwidth and resource consumption on both server and client sides. Aimed at enhanced scalability, we share our design of AsTree, an audio subscription architecture. By a cascading tree topology and media plane-based audio selection, AsTree dramatically reduces the number of signaling messages and audio streams to forward. Practical deployment in Lark reduces audio and video stall ratios by more than 30% and 50%. We also receive 40% less negative client reviews, strongly proving the value of AsTree.

1 Introduction

Real-time multi-party video conferencing has been ubiquitous in our daily lives. Lark [1], as one of a rich set of heavily used services and products (*e.g.*, Zoom [2], Microsoft Teams [3], DingTalk [4]), serves tens of millions of users, and benefits vast many industries such as online education, international business, interactive social networking, *etc.*

Rather than focusing on simulcast-based video subscription like many existing literature [5, 6], we report our experiences of designing and deploying AsTree in Lark, an audio subscription architecture targeted at massive-scale conferences (*e.g.*, scaling up to hundreds and thousands of participants per room). Notably, there are several key differences between audio and video subscription, posing challenges to the architecture for the former correspondingly.

Whether to subscribe to a stream. Subscription and unsubscription to video streams can be the outcome of a subscriber's proactive operations, *e.g.*, when the subscriber scrolls through the participant list, or pins/unpins a specific publisher. In comparison, a subscriber mostly needs to be passively subscribed to an audio stream, as long as its publisher is actively speaking. That makes the audio subscription relations between conference participants unpredictable, because we cannot deterministically predict when and which participant will speak.

When to subscribe to a stream. Subscribing to a video stream requires 4 steps: (1) a publisher broadcasts a Publish signaling message when turning on camera, (2) a subscriber sends an Subscribe message (not necessarily closely follow previous step), (3) an overlay cascading path is established between the publisher and subscriber, (4) the video stream is cascaded and distributed. For an audio stream, however, the above steps cannot wait until a participant becomes unmuted. Considering that audio is generally more latency-sensitive than video, the signaling round trip and on-demand establishment of cascading path induce unwanted delay. What is worse, there would be loss of audible information if a participant starts talking right after clicking the unmute button.

How many streams to subscribe. Even if all participants open their cameras, each one subscribes to a bounded number of video streams, depending on the user interface layout. Yet when it comes to audio subscription, the number of audio streams for subscription may increase indefinitely, since every participant in a room may unmute. At a massive conference scale, subscribing to all unmuted participants may lead to considerable signaling overheads and bandwidth consumption.

In the early stage of our deployment, massive conferences where all participants are allowed to unmute themselves were uncommon. So we assumed audio consumes much lower bandwidth than video at that time, and employed a straightforward "subscribe-to-all" strategy for audio subscription (denoted as FullAud henceforth). To improve timeliness of audio signal, subscription to a participant is aggressively initiated when they joins a conference room, without waiting for them to unmute. Since most participants start muted by default, we expect the signaling round trip and cascading path establishment to finish before a participant actually starts talking. Along with that, each participant keeps publishing an audio stream while in the conference. When the microphone is muted, the audio stream simply contains a few Discontinuous Transmission (DTX) frames [7]. Then, the signaling message emitted from a participant clicking the mute/unmute button is used to toggle the displayed audio status seen by other participants. As far as we know, at least until the pandemic, many products also default to full audio subscription [8–10], though their realizations may differ.

Not surprisingly, as our DAU (daily active users) continuously grows and massive-scale conferences become increasingly frequent, FullAud starts to face scalability issues.

- **Signaling storm.** Influx of participants often happen at the start of a conference, inducing surging signaling messages for audio subscription. As a result, both audio and video are more likely to suffer from network and host congestion. Besides, the signaling broadcast generated every time a

participant mutes/unmutes the microphone interferes with cascading and distribution of media data to some extent.

- **Client-side resource overheads.** When a single participant has to fetch tens or hundreds of audio streams, the last-mile access link can be easily overwhelmed, increasing chances of choppy audio and video rebuffering. Meanwhile, CPU, memory, and battery power on user devices consistently operate under excessive loads, and thus, tend to cause performance issues such as overheating that eventually impair audio/video quality, as well.
- **Server-side resource overheads.** A room with N participants distributes $O(N^2)$ downlink audio streams at the edge. The quadratic scaling egress bandwidth makes FullAud substantially less affordable. In addition, the ever-expanding conference scale urges prohibitive investment in WAN bandwidth and media servers, considering that (1) individual media servers are more likely to be overloaded in densely populated regions, and (2) massive participants are more geographically dispersed and cascade more server clusters.

The AsTree architecture is designed to overcome the above issues. It is inspired by an empirical observation: at any time during a multi-party conference, almost all critical information comes from a small number of audio streams [10, 11]. Therefore, a majority of audio streams can be ignored from subscription. To effectively carry out that idea in a practical system, we should fulfill the following objectives.

- **Light-weight signaling.** For agile conference management, the selection of audio streams and notification to participants should not only avoid adding new signaling messages, but also significantly alleviate the existing signaling storm.
- **No single point of failure.** Audio stream selection needs to compare attributes of all audio streams. Gathering streams to a single server for that purpose should be avoided. Otherwise, that server becomes the computing and bandwidth bottleneck, whose failure impacts all participants in a room.
- **Compatibility with simulcast.** We want the new design for audio subscription to induce minimal changes to the already mature video simulcast architecture, and enable independent engineering development and modular upgrade.

As a large-scale system in a well-explored field, AsTree shares similarities with some existing works (*e.g.*, separation of media and control planes [5], cascaded media servers [12, 13]). Importantly, many conferencing service providers disclose limited information on how they scale the audio subscription architecture. To the best of our knowledge, we are the first to provide comprehensive experiences on that. Our main contributions are highlighted below.

- AsTree constructs a cascading tree between media servers for each conference room to lower WAN bandwidth consumption. It assigns a delegate server in each region connecting other intra-region media servers, and calculates an inter-region tree cascading region delegates.
- AsTree selects a subset of audio streams composed of dominant speakers at each media server, without control plane involvement. Effectively, a subscriber only subscribes to those audio streams containing valid vocal signal. Moreover, it eliminates the signaling broadcast when a participant mutes/unmutes the microphone.
- We present extensive results from benchmarking tests and practical deployment. AsTree is shown to significantly outperform FullAud. To be specific, negative reviews are reduced by 50%, audio stall is reduced by 30%, and video stall is reduced by 50%. Also, a media server can hold conferences with an order of magnitude more participants.

This work does not raise any ethical issues.

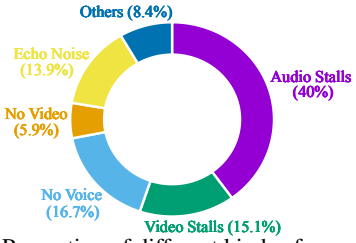


Figure 1: Proportion of different kinds of user complaints

2 Motivation

Thanks to the deployment of simulcast, Lark's video quality of experience (QoE) has been notably improved, making audio stall the most common category among millions of our user complaints (as in Figure 1, collected by post-call questionnaire with non-exclusive options). That confirms the importance of evolving the audio subscription architecture. To fully motivate AsTree and gain some insights, we explain the scalability issues of our early-stage FullAud implementation in more detail, and analyze the drawbacks of several strawman solutions.

2.1 Scalability Issues of FullAud

2.1.1 Local Signaling Overheads

A centralized signaling unit is both the gathering point of signaling reports and originating source of signaling broadcasts, *e.g.*, so as to maintain a full picture of the conference room and direct the establishment of cascading links, respectively. To alleviate its pressure, we deploy local signaling units co-located with media servers, and aggregate signaling messages between centralized and local signaling units into $O(1)$ per broadcast. However, each local signaling unit generates $O(N)$ messages while signaling its connected participants.

As an example, Figure 2 illustrates the signaling round trip for audio subscription. FullAud amortizes the signaling delay to the moment a participant joins a conference, and thus, merges the broadcast of Join and Publish. It also triggers a consequent ingestion of $O(N)$ Subscribe messages. Additionally, signaling broadcasts also happen when participants (1) click the mute/unmute button and (2) leave the room.

If combining all participants in a room, the volume of signaling messages borne by local signaling units is as high as

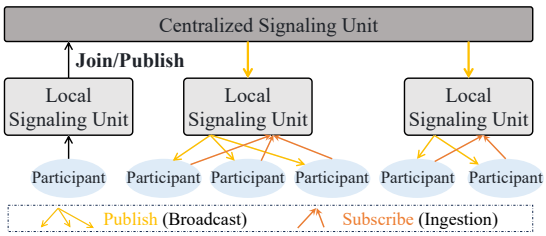


Figure 2: Signaling round trip for audio subscription in FullAud

Video Stream	Focus Window	Thumbnail
	700 Kbps	120 Kbps
Audio Stream	Unmuted	Muted (DTX only)
	64 Kbps	1 Kbps

Table 1: Approximate bandwidth requirements of video and audio streams (video based on Lark's mobile app)

$O(N^2)$. Although each message does not exceed several tens of bytes, massive-scale conferences may still suffer from signaling storm and degraded user experience, especially when:

- A large number of participants join within a short time period, which is common at the starting phase of a conference.
- Highly concurrent conferences during peak hours already leave a high CPU usage on the edge nodes.
- Participants repeatedly unmute/mute for different purposes, including playing with the button aimlessly.

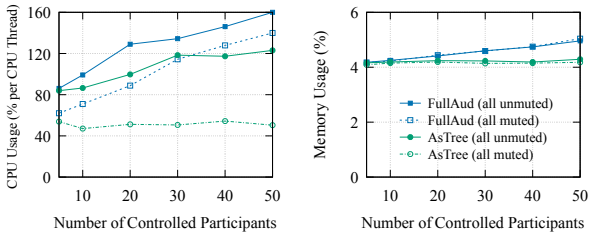
As will be demonstrated in §6.1, the signaling storm may consume even higher CPU than media processing. Regardless of the frequency of signaling storm, the possibility of its occurrence prevents us from scheduling many participants to the same media server. That limits the maximum number of participants a single media server can accommodate per room under FullAud, leading to inefficient utilization of server machine resources.

2.1.2 Client-Side Overheads

FullAud's client-side bandwidth consumption increases linearly with the room size. To intuitively compare it with video subscription, let us consider a simple scenario: a participant joins a conference using Lark Mobile on a smartphone. Assume speaker layout is used, with a 720p full-screen focus video and a 90p floating thumbnail. Referring to [14] and our internal configurations, bandwidth requirements of individual video and audio streams are approximated as in Table 1.

Each participant needs 0.82 Mbps to fetch the focus and thumbnail videos, regardless of the room size. When there are more than 12 unmuted participants at the same time, audio will consume more bandwidth than video. Even if all participants are muted, their DTX-based audio streams will require the same order of magnitude bandwidth as video with more than 100 participants. We should note that this is a conservative estimation, since we do not consider possible reduction in video resolution to resist weak networks, and needs for higher-quality audio for increased immersion.

In addition, we leverage the stress test system at Lark to demonstrate FullAud's hardware loads on user devices. We use a Redmi Note 13 smartphone to join setup conferences with different number of controlled participants, and monitor its CPU/memory usage (Figure 3). All controlled participants do not publish video streams to isolate the overheads of audio subscription. We test two extreme cases where controlled participants are either all unmuted or all muted.



(a) CPU Overheads **(b)** Memory Overheads

Figure 3: Client-side overheads induced by FullAud

As expected, FullAud's resource usage scales nearly linearly with the number of participants. Because the smartphone has to receive and process muted audio streams, when the number of muted controlled participants increases from 5 to 50, it occupies more than half an additional CPU core. It has almost the same memory usage in both cases. That means some low-end smartphones may not even have enough capabilities to join a conference with hundreds of participants. For comparison, we also depict AsTree in Figure 3. In AsTree, the smartphone subscribes to a limited number of audio streams, excluding any muted streams. Thus, its resource usage remains relatively stable with different number of muted participants. When there are 50 muted participants, AsTree's CPU and memory usage are 64% and 17% lower than FullAud, respectively. We note that all controlled participants use the same audio track, causing the selected dominant speakers to change frequently. That explains the increasing CPU usage of AsTree when there are more unmuted participants.

2.1.3 Server-Side Overheads

FullAud effectively creates an overlay mesh cascading every pair of media servers. As a result, a room that involves N participants and M media servers consumes $O(MN)$ and $O(N^2)$ bandwidth on inter-server WAN and egress links, respectively.

More importantly, a larger conference scale alone can raise server-side expenses, even with the same user population. Assume a media server instance has the CPU, memory and associated egress link bandwidth that are just enough to distribute 100 audio streams. Then, one such instance can hold 50 concurrent 2-party audio-only conferences, but we need 100 instances to have a single 100-party conference (*e.g.*, each instance serves a single participant). Moreover, the overall egress bandwidth consumption is 99×, and the same amount of WAN bandwidth is needed for cascading. Although ignoring the influence of many practical factors, such a comparison illustrates the possibly forbidding costs while scaling FullAud. Similar issue is mitigated by simulcast in video subscription, where the number of video streams distributed to an individual participant is capped by the user interface.

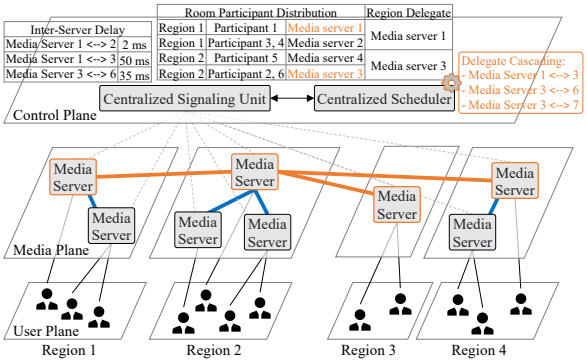


Figure 4: Overview of the AsTree architecture (local signaling units co-located with media servers are omitted)

2.2 Strawman Solution Analysis

Several strawman solutions can reduce the number of audio streams subscribed by each participant. Though seemingly simple, they come with various drawbacks.

(1) Restricted unmute privilege. The most straightforward way to enforce subscription to fewer audio streams is to set a hard limit on the maximum number of concurrent unmuted participants. Many online webinars adopt that solution. Therein, an audience member should request to unmute from a host, and be muted after finishing. That way, subscribing to all active speakers does not generate high overheads.

Drawbacks: Having to apply and compete for opportunities to speak induces inconvenient manual operations. One such interaction easily takes several seconds. Thus, the solution has a much narrower scope other than lecture-style conferences. It is not applicable in emerging scenarios requiring strong interactivity, such as online social and virtual karaoke.

(2) Signaling-based audio selection. While publishing to the local media server,[1] a participant also signals the attributes of their audio stream to the centralized signaling unit. Based on the collected audio attributes, the signaling unit leverages a controller to select a subset of audio streams to be forwarded. Each media server waits for instructions from the signaling unit to determine which ingested audio streams to cascade.

Drawbacks: This solution introduces new signaling messages for audio stream selection, which should be sent frequently to reflect dynamics in audio attributes (*e.g.*, fluctuating voice volume, *etc.*). That exacerbates the signaling storm issue. What is worse, user experience is more vulnerable due to the additional signaling round trip for a selected audio stream.

(3) Select before distribute A media server still pulls one audio stream from each participant, whether they are unmuted or not. Whereas in the downlink direction, a selection logic is added to each media server, such that only selective audio streams are distributed. Besides that, the publishing and cascading of audio streams are kept the same as in FullAud.

Drawbacks: Compared with the previous two solutions, this solution puts no limit to applicable use cases, and avoids complicated signaling interactions for audio selection. Yet it is still based on full audio subscription, and media servers are still cascaded by a mesh. An audio stream, whether selected or not, needs to be received and processed by all media servers involved in a conference. That provides limited help to control server-side costs on WAN bandwidth and media servers. In fact, that is an important reason why Jitsi had to temporarily disable mesh cascading during the COVID-19 pandemic [15].

[1] A local media server specific to a participant is the server to which the participant is directly connected. A local media server specific to an audio stream is the local media server of its publisher.

3 AsTree Design Overview

Figure 4 presents the high-level architecture of AsTree. Similar to [5], it is composed of user plane, media plane, and control plane. The user plane and media plane are divided into separate geographical regions, and the control plane is logically centralized.

AsTree is based on selective forwarding units (SFUs), which has been the dominating choice by modern multi-party conferencing providers [5, 6, 16, 17]. Participants connect to nearby local SFU media servers. Cascaded SFUs form overlay paths between participants, and forward media streams without decoding them. Furthermore, one or more SFUs involved in a conference room form an audio selection tree (hence the acronym AsTree[2], details in §4) as opposed to a mesh. Each region has a single media server as the region delegate. The other media servers are cascaded to their intra-region delegates. Then, all the delegate servers are connected by a spanning tree encompassing the corresponding regions. Such an AsTree topology is calculated by a centralized scheduler in the control plane. For that purpose, it synthesizes distribution of participants in the same room and latency between media servers, which are collected by the signaling unit.

Beyond that, the control plane does not participate in selection of audio streams. Instead, audio selection is conducted hop by hop at the media plane, *i.e.*, each media server only selects at most L loudest active speakers (details in §5). All participants, including those muted ones, still keep publishing audio streams to their local media servers, but muted audio streams will be excluded from audio selection. Thus, the number of audio streams forwarded on any cascading link and distributed to any participant is capped by a constant parameter. Moreover, no signaling broadcast is needed when a participant mutes/unmutes the microphone. The audio status of each participant displayed on user interface directly corresponds to the results of audio selection, *i.e.*, only those whose audio streams are selected are displayed as unmuted.

4 Calculation of Cascading Tree

In Lark's deployment, the division of regions is fairly static, depending on geo-planning of server clusters. The calculation of the AsTree topology focuses on the two hierarchies as outlined in §3, *i.e.*, intra-region and inter-region cascading.

[2] AsTree may stand for the audio subscription architecture or the cascading tree topology. The context where it is used can help avoid ambiguity.

4.1 Intra-Region Cascading

The core of intra-region cascading is the selection of region delegate. To lower the implementation complexity, we do not adopt complicated leader election algorithms (*e.g.*, [18, 19]) from the field of distributed systems. Instead, we take a "first comer elected" method. When a media server is the first in its region to join a conference room, it is configured to be the region delegate. Subsequent media servers joining the same room in the same region are cascaded to that delegate.

The above simple method is out of three considerations. First, the one-way latency between media servers in the same region is usually much lower than the industrial target for acceptable one-way latency (*e.g.*, above 200 ms between individual participants [20–22]). Changing the elected region delegate delivers similar audio QoE in most cases. Second, the interaction delay between intra-region participants is kept minimal, because their audio streams reach each other without leaving the region. Third, since each media server only forwards at most L audio streams, the region delegate will not face a high processing load. According to our experiences, the number of audio streams cascaded from non-delegate servers to a region delegate rarely exceeds two dozens, because many participants stay muted in a majority of time. That is far below the capacity of a single media server (§6.1).

A region delegate remains in the cascading tree, unless all participants in its region leave the room (details in §4.3).

4.2 Inter-Region Cascading

A spanning tree can connect a set of region delegates with the fewest cascading links. Each cascading link represents an overlay path, and the cascading links between a pair of delegates form a cascading path. The calculation of cascading spanning tree is formulated as follows.

$$
\begin{aligned}
\textit{Input}: \quad & \mathcal{R} = \{ R_1, R_2, \cdots, R_M \} \\
& \mathcal{N} = \{ N_1, N_2, \cdots, N_M \} \\
& \mathcal{D} = \{ d^L_{ij} \mid 1 \le i < j \le M \} \\
\textit{Output}: \quad & \mathcal{T} = \{ x_{ij} \mid 1 \le i < j \le M \} \\
\textit{Subject to}: \quad & x_{ij} \in \{ 0, 1 \}, \quad \forall\ 1 \le i < j \le M \\
& \sum_{j=i+1}^{M} x_{ij} + \sum_{k=1}^{i-1} x_{ki} > 0, \quad \forall\ 1 \le i \le M \\
& \sum_{i=1}^{M-1} \sum_{j=i+1}^{M} x_{ij} = M - 1
\end{aligned}
$$

We use $\mathcal{R}$, $\mathcal{N}$, and $\mathcal{D}$ to denote the set of M ($M \ge 2$) region delegates, the number of participants in each region, and the cascading link RTT between each pair of delegates, respectively. They are updated by the signaling messages triggered when a participant joins or leaves. The cascading link RTTs are measured passively via recent media flows, or proactively with infrequent probes. To lower complexity, d^L_{ij} is regarded as undirected, and takes the larger value between the RTT measured at R_i and at R_j.

The inter-region spanning tree should aim at objectives of practical importance. Therefore, classic minimum spanning tree (MST) algorithms that minimize the total RTT of involved cascading links are not suitable in this case, because it may lead to cascading paths with arbitrarily long RTTs (Figure 5d). We provide several alternatives to generate the inter-region spanning tree. Meanwhile, we illustrate using an example with 4 regions in Figure 5. The RTT statistics in the figure refer to public Azure network latency [23]. Again, we defer the discussion on updating the spanning tree along with changes in $\mathcal{R}$ and $\mathcal{N}$ to §4.3.

4.2.1 Minimized Longest Path RTT

In a spanning tree, the cascading path with the longest RTT largely determines the worst-case end-to-end audio latency. Minimizing the longest path RTT guarantees the lower bound of QoE for all participants in a room. Given the above problem formulation, the objective can be expressed as:

$$
\textit{Objective}: \quad \min \left(\arg\max_{1 \le i < j \le M} w_{ij} \cdot d^P_{ij} \right)
$$

where d^P_{ij} denotes the RTT of cascading path between R_i and R_j. In addition, we add a path weight w_{ij} to enable priority control among different regions. For example, we can assign a smaller value to w_{ki} and w_{ij} ($k < i < j$) than the other weights, if R_i is favored as the root of the spanning tree (*e.g.*, when the conference host is in the same region as R_i and we need to minimize the delay from the host to other participants).

As shown in Figure 5a, when all path weights equal 1, the spanning tree rooted at West US minimizes RTT of the longest cascading path, which is between West Europe and East Asia (299 ms). However, if the host locates in East Asia and the three cascading paths originating from there have a smaller weight of 0.5, the root of the optimal spanning tree will change to delegate server in East Asia (Figure 5b).

4.2.2 Minimized Weighted Average Path RTT

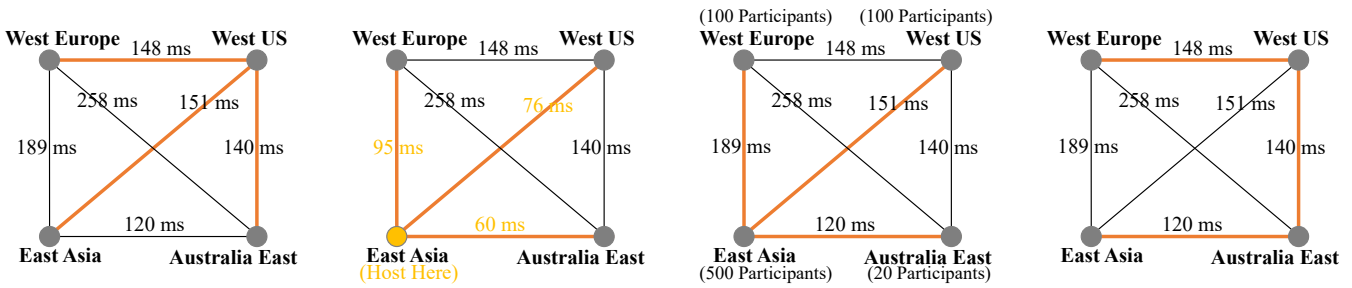
Sometimes it is commercially advantageous to enhance experience for a majority of participants. That leads to the objective of minimizing weighted average of cascading path RTTs, based on participant distribution between regions:

$$
\textit{Objective}: \quad \min \sum_{1 \le i < j \le M} w_{ij} \cdot d^P_{ij} \cdot (N_i + N_j)
$$

Compared with §4.2.1, the optimal cascading spanning tree under unity value path weights is now rooted at East Asia (Figure 5c), because 5× participants in East Asia can benefit from shorter cascading paths. However, if the weights of three cascading paths from West US is decreased to 0.5, the optimal root delegate will still be in West US. We omit that case in Figure 5 due to limited space.

4.2.3 One Single Master Delegate

Both objectives on above increase the calculation complexity compared with classic MST algorithms. For cost-effectiveness, we propose a simple heuristic instead of trying



(a) Min Longest Path **(b)** Min Longest Path (Weighted) **(c)** Min Weighted Average RTT **(d)** MST

Figure 5: Illustrating example of inter-region cascading ((a), (c) and (d): all path weights equal 1; (b) path weights between East Asia and other regions equal 0.5, RTTs of three corresponding cascading paths are weight-adjusted and shown in yellow text)

to find the optimal solution to the optimization objectives. It enforces a cascading structure similar to that in the intra-region scenario: one of the region delegates acts as the master delegate, and directly cascades to other delegates. On that basis, the unweighted objective in §4.2.1 is adopted while selecting the master delegate. We postpone exploration on the necessity and effectiveness of tuning path weights to future work. By doing so, the needed computation is determined by the number of regions per room, which is usually a relatively small number in practice. We do not choose the objective in §4.2.2, because it is sensitive to fluctuations in $\mathcal{N}$.

In Figure 5, our heuristic actually generates the optimal spanning tree under the objective in §4.2.1 (Figure 5a&5b).

4.3 Reaction to Room Dynamics

To lower the workloads of engineering development and complexity of failover, our implementation tries to avoid disassembling established cascading links in reaction to dynamics in the involved regions. Next, we elaborate on how we handle participant joining and leaving, respectively.

4.3.1 Participant Joining

We focus on how to update inter-region cascading upon newly joined participants. The intra-region case has been covered in §4.1. The following is the process of constructing an AsTree from an initially empty room.

- There is no need for inter-region cascading, until a conference involves a second region. At that time, it suffices to cascade the only two delegates.
- When the third region is added, one of the existing two delegates is selected as the master delegate (§4.2.3). Then, the third region's delegate cascades to the master, without influencing the established cascading link.
- As more regions join, we stick to the same master delegate, unless the spanning tree cannot fulfill an empirical threshold of end-to-end audio latency.

We understand that starting from the third region, the selected master delegate may not be the optimal choice under the heuristic in §4.2.3 and the objective in §4.2.1 without changing established cascading links. Yet fortunately, in practical deployment, we find the need to change master delegate is rare, especially considering that most participants are from the same country as the first joiner in a majority cases [20], and the host tends to join in the beginning of a conference.

4.3.2 Participant Leaving

Ideally, if the last participant connected to a media server in a room goes offline, the media server can be removed from the corresponding AsTree. For brevity, we call such a media server an idle server specific to the AsTree topology.

It is safe to remove an idle server if it is a leaf node in the AsTree. That is the case if it is not the region delegate, or it is the only media server in its region but not the master delegate. Meanwhile, the intra-region or inter-region cascading connections on that idle server can as well be closed, if they do not carry other multiplexed traffic.

Otherwise, the idle server is a region delegate or the master delegate, and cascades other media server(s). In the former case, if we re-select the delegate, only participants under the new delegate will become closer to others by one intra-region cascading link. Provided the latency requirements of conferencing applications nowadays, we assess that such a benefit does not compensate for the complexity of tearing down and re-establishing cascading links. Therefore, we retain the role of the idle server as region or master delegate, since that does not worsen remaining participants' QoE anyway.

As more participants leave, those idle delegate servers can be removed from the AsTree topology when the media servers below them are all removed, making them the leaf nodes.

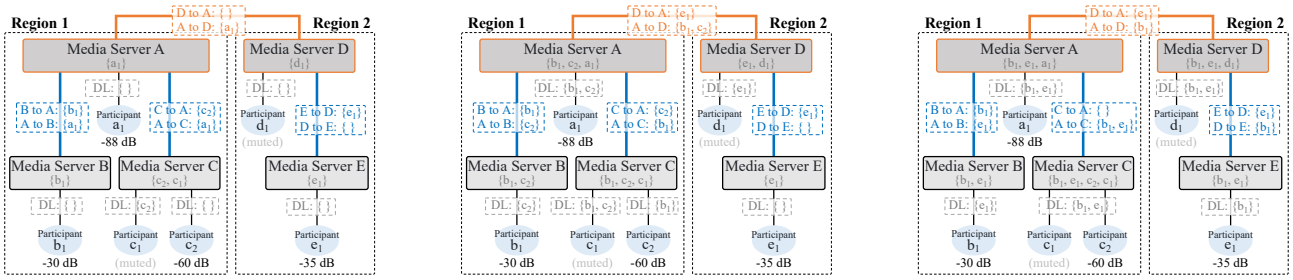
5 Select-Before-Forward Audio Selection

Different from strawman solution 3 in §2.2, AsTree expands audio selection to all cascading links. Intuitively, "select before distribute" is replaced by more generalized "select before forward". In this section, we illustrate the audio selection process, as well as how we implement dominant speaker identification and control signaling overheads.

5.1 Hop-by-Hop Audio Selection Process

All media servers adhere to a unified audio selection logic, irrespective of their roles in the cascading tree. They always select audio streams out of the following set:

- Those ingested from the connected participants.
- Those received from the cascading neighbors.



(a) Local Consensus **(b)** Intra-Region Consensus **(c)** Inter-Region Consensus

Figure 6: Illustrating the process of audio stream selection in AsTree

Then, when forwarding and distributing the selected audio streams, a media server does not send them back to where they are received. For example, a media server never sends an audio stream to its publisher again.

Another important characteristic of hop-by-hop audio selection is that it achieves asynchronous consensus on the selected active speakers among distributed media servers. Figure 6 exemplifies 3 possible states during the process (without being comprehensive). It depicts a scenario where participant a_1, b_1, c_2, and e_1 unmute together, and each media server only selects 2 audio streams for illustration.

- *Local consensus* (Figure 6a): At this phase, the unmuted audio streams are disseminated by their respective local media servers, but have not reached any cascading neighbors. So only participant c_1 hears its sibling speaker c_2.
- *Intra-region consensus* (Figure 6b): If the inter-region cascading link has much longer RTT, each region may agree on the selected intra-region active speakers, before receiving audio streams from other regions. Thus, participant c_2 is temporarily selected within region 1.
- *Inter-region consensus* (Figure 6c): In the end, every media server selects the same set of audio streams. Compared with Figure 6b, participant e_1 replaces c_2 as the second active speaker in region 1.

Figure 6 also shows that muted participants c_1 and d_1 also publish audio streams to their local media servers, which is the same as in FullAud. However, those muted audio streams are not considered in audio selection.

5.2 Dominant Speaker Identification

In Lark, RTP packets of an audio stream carry a header extension that indicates audio level [24]. It takes values from 0 to 127, representing $0 \sim -127$ dB. Media servers leverage that to select dominant active speakers, without processing raw audio. To avoid interference from non-vocal noise, we compare the weighted average audio level of each stream. It is computed from the most recent 15 audio packets (about 300 ms) of a stream, which are maintained in a per-stream ring buffer. To limit computation overheads, a media server computes a weighted average sample every 5 packets for each stream, and conducts audio selection every 50 ms (pseudo-code in Algorithm 1, with some engineering details omitted).

Instead of directly selecting L audio streams, we first pre-select at most L_i streams each time. If there are already L_i streams in the selected set S_A, a pre-selected audio stream is added to S_A only if it is louder than the least loud stream in S_A by an empirical margin (*extraCushion*). That limits the occurrence of frequent switching of selected streams. Once a stream is selected, it remains in S_A for at least a *smoothTime*. In our deployment, we set $L_i = 4$ and $L = 10$ by default.

Algorithm 1: Audio Selection

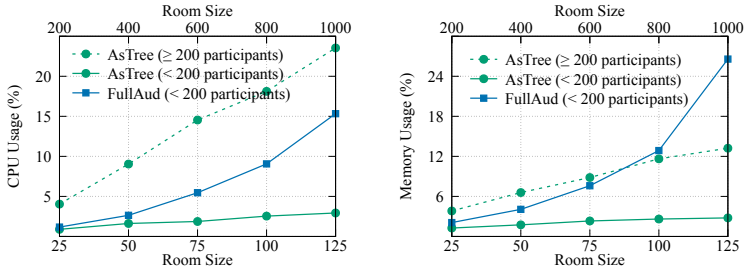
```
// S_A: set of selected audio streams
Pre-select at most L_i loudest audio streams as set S_t ;
foreach stream s ∈ S_A do
    if s ∈ S_t then
        s.lastSelected = now ;   S_t = S_t \ {s} ;
    else if now − s.lastSelected > smoothTime then
        S_A = S_A \ {s};
// louder stream has smaller audio level
foreach s ∈ S_t in order of increasing audio level do
    if |S_A| == L then break ;
    if |S_A| < L_i or s.audioLevel + extraCushion <
      GetMaxAudioLevel(S_A) then
        s.lastSelected = now ;   S_A = S_A ∪ {s} ;
```

As defined in [24], an audio level of 127 indicates a silent sample. Muted streams composed of silent samples are excluded from audio selection.

5.3 Signaling Overhead Mitigation

No Publish broadcast for audio. We do not broadcast Publish signaling messages specific to audio streams. The audio subscription process is initiated by media servers instead. Specifically, each media server caches the incoming SDP offers. Whenever a new active speaker is selected, it sends an updated SDP answer to cascaded media servers or connected participants. Immediately following that starts the transmission of audio RTP packets. Besides lowering signaling overheads, that avoids the subscription signaling round trip, and enables smooth change in the selected active speakers.

Aggregated Join. The Join message, unmerged from Publish, is solely used to notify the arrival of a new participant. When there are already a large number of participants in a room (*e.g.*, more than 50), it is both rare and hard for us to keep a close eye on every change to the participant list. Therefore, we do not necessarily need a Join broadcast specific to each newly joined participant. In that case, the local signaling unit aggregates



(a) CPU Usage **(b)** Memory Usage

Figure 7: Benchmark: AsTree uses less CPU and memory (Results of AsTree with ≥ 200 participants depicted in dashed line, refering to x2-axis on the top of the figures)

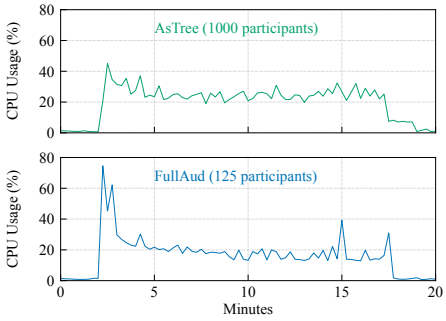


Figure 8: Example Benchmark Trials

Participant Type	Type Description
Type A	Active speakers that open both camera and microphone, and keep talking
Type B	Participants that leave their microphone unmuted but stay silent
Type C	Audience that close both the camera and the microphone

Table 2: Types of participants in benchmarking tests

newly joined participants in a short interval (*e.g.*, every 1 second) into a single signaling message. The cancellation of audio Publish and aggregation of Join dramatically mitigate signaling storm in the beginning of conferences.[3]

No Mute/Unmute signaling. In FullAud, the displayed audio status on user interface is updated strictly according to the Mute/Unmute broadcast, which has to be triggered each time a participant clicks the mute/unmute button. AsTree totally eliminates that kind of signaling broadcast. A participant is displayed as unmuted if and only if the corresponding audio stream is selected. That said, an unmuted silent participant will not be displayed as unmuted. Such a mismatch is not a very important factor in terms of user experience. Indeed, we do not receive any negative reviews on that front.

6 Benchmark

FullAud has been fully replaced by AsTree in Lark's current deployment. To compare them from a micro-level perspective, we conduct benchmarking stress tests in our test environment. Media servers there are co-located with but isolated from our production environment servers, *i.e.*, test traffic does not interfere with real-world user traffic. Three types of participants are introduced as detailed in Table 2. Through command line parameters, we can initiate a controlled number of virtual participants for each type, and specify audio/video files as their microphone/camera output. Each virtual participant runs an instance of our Linux SDK on specialized server clusters in each region. Because we only focus on evaluation of the audio subscription architecture, we need to eliminate the influence of video simulcast. Thus, we use a single-resolution (360p) video to emulate the camera video of type-A participants. Also, we limit the number of type-A participants so that all participants in a room subscribes to the same set of all published video streams. Besides, we feed two separate recorded audio tracks to type-A and type-B participants, respectively.

Unless otherwise specified, we average results of 10 runs in each test setup, to avoid interference of other test traffic.

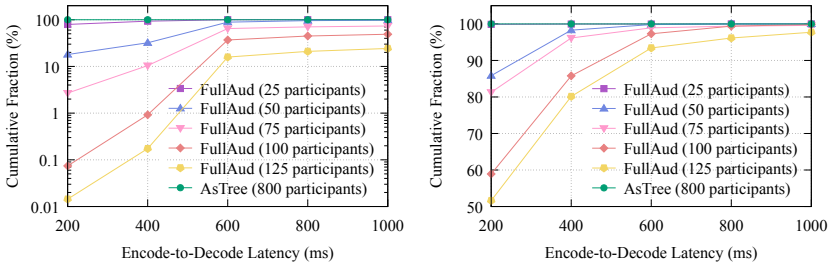
6.1 Media Server Capacity

AsTree forwards and distributes fewer audio streams and signaling messages than FullAud, and frees up considerable CPU and memory resources on media servers. To quantify that, we set up rooms with varying number of participants in a single region, all of which are forced to the same media server instance. Each conference lasts for 15 minutes, and involves 4 type-A participants, 4 type-B participants, and different number of type-C participants. In the beginning of each conference, we launch 50 participants per second, until reaching the target room size.

We first calculate average CPU and memory usage in the last 10 minutes of each trial (Figure 7). AsTree scales approximately linearly, and achieves increasingly significant gains over FullAud. That is because FullAud enforces quadratic subscriptions to all type-B and type-C participants, even though they mostly publish DTX frames. With 125 participants in a room, AsTree consumes 80.9% and 89.5% less CPU and memory than FullAud, corresponding to 5.2× conferences that can be held on a single server. Intuitively, the CPU busy time occupied by a 600-participant conference using AsTree is barely not enough for 125 participants under FullAud; the memory consumed by a 1000-participant AsTree-based conference can only support 100 participants under FullAud.

Figure 7 accounts for consumption of media processing in steady state. The capacity of a single media server also depends on the signaling overheads upon participants joining. We present the trend of CPU usage from a 125-participant conference under FullAud in Figure 8. Although it stabilizes at around 15% CPU usage, the initial peak is as high as 75% when many participants join together. In fact, given how we initiate a conference (*i.e.*, participants joining in bursts of 50), the media server instance crashes in the process of accommodating 150 participants using FullAud. That is why we only show FullAud's results with up to 125 participants in Figure 7. In comparison, the upper portion of Figure 8 exemplifies a 1000-participant room based on AsTree. The initial signaling peak is not obvious, despite having 8× participants.

[3] Leave/Unpublish in FullAud is replaced similarly.



(a) Audio **(b)** Video

Figure 9: Benchmark: AsTree reduces end-to-end latency

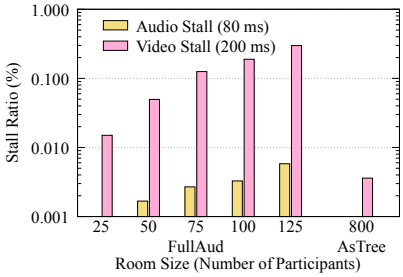


Figure 10: Benchmark: AsTree lowers stall ratio for both audio and video

Taking both media and signaling processing into consideration, AsTree can increase both the maximum conference scale and the number of concurrent rooms supported by a single server instance by as high as an order of magnitude, especially when it comes to massive-scale conferences.

6.2 User Experience

The decreased volume of traffic forwarded and distributed by AsTree mitigates cascading and last-mile congestion. That should eventually improve both audio and video QoE. To validate that, we test with a similar room setup as above, with 4 type-A participants, 4 type-B participants, and various type-C participants. This time, we load-balance participants to multiple media servers within the same region, instead of sticking to a designated media server.

We start with end-to-end latency (Figure 9), measured as the duration from when a media frame is encoded at publisher to when the same frame is decoded at subscriber. For AsTree, all audio and video frames are decoded within 200 ms. Yet due to interference from signaling storm and super-linear number of subscribed audio streams, the performance of FullAud deteriorates quickly as the room size grows. For instance, with 125 participants per room, merely 0.014% of audio frames are decoded within 200 ms, and the proportion is 51.6% for video frames. Even when we relax the latency target to 1000 ms, a 800-participant AsTree-based conference is 312% and 2.3% better than a 125-participant FullAud-based conference in terms of audio and video latency, respectively.

We note that the DTX frames from type-B and type-C participants, which have the lowest transmission and decoding priority, are also included in calculation. That leads to the extremely high audio latency by FullAud in Figure 9a. However, our test results still correctly reflect the performance trend of FullAud under different room sizes. That can be confirmed by the video latency in Figure 9b, as well.

Furthermore, lower end-to-end latency contributes to fewer stalls. In Figure 10, we plot the audio and video stall ratios, computed as the percentage of playback time in which the stall is longer than 80 ms and 200 ms, respectively. FullAud accomplishes zero audio stall only under a room size of 25 participants, while AsTree maintains zero audio stall with 800 participants. As for video stall, AsTree with 800 participants achieves a dramatic reduction of nearly two orders of magnitude, compared with FullAud with 125 participants.

Therefore, by optimizing the architecture of audio subscription, AsTree simultaneously delivers better audio and video experience at a larger conference scale.

7 Deployment

AsTree's initial deployment in Lark dates back to August 2021. It completely replaced FullAud in March 2022, and has been running at full scale since then for over two years. To demonstrate its effectiveness in improving user experience, we present the trend of three key indicators over 30 weeks ending in January 2022 in Figure 11. During that period, Lark served more than 100 million conferences. All data are normalized for confidentiality. We note that there were several parallel optimizations in Lark's audio codec starting late January 2022. We do not show results beyond that to isolate the gains by AsTree.

QoE Metrics. The audio and video stall ratios in Figure 11 account for stall duration longer than empirical thresholds of 80 ms and 200 ms, respectively. Over the presented period after AsTree was deployed, they were improved significantly at the same time: the median and 95th percentile audio stall ratios were reduced by more than 30% and 45%; corresponding reductions in video stall ratios both exceeded 50%. In practical deployment, an important factor contributing to such improvements is that AsTree alleviates congestion on participants' access links, which usually have more limited bandwidth than our test environment for stress tests (§6.2).

Additionally, we present the audio stall ratio partitioned in different range of room sizes in Figure 12. It is averaged from a recent month. We can see that a conference with more than 350 participants does not even double the audio stall ratio over a small-scale conference with only 3 to 5 participants. We regard that as acceptable for a global system like Lark, considering increasingly heterogeneous participant access bandwidth and geo-distributed regions under larger rooms.

User Satisfaction. Along with better QoE comes higher user satisfaction. We observed a reduction of around 40% in the ratio of negative reviews in Figure 11. The ratio of negative reviews complaining specifically about bad audio or video experience was also reduced by 40%, closely following the trend of overall negative reviews. That indicates the QoE gains by AsTree were the dominating reason that raised our

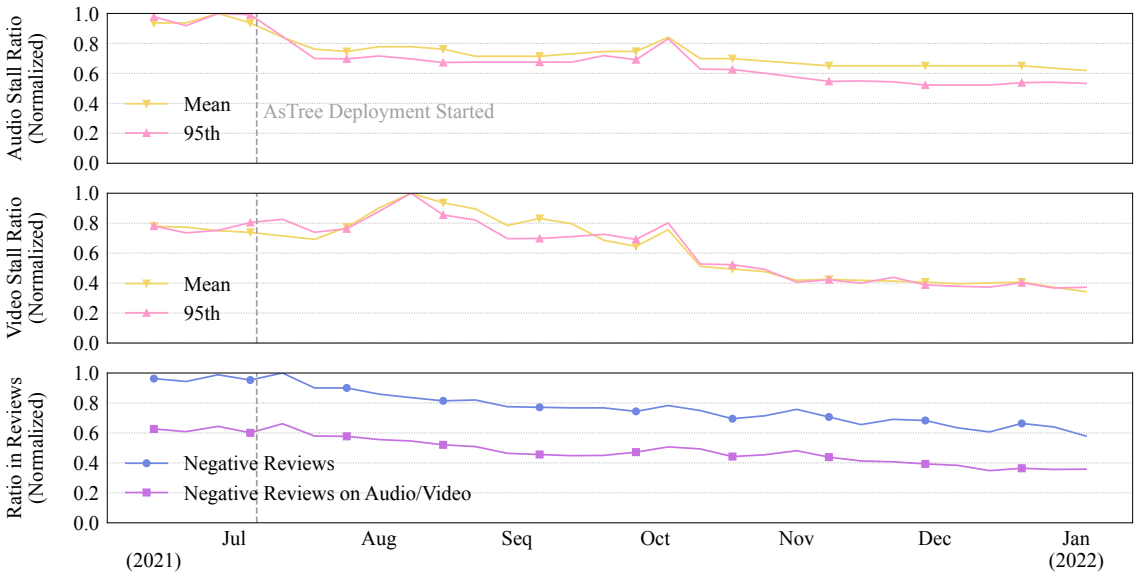


Figure 11: Trend of stall ratios, and negative review ratio after AsTree was deployed (median and 95th stall ratios are normalized independently)

user satisfaction. Moreover, if we look at the ratio of negative reviews under different room sizes (Figure 13, averaged from a month for both architectures), AsTree effectively narrows the gap of user experience under large and small conferences.

Cascading Complexity. A remaining concern on AsTree is whether it induces excessive cascading links and impacts user experience, *i.e.*, due to selection of region and master delegates. Enhanced user satisfaction shown above implies that does not occur to most participants. To confirm that more directly, we monitored the number of cascading links needed to forward and distribute a subscribed stream for two weeks earlier in 2024. On average, Lark only increases the number of cascading links per stream by 5.7% compared with other ByteDance RTC applications. Thus, the AsTree cascading topology has restricted adverse influence on QoE, indeed.

8 Discussion

In this section, we share some practical insights drawn from deployment experiences, and potential future work directions.

8.1 More Implementation Details

Participant aggregation. Because signaling storm is significantly mitigated, AsTree enables us to aggregate more participants on a single media server, so as to reduce the number of intra-region cascading links. Participant aggregation saves server-side costs on media servers and bandwidth, and lowers the complexity of participant migration upon failover. While achieving aggregation, we may need to redirect a joining participant to a different media server from the one retrieved through DNS. In that case, we notify the participant with a customized signaling message. Other approaches, such as DNS redirection as adopted by many CDN systems [12, 25], may also fulfill the same purpose. Meanwhile, to avoid overloading a single media server, we configure the upper limit of aggregation to 200 participants per room per server.

Cascading connection reuse. Before forwarding audio streams to and from a newly involved media server, we need to establish a cascading connection and accomplish SDP negotiation. To reduce latency and start media transmission as soon as possible, we try to reuse existing connections on the same cascading link. That means a cascading connection may carry audio streams of multiple conference rooms. Two corresponding configuration parameters are introduced. First, we restrict the maximum number of (RTP) streams allowed on a cascading connection, in case they impact the processing efficiency of the thread worker. Second, we set a timeout of several seconds before closing an idle cascading connection, during which it may be reused again.

Decoupled video and audio subscription. In the media plane, the AsTree cascading tree topology is independent of the routing and forwarding of video streams. In the control plane, the bandwidth reserved for audio streams before allocating to video streams of different bitrates is bounded and thus, more predictable. That ensures great flexibility for innovations in either video or audio subscription architecture. We do not need to worry about interfering video simulcast while implementing and deploying AsTree. As for the synchronization of video and audio streams of the same publisher, we rely on jitter buffer on client side. That fulfills the current latency requirements of multi-party conferencing.

Fault tolerance. Fault tolerance in AsTree is in essence the same as in video simulcast, *i.e.*, ensuring connectivity between each pair of participants. We mainly handle two types of failures, which are broken cadcading link and crashed server/cluster. The routing of each overlay cascading link is managed by our centralized SDN controller. When we observe abnormal RTT inflation and loss rate on a specific cascading link, we switch to a back-up path without awareness of conference participants and SFU media servers. If a

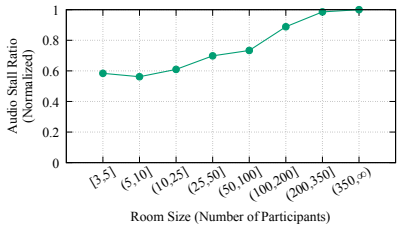


Figure 12: Audio stall ratio after AsTree is fully deployed

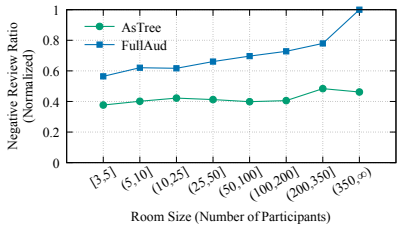


Figure 13: Negative review ratios before/after AsTree's deployment

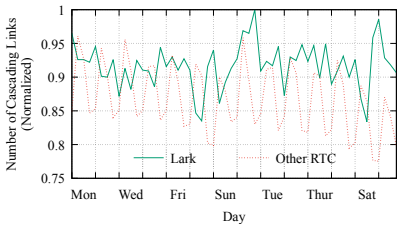


Figure 14: Number of cascading links per stream subscription

media server or a cluster crashes, participants already connected to a faulty server will go through a connection timeout, triggering a request to the control plane for a new SFU. The original faulty server is replaced by the new SFU in the AsTree topology, and overlay cascading paths are re-established. In that process, the faulty server(s) will be blacklisted by our scheduler until they recover from connection issues.

8.2 Other Implementation Options

Mesh *vs.* AsTree. The select-before-forward audio selection as detailed in §5 can also work when media servers are cascaded by a mesh. In that case, each media server selects audio streams from those injected from its connected participants, and forwards to all the other media servers. Compared with an AsTree, a mesh topology has shorter cascading paths on average, but induces more cascaded audio streams. For example, a conference room involving M media servers cascades at most $2L(M-1)$ audio streams with a cascading tree, which is no larger than $L \cdot M(M-1)$ when using a mesh. Considering that AsTree can satisfy the latency requirements of multi-party conferencing, we adopt it for lower deployment cost.

Audio stream SSRCs. Some existing service providers use a fixed set of negotiated RTP SSRCs [26] for the selected audio streams [27, 28]. Our web SDK also adopts similar solution, due to lower development workloads based on the WebRTC codebase. However, that requires the media server to make sure the participants receive consecutive sequence number and monotonous timestamp under each SSRC. To avoid such processing overheads, while implementing the native SDK of Lark application, we choose to let each media server generate a random SSRC each time an audio stream is selected, even if it is from a speaker that is previously selected. That said, different participants may observe different SSRCs for the same selected audio stream at the same moment; and the same participant may observe different SSRCs for the same speaker during a conference. The identity of each selected speaker is notified to participants through the updated SDP answer triggered by the media server (§5.3).

8.3 Effectiveness and Applicability

Benefits for all conference scales. Although designed to support massive-scale conferences, AsTree achieves significant gains in small-size conferences, as well. That is because a muted audio stream, although requiring lower bandwidth, consumes almost the same amount of memory and comparable CPU as an audio stream from an active speaker. Therefore, AsTree can serve as a general-purpose architecture that benefits a broad spectrum of cases, and we enable it by default.

Extended application scenarios. AsTree has been adapted to support more advanced use cases. An example is to provide simultaneous interpretation for foreign participants in international conferences. A channel attribute can be added to the original and interpretation audio streams. Participants can subscribe to the original channel, the interpretation channel, or both channels on demand from Lark user interface, and selection of audio streams at each media server can be conducted accordingly on a per-channel basis.

8.4 Future Directions

Joint video and audio optimization. Although audio and video subscription are decoupled in media plane, there may be QoE benefits to jointly optimize the cascading topology of video and audio streams in control plane. For example, in a typical conference where an invited speaker presents slides via screen sharing, the audio and screen video streams of the presenter may share a cascading tree rooted at the corresponding local media server, which also facilitate the synchronization of the speaker's audio and video signals.

Dynamic AsTree topology deformation. Given latency requirement of multi-party conferencing nowadays (*e.g.*, up to hundreds of milliseconds), we adopt a straightforward heuristic to select intra-region and master delegates, and rarely change the role of each media server (§4.3) during an ongoing conference. Yet doing so improves user experience in some cases. In the above example with an invited speaker, if we continue to the next speaker on the agenda who is in another region, it lowers the latency from the new speaker to the audience to switch the master delegate, as well. Such a deformation may even become inevitable as we keep pursuing more real-time interactivity. Therefore, we plan to work on a highly efficient deformation mechanism based on an established cascading topology. In the example of switching to a new speaker, it may be triggered by dynamically assigned cascading path weights. Besides, it can also be used for media server migration and failover.

Formulation of AsTree calculation. Similar cascading topology may also apply in other application scenarios such as live streaming. Our current objectives listed in §4.2 only optimize

the cascading path latency. When aiming at more latency-sensitive future-proof applications, we may need to refine the problem formulation and express the QoE target more accurately. For example, an additional term representing per-media server processing latency may be added, if that accounts for a fair proportion in the overall end-to-end latency. For compute-intensive applications (*e.g.*, cloud rendering [29, 30], artificial video generation [31]), the computing resource consumption may be explicitly captured, especially when leveraging limited computing power at the edge [32]. Furthermore, in extreme cases with multiple concurrent continuous publishers, a more complicated topology than a single tree may be needed.

9 Related Works

Dominant speaker identification (DSI). DSI is not a new problem in the context of multi-party conferencing. Volfin *et al.* identified the dominant speaker by computing per-participant immediate, medium-term, and long-term speech activity scores [11]. However, the computation was based on frequency representation of audio frames. That is not available on an SFU without decoding audio streams. Inspired by [11], Jitsi introduced LastN [10], and computed speech activity score directly from audio levels carried in RTP header [24]. Different from AsTree, LastN was used to make sure only dominant speakers' video streams are forwarded. It remained full subscription for audio streams. Similar DSI method is often utilized in video subscription to track and render the active speaker view [33, 34].

Audio selection. Full audio subscription was adopted by many service providers previously, some of which stick to that till now (*e.g.*, Twilio [8]). As the popular conference scale continues to grow, several service providers have started to support audio selection based on DSI in recent years. The audio mixer in early-stage WebRTC codebase mixed at most 3 audio sources [35], but it ran on client side. Google Meet and Google Duo forwarded the 3 loudest speakers with fixed SSRCs [27]. Jitsi introduced a "loudest" feature, which only forwards a configured number of loudest audio streams [36]. Agora allowed audio-strength stream selection based on audio volume [37]. Nevertheless, limited details are revealed about those schemes. Some providers such as Jitsi and Agora still disable audio selection in their default configuration. AsTree is the first to share the architecture and performance of audio selection from a large-scale practical deployment.

Conferencing architecture. For audio subscription, there are still a few services implementing Multipoint Control Unit (MCU) to get a single mixed audio streams [38, 39]. Yet for video subscription, SFU has been the choice by a majority of popular conferencing providers [6, 16, 17, 40], due to lower overheads on server side. Both simulcast and SVC can rely on the SFU architecture. In simulcast, each participant encodes multiple video streams in different bitrates, and the SFU determines which stream to forward to which participant. A representative example is GSO-Simulcast [5], where a centralized controller orchestrated the resolution and bitrate of video streams. SVC leverages scalable video codec [41] to include multiple layers in a single video stream, so that it can be adapted to multiple bitrates. Zoom is claimed to belong to this category [42]. AsTree is compatible with the SFU-based video subscription architecture.

In addition, focusing on routing in the media plane, existing works such as LiveNet [12] and VIA [43] constructed overlay cascading paths, enabling participants to access the service through the nearby edge. Aimed at resource provisioning for conferencing services, Switchboard [20] enforced peak-aware server allocation and capacity provisioning. Those works are complementary to AsTree.

Measurements of multi-party conferencing. Recognizing the importance of multi-party conferencing and the challenges it brings to the network, researchers conducted extensive measurements of popular applications on the market [6, 44–48]. On one hand, by analyzing their protocol encapsulation and session behaviors from collected traces, they confirmed the usage of simulcast by many conferencing applications, including Google Meet, Microsoft Teams, WebEx [44]. On the other hand, owing to lack of abundant resources of media servers, it is hard to capture the global picture under massive conference scales. For example, authors of [46] conducted small-scale experiments with only up to 6 participants.

10 Conclusion

In this paper, we present our design of AsTree, the architecture specifically for audio subscription aimed at massive-scale multi-party conferencing in Lark. AsTree evolved from our early-stage implementation of FullAud, which was based on full audio subscription. For each conference room, it constructs a two-hierarchy (intra-region and inter-region) cascading tree composed of all involved media servers, and enforces per-hop audio selection while forwarding audio streams. AsTree eliminates the signaling broadcasts triggered by participants' muting/unmuting operations.

After serving millions of conferences in several years of deployment, AsTree is proved to significantly improve the scalability of Lark. Thanks to dramatically mitigated signaling storm and fewer number of audio streams to forward, AsTree improves the capacity of media servers by nearly an order of magnitude compared with FullAud. Meanwhile, less congestion on last-mile and cascading links results in enhanced QoE. In our practical deployment, the median audio and video stall ratios are reduced by 30% and 50%, respectively. That comes with a 40% reduction in the ratio of negative reviews.

To fulfill even more stringent latency requirements, we envision more flexible and dynamic cascading topology calculated from more realistic QoE objectives in the future.

Acknowledgements

We thank our shepherd, Kurtis Heimerl, and the anonymous SIGCOMM reviewers for their valuable comments.

References

[1] Lark. https://www.larksuite.com/. Accessed: 2024.09.

[2] Zoom. https://zoom.us/. Accessed: 2024.09.

[3] Microsoft Teams. https://www.microsoft.com/en-us/microsoft-teams/group-chat-software. Accessed: 2024.09.

[4] Alibaba DingTalk. https://www.dingtalk.com/. Accessed: 2024.09.

[5] Xianshang Lin, Yunfei Ma, Junshao Zhang, Yao Cui, Jing Li, Shi Bai, Ziyue Zhang, Dennis Cai, Hongqiang Harry Liu, and Ming Zhang. GSO-simulcast: global stream orchestration in simulcast video conferencing systems. In *ACM SIGCOMM*, 2022.

[6] Oliver Michel, Satadal Sengupta, Hyojoon Kim, Ravi Netravali, and Jennifer Rexford. Enabling passive measurement of zoom performance in production networks. In *ACM IMC*, 2022.

[7] Jean-Marc Valin, Koen Vos, and Timothy B. Terriberry. Definition of the Opus Audio Codec. RFC 6716, September 2012.

[8] Twilio Track Subscriptions. https://www.twilio.com/docs/video/api/track-subscriptions. Accessed: 2024.09.

[9] Alibaba RTC SDK. https://www.alibabacloud.com/help/en/live/user-guide/android-alirtcengine-class#0155f66448env.

[10] Boris Grozev, Lyubomir Marinov, Varun Singh, and Emil Ivov. Last N: relevance-based selectivity for forwarding video in multimedia conferences. In *ACM NOSSDAV*, 2015.

[11] Ilana Volfin and Israel Cohen. Dominant speaker identification for multipoint videoconferencing. *Computer Speech & Language*, 27(4):895–910, 2013.

[12] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, et al. Livenet: a low-latency video transport network for large-scale live streaming. In *ACM SIGCOMM*, 2022.

[13] Boris Grozev. Improving Scale and Media Quality with Cascading SFUs. https://webrtchacks.com/sfu-cascading/. Accessed: 2024.09.

[14] Bandwidth Requirements for Lark Video Meetings. https://www.larksuite.com/hc/en-US/articles/360048488053-what-are-the-bandwidth-requirements-for-video-meetings. Accessed: 2024.09.

[15] Boris Grozev. Bridge cascading with geo-location is back. https://jitsi.org/blog/bridge-cascading-is-back/, 2022.

[16] Jitsi Videobridge. https://jitsi.org/jitsi-videobridge/. Accessed: 2024.09.

[17] Amazon Chime Video Simulcast. https://aws.github.io/amazon-chime-sdk-js/modules/simulcast.html. Accessed: 2024.09.

[18] Sudarshan Vasudevan, Jim Kurose, and Don Towsley. Design and analysis of a leader election algorithm for mobile ad hoc networks. In *IEEE ICNP*, 2004.

[19] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *ACM-SIAM SODA*, 2006.

[20] Rahul Bothra, Rohan Gandhi, Ranjita Bhagwan, Venkata N Padmanabhan, Rui Liang, Steve Carlson, Vinayaka Kamath, Sreangsu Acharyya, Ken Sueda, Somesh Chaturmohta, et al. Switchboard: Efficient Resource Management for Conferencing Services. In *ACM SIGCOMM*, 2023.

[21] Recommendation ITU-T G.114. One-way transmission time. Telecommunication standardization sector, International Telecommunication Union, May 2003.

[22] Twilio Glossary: What is Latency? https://www.twilio.com/docs/glossary/what-is-latency. Accessed: 2024.09.

[23] Azure Network Latency. https://learn.microsoft.com/en-us/azure/networking/azure-network-latency. Accessed: 2024.05.

[24] Jonathan Lennox, Enrico Marocco, and Emil Ivov. A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication. RFC 6464, December 2011.

[25] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. *ACM SIGCOMM*, 2015.

[26] Henning Schulzrinne, Stephen L. Casner, Ron Frederick, and Van Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.

[27] Gustavo Garcia. Meet vs. Duo – 2 faces of Google's WebRTC. https://webrtchacks.com/meet-vs-duo-2-faces-of-googles-webrtc/, 2022.

[28] Jaya Allamsetty. Improving performance on very large calls: introducing SSRC rewriting. https://jitsi.org/blog/improving-performance-on-very-large-calls-introducing-ssrc-rewriting/, 2024.

[29] Serhan Gül, Dimitri Podborski, Jangwoo Son, Gurdeep Singh Bhullar, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. Cloud rendering-based volumetric video streaming system for mixed reality services. In *ACM MMSys*, 2020.

[30] Yongjie Guan, Xueyu Hou, Nan Wu, Bo Han, and Tao Han. MetaStream: Live Volumetric Content Capture, Creation, Delivery, and Rendering in Real Time. In *ACM MobiCom*, 2023.

[31] Sora: Creating video from text. https://openai.com/index/sora/. Accessed: 2024.09.

[32] Na Li and Yao Liu. EVASR: Edge-Based Video Delivery with Salience-Aware Super-Resolution. In *ACM MMSys*, 2023.

[33] Zoom SDK. https://zoom.github.io/zoom-sdk-android/us/zoom/sdk/InMeetingServiceListener.html#onActiveSpeakerVideoUserChanged-long-. Accessed: 2024.09.

[34] Azure Communication Services SDK: Get active speakers within a call. https://learn.microsoft.com/en-us/azure/communication-services/how-tos/calling-sdk/dominant-speaker. Accessed: 2024.09.

[35] WebRTC Audio Mixer. https://source.chromium.org/chromium/chromium/src/+/main:third_party/webrtc/modules/audio_mixer/audio_mixer_impl.h;l=50;drc=0df0faefd5ea4df2364cccc0f8449431bf8200d0. Accessed: 2024.09.

[36] Jitsi Loudest: Only forward audio packets of the loudest speakers in a conference. https://github.com/jitsi/jitsi-videobridge/pull/1677/commits. Accessed: 2024.09.

[37] Agora: Audio Strength Stream Selection. https://docs.agora.io/en/video-calling/advanced-features/audio-strength-stream-selection?platform=android. Accessed: 2024.09.

[38] Amazon Chime Meeting Architecture. https://docs.aws.amazon.com/chime-sdk/latest/dg/meetings-sdk.html#mtg-arch. Accessed: 2024.09.

[39] Quobis: Hybrid architecture based on MCUs and SFUs. https://quobis.com/2021/05/03/sfus-vs-mcus-which-is-the-best-way-to-manage-multi-conferencing/. Accessed: 2024.09.

[40] An introduction to WebRTC Simulcast. https://blog.livekit.io/an-introduction-to-webrtc-simulcast-6c5f1f6402eb/, 2021.

[41] Scalable Video Coding (SVC) Extension for WebRTC. https://www.w3.org/TR/webrtc-svc/. Accessed: 2024.09.

[42] A Short on How Zoom Works. https://highscalability.com/a-short-on-how-zoom-works/. Accessed: 2024.09.

[43] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, et al. Via: Improving internet telephony call quality using predictive relay selection. In *ACM SIGCOMM*, 2016.

[44] Antonio Nistico, Dena Markudova, Martino Trevisan, Michela Meo, and Giovanna Carofiglio. A comparative study of rtc applications. In *IEEE International Symposium on Multimedia (ISM)*, 2020.

[45] Insoo Lee, Jinsung Lee, Kyunghan Lee, Dirk Grunwald, and Sangtae Ha. Demystifying commercial video conferencing applications. In *ACM Multimedia*, 2021.

[46] Hyunseok Chang, Matteo Varvello, Fang Hao, and Sarit Mukherjee. Can you see me now? a measurement study of zoom, webex, and meet. In *ACM IMC*, 2021.

[47] Mehdi Karamollahi, Carey Williamson, and Martin Arlitt. Packet-level analysis of zoom performance anomalies. In *ACM/SPEC ICPE*, 2023.

[48] Albert Choi, Mehdi Karamollahi, Carey Williamson, and Martin Arlitt. Zoom session quality: A network-level view. In *PAM*, 2022.