DecVi: Adaptive Video Conferencing on Open Peer-to-Peer Networks

Jingren Wei
wei.1276@osu.edu
The Ohio State University
Columbus, Ohio, USA

Shaileshh Bojja Venkatakrishnan
bojjavenkatakrishnan.2@osu.edu
The Ohio State University
Columbus, Ohio, USA

ABSTRACT

Video conferencing has become the preferred way of interacting virtually. Current video conferencing applications, like Zoom, Teams or WebEx, are centralized, cloud-based platforms whose performance crucially depends on the proximity of clients to their data centers. Clients from low-income countries are particularly affected as most data centers from major cloud providers are located in economically advanced nations. Centralized conferencing applications also suffer from occasional outages and are embattled by serious privacy violation allegations. In recent years, decentralized video conferencing applications built over p2p networks and incentivized through blockchain are becoming popular. A key characteristic of these networks is their openness: anyone can host a media server on the network. The reason, however, also leads to a security problem: a server may obfuscate its true location in order to gain an unfair business advantage. We propose DecVi, a decentralized multicast tree construction protocol that adaptively discovers efficient tree structures based on an exploration-exploitation framework. DecVi is motivated by the combinatorial multi-armed bandit problem and uses a succinct learning model to compute effective actions. Despite operating in a multi-agent setting with each server having only limited knowledge of the global network and without cooperation among servers, experimentally we show DecVi achieves similar quality-of-experience compared to a centralized globally optimal algorithm while achieving higher reliability and flexibility.

KEYWORDS

video conferencing, decentralization, peer-to-peer

1 INTRODUCTION

Video conferencing has become an ubiquitous method of virtual interaction for many—with millions of meetings happening daily. A bulk of the video conferencing sessions today occurs over cloud-based platforms such as Zoom, Microsoft Teams and Cisco Webex. In these systems, media servers housed in massive data centers act as central points of aggregation for receiving and distributing video streams across participants of a conferencing session [7]. In recent years a number of decentralized and open peer-to-peer (p2p) video conferencing platforms are also being developed in the industry [3]. Decentralized conferencing solutions carry unique benefits making them promising as alternative platforms to existing centralized counterparts. Some of the important benefits include:

- **Quality-of-experience (QoE).** Centralized video conferencing frameworks employ only a limited number of data center regions for media service, which causes disruptions to sessions under high demands, and excessive streaming lag.

- **Availability.** In a p2p network it is unlikely for multiple, independent servers to go down at the same time, minimizing chances of a network-wide outage. Whereas in centralized platforms outages are not uncommon, affecting millions of clients each time [4].

- **Privacy and censorship.** There is also a growing concern about the privacy and censorship practices of centralized video conferencing providers [1]. In contrast, the open nature of p2p conferencing systems makes it a fully transparent medium where no single party has the ability to unilaterally censor users or collect their data.

A key challenge in building a large scale and open p2p conferencing network is how to determine efficient paths over which to deliver client media streams. For good QoE, a streaming path must have low latency and sufficient bandwidth available to carry video chunks of appropriate quality, as requested by a receiving client’s application. Thus, for conferencing sessions involving a large number of clients it becomes necessary to route streams over multiple servers structured as a multicast tree. Determining a good subset of servers for the multicast tree, and how to structure the multicast tree to maximize QoE, are nontrivial problems whose solution depends on various factors. These factors are not only highly heterogeneous across peers but can also be time varying which further complicates the problem.

We present DecVi, a fully decentralized and efficient route computation algorithm for large scale, p2p conferencing networks.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICDCN 2023, January 4–7, 2023, Kharagpur, India
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9796-4/23/01 ... $15.00
https://doi.org/10.1145/3571306.3571432
DecVi a multicast tree of servers is built for each source of a video stream in a conferencing session, which is iteratively refined over time using an exploration-exploitation framework for increasing client QoE. DecVi is inspired by the combinatorial multi-armed bandit problem [11], with a server in a multicast tree treating downstream servers to whom it must forward streams to and the corresponding stream qualities, as the ‘arms’ of a bandit problem. The streaming latency and quality of stream received by a client form the ‘reward’ earned, which is communicated by the client to the multicast tree servers as feedback.

Prior works have proposed distributed algorithms based on analytical models for multiparty video conferencing with multiple media servers for relaying [9]. A model-based algorithm requires clients and servers to have knowledge of the model parameters to run. In contrast, DecVi is an entirely adaptive solution which does not require clients or servers to have knowledge of the system parameters.

More recently, a number of works have presented p2p conferencing designs for WebRTC [10]. These designs, however, consider only a single media server for connecting the clients. With the development of scalable video coding [6], selective forwarding units (SFUs) are emerging as lightweight, low-delay media servers (with forwarding delay typically < 20ms [5]). The low complexity of SFUs has encouraged operators to deploy a cascade of SFUs for geo-distributed conferencing sessions for efficiency [8]. Dynamically deciding which SFUs to utilize for a conferencing session with cascaded SFUs is still an active area of research.

The contributions of this paper are:

(1) We formulate multicast tree construction for video conferencing in open, p2p systems as a distributed learning problem. A learning approach to p2p design for streaming video has not been previously proposed, to our best knowledge.

(2) We propose a decentralized algorithm for tree construction inspired by the combinatorial multi-armed bandit problem. However, our work is different from standard multi-agent bandit settings which consider all agents working on instances of the same bandit problem or sharing the same reward function globally. The algorithm and empirical results we provide may be of independent interest to the multi-agent bandit research community.

2 SYSTEM MODEL

We consider a video conferencing session between a set of clients $C$, streamed using a subset of media servers from a set $S$ of available media servers. Each client $c \in C$ is the source of a video stream that it seeks to distribute to all the other clients in $C$. A client distributes its stream by forming a multicast tree with the client as the root, all the other clients as the leaves, and one or more media servers as the interior nodes of the tree (Figure 1). Each client forms its multicast tree independently of the multicast trees of other clients.\footnote{Our approach can be generalized to construct a single, common multicast tree to disseminate all streams. We leave a systematic study of such an approach to future work.}

In the remainder we therefore restrict ourselves to a single client source $c_0 \in C$, and focus on how to construct an efficient multicast tree for $c_0$.

2.1 Network Model

We model the streaming network as a graph $G$ with SFUs $S$ and clients $C$ forming the nodes. For two nodes $u, v \in G$, there is a link $(u, v) \in G$ between the nodes if $u$ sends a stream to $v$ in the multicast tree for $c_0$. Each link $(u, v) \in G$ has a latency $l(u, v) \geq 0$, which is the time it takes for packets sent from $u$ to reach $v$. Each SFU $s \in S$ has a bandwidth limit of $b_s \in \mathbb{N} = \{0, 1, 2, \ldots \} \geq 0$, which is the maximum rate at which it can upload streams to other nodes.

Video is encoded through scalable video coding at the source with a maximum of $Q \in \mathbb{N}$ layers. An SFU $s \in S$ that receives $q$ layers of the stream may forward at most $q$ layers to each of its downstream nodes in the multicast tree. Latency of a path $c_0, s_0, s_1, \ldots, s_k, c$ in the multicast tree is the overall delay $l(c_0, s_0) + \sum_{i=0}^{k-1} l(s_i, s_{i+1}) + l(s_k, c)$ incurred by video packets from the source to the destination client.

2.2 Multicast Tree Construction

We consider fully distributed algorithms for the multicast tree construction. Initially the tree comprises of just a single SFU which disseminates the stream to all clients. While deciding the downstream nodes, the SFU must also decide the number of layers to send to each of the nodes, and the subset of clients each node (that is not a client) is responsible for distributing the stream to. We call these decisions as the SFU’s action. If a node is responsible for distributing the stream to a client $c \in C$, the node must take actions such that the node is an ancestor of $c$ in the multicast tree. The source $c_0$ can also change its immediate downstream SFU. We assume $c_0$ always sends its stream to a single SFU, which can then distribute the stream to other SFUs if needed. SFU and $c_0$’s actions are executed such that the video stream is received uninterrupted by the other clients at all times. To alter the multicast tree, the source $c_0$ sends a special trigger_action packet along with the video stream down the existing multicast tree. When a node receives a trigger_action packet, it takes an action and forwards the message to its latest set of downstream nodes. We call each trigger_action packet sent by $c_0$ as a round.

2.3 Objective

The objective is to construct a multicast tree which maximizes the aggregate reward (utility) of the clients. From the feedback received from clients, each SFU $s$ on the multicast tree computes the reward for client $c \in C_s$ as

$$r_s(c) = -d(s, c) + aq(c)/q^*(c),$$

where $C_s$ is the set of clients that $s$ is responsible for, $d(s, c)$ is the delay between $s$ and $c$ as estimated from the get_delay packet and $q(c)$ is the number of layers $c$ receives. $\alpha \geq 0$ is a user-defined parameter that dictates the trade-off between optimizing delay and video quality. The SFU chooses its actions so that the aggregate reward $\sum_{c \in C_s} r_s(c)$ is maximized.

3 MOTIVATION

To illustrate the problem and its challenges, consider the example of a conference session with 5 clients and four SFUs as shown in Figure 1(a). Suppose the nodes are located on a two-d space with the Euclidean distance between two nodes signifying the latency
of sending a packet between the nodes. We focus on client 0 as the source in this example. Each SFU has a bandwidth limit of 6 units, while each client requests to receive 3 layers of the video stream with as low a latency as possible. The source client generates 5 layers. In a centralized system, the source sends the stream to a central SFU which then forwards the stream to all other clients. However, with a bandwidth limit of 6 units, the central SFU cannot support high-quality streams (3 layers) to all 4 clients. Whereas by using 3 SFUs in the multicast tree, we can deliver 3 layer streams to all 4 clients at the cost of a marginal increase in latency compared to the centralized case. The key challenge is how we can discover such efficient multicast tree structures through decentralized algorithms in an open setting, where comprehensive knowledge of locations of all SFUs currently online and their capacities are not known. Even with global knowledge, computing optimal tree structures is computationally expensive (§5). In our proposed algorithm DecVi (§4), starting from an initial distribution tree SFUs gradually updates the tree to the state shown in Figure 1(a). Figure 1(b) shows an example of a multicast tree that is suboptimal for this setting. Whereas DecVi automatically learns to ignore SFU 3 to produce the best tree configuration.

4 DecVi Design

We present DecVi, an adaptive, decentralized algorithm to compute efficient multicast routing trees in a short amount of time. Each agent (i.e., an SFU or client) maintains a succinct model of the environment based on its past interactions with the network (past actions, and observed rewards) using which it computes the best action to take next. Using a model provides a convenient way to summarize observations from past interactions, without significant loss of useful information, while consuming minimal storage. DecVi also balances exploitative actions with exploratory actions, in which an action is randomly selected, which helps discover unseen SFU candidates with potentially good performance. An outline of DecVi is presented in Algorithm 1.

When an SFU $s$ in the multicast tree receives a trigger_action message, it uses the current model to estimate what is the best action to take. To encourage exploration, the best action predicted by the model is executed with probability $1 - \epsilon$, where $\epsilon \in (0, 1)$ is a configurable parameter. With probability $\epsilon$ a random action is taken. To avoid frequent tree updates, in practice we can run DecVi in the background to discover new, efficient trees while using a stale tree to perform the actual video streaming.

Model (model in Algorithm 1). At each SFU $s$ (and client $c_0$), we consider a complete bipartite graph model $G(S, C_s)$ where $S$ is the set of SFUs known to $s$ (including itself) and $C_s$ is the set of clients $s$ is responsible for. For simplicity, in our evaluations we assume $\mathbb{S} = S$, the global set of SFUs. Each edge $(s, c') \in G$ has two weights associated with it: $\phi(s, s', c')$ which is an estimate of the latency between $s$ and $c'$ if the stream is forwarded through $s'$ (and $s$ is responsible for delivering the stream to $c'$), and $\hat{\phi}(s', c')$ which is an estimate of the average number of layers $c'$ receives if it forwards the stream to $s'$ (and $s$ is responsible for delivering the stream to $c'$ per unit layer sent to $s'$). In other words, if $s$ sends $q(s, s')$ layers to $s'$ and makes $s'$ responsible for $c'$, we estimate $q(s, s') \times \hat{\phi}(s', c')$ as the number of layers $c'$ receives. Maintaining an estimate of $\hat{\phi}(s', c')$ of the number of layers received by $c'$ per unit layer sent to $s'$ allows us to estimate the number of layers received by $c'$ based on the action $q(s, s')$ at $s'$.

Computing the best action (EstimateBestAction() in Algorithm 1). An action at SFU $s$ can be represented using two functions: a client assignment map $a(s, \cdot) : C \rightarrow S$ and number of layers forwarded $q(s, \cdot) : S \cup C \rightarrow \mathbb{N}$ where $\mathbb{N} = \{0, 1, 2, \ldots\}$. If $a(s, c') = s'$, it means $s$ forwards a non-zero number of layers to $s'$ while also informing $s'$ to be responsible for $c'$. If $q(s, s') > 0$ for $s' \in S$, it means SFU $s$ forwards $q(s, s')$ layers to $s'$. Similarly, if $q(s, c') > 0$ for $c' \in C$, it means $s$ forwards $q(s, c')$ layers directly to $c'$. For consistency, we must have $q(s, s') > 0$ iff $\exists c' \in C$ such that $a(s, c') = s'$ for any $s' \in S$. Similarly, $q(s, c') > 0$ iff $a(s, c') = s$ for any $c' \in C$.

\begin{algorithm}
\caption{DecVi: Algorithm outline for computing action at SFU $s$ during a round.}
\begin{algorithmic}
\State \textbf{input}: trigger_action packet from parent node, number of layers $q_{in}(s)$ received from parent node of $s$, set of clients $C_s$ that $s$ is responsible for, IP addresses of clients in $C_s$, layer requirements of clients in $C_s$, exploration parameter $\epsilon \in (0, 1)$
\State \textbf{output}: action $= \{ \text{set of nodes } \Gamma_s \text{ to forward stream to, number of layers to forward to each } s' \in \Gamma_s, \text{ set of clients } C_{s'} \text{ each } s' \in \Gamma_s \text{ is responsible for } \}$
\State /* Update model based on feedback received since last model update. */
\State model $\leftarrow$ UpdateModel(model, feedback)
\State /* Compute action balancing exploitation and exploration */
\State $r \leftarrow$ sample random number uniformly in $[0, 1]$
\If {$r > 1 - \epsilon$}
\State action $\leftarrow$ EstimateBestAction(model)
\Else
\State action $\leftarrow$ RandomAction()
\EndIf
\State Execute action and send trigger_action to all downstream nodes $\Gamma_s \in \text{action}$
\end{algorithmic}
\end{algorithm}
For any candidate action \((a(s, \cdot), q(s, \cdot))\) where \(a, q\) are consistent with each other as noted above, we can estimate the reward incurred by the action as

\[
\sum_{c' \in C_a} \left( -\hat{d}(s, a(s, c'), c') + \alpha \frac{\min(q(s, a(s, c')), \hat{q}(a(s, c'), c'), q(c'))}{q(c')} + \alpha \frac{\min(q(s, c'), q(c'))}{q(c')} \right)
\]

following our reward model (§2.3). The \texttt{EstimateBestAction()} outputs the action with the highest estimated reward. We use an integer-quadratic-program (IQP) to compute the best action as follows:

\[
\max \sum_{c' \in C_a} \left( -\sum_{s' \in S} x(s', c')\hat{d}(s, s', c') + \sum_{s' \in S, s' \neq s} \alpha x(s', c') \frac{q(s, s', c')}{q(c')} + \alpha x(s, c') \frac{q(s, c')}{q(c')} \right)
\]

such that

\[
x(s', c') \in \{0, 1\} \quad \forall s' \in S, c' \in C_a
\]

\[
q(s, s') \in \mathbb{N} \quad \forall s' \in S, s' \neq s
\]

\[
q(s, s', c') \in \mathbb{N} \quad \forall s' \in S, s', c' \in C_a
\]

\[
q(s, c') \in \mathbb{N} \quad \forall c' \in C_a
\]

\[
q(s, s', c') = \min(q(s, s'), \hat{q}(s', c'), q(c')) \forall s' \in S, s', c' \in C_a
\]

\[
q(s, c') \leq q(c') \quad \forall c' \in C_a
\]

\[
\sum_{s' \in S} q(s, s') + \sum_{c' \in C_a} q(s, c') \leq \min(b_s, q_m(s)).
\]

Here, \(x(s', c')\) is an indicator variable that denotes whether client \(c'\) is mapped to SFU \(s\); \(q(s, s')\) denotes the number of layers \(s\) sends to a downstream neighbor \(s'\); \(q(s, s', c')\) is an estimate of the number of layers client \(c'\) receives if the stream for \(c'\) is sent through \(s\); \(q(s, c')\) is the number of layers \(s\) sends to client \(c'\) if \(s\) forms a direct connection to \(c'\). Eq. (7) says that if \(q(s, s')\) layers are sent by \(s\) to \(s'\) and \(s'\) is responsible for \(c'\), then the number of layers received by \(c'\) is at most \(q(s, s')\hat{q}(s', c')\) and at most \(q(c')\) (number of layers requested by \(c'\)). Similarly Eq. (8) says the number of layers sent by \(s\) directly to a client \(c'\) cannot exceed client \(c'\)'s demand \(q(c')\). Finally Eq. (9) says the total number of layers sent by \(s\) cannot exceed the bandwidth limit of \(s\). Note that despite the non-linear min function in Eq. (7), the constraint can be converted in to a linear constraint using auxiliary variables. In our experiments, we observe the above IQP finds the exact optimal solution for small to moderate sized problem instances with few tens of nodes (see §5). For larger networks, we run the above optimization with a time cutoff specified.

\textbf{Updating the model (UpdateModel() in Algorithm 1).} When a client or SFU is first included in the bipartite graph model \(G\), the latency and quality parameters are initialized to zero. Each time an action is played, the SFU updates its model using the feedback received from the clients as follows. If the stream for client \(c' \in C_a\) is routed through SFU \(s' \in S\), we have

\[
\hat{d}(s, s', c') = \hat{d}(s, s', c') + \eta (d(s, c') - \hat{d}(s, s', c'))
\]

\[
\hat{q}(s', c') = \hat{q}(s', c') + \eta' (q(c') - \hat{q}(s', c'))
\]

where \(\eta, \eta'\) are step size parameters.

\textbf{Increasing stability.} Depending on the depth of an SFU in the multicast tree, it can take several rounds to evaluate the true quality of the SFU’s actions. SFUs in each of those levels must perform sufficient exploration and exploitation rounds in their respective action spaces to determine an efficient topology for the subtree. The higher up SFU \(s\) is in the subtree, the greater could be the number of rounds necessary to assess the true quality of \(s\)’s action.

\section{Evaluation}

\subsection{Simulator Design}

We evaluate DecVi on a custom event-based simulator written in Python. Each packet (message) sent or received by a node in the network is an event. Events are scheduled to occur following the network’s stipulated link delays using a linked list of queues datastructure. Each node of the linked list is associated with a time stamp and an event queue. The nodes are ordered by the time stamp. An event is first scheduled by adding the event to the event queue and then evaluated when the event queue is being evaluated. The nodes of events are evaluated based on the order of the nodes, which is the chronological order.

\subsection{Experiment Setup}

We consider a network of clients and SFUs, with each node assumed to be located at a point on a 2D plane. The latency of sending a packet between any two nodes is set to be proportional to the Euclidean distance between the nodes. Every 5s, a client initiates a trigger action command which is propagated down its multicast tree trigerring actions at the SFUs. Each time when the algorithm is activated, it is recorded as one round.

\textbf{Baselines.} We compare DecVi’s QoE against the global optimum computed using integer programming. The global optimum is a centralized scheme that takes complete information about the SFUs and clients locations, SFU bandwidth limits and client QoE requirements as input to compute the best multicast trees. Constraining the distribution paths to be a tree introduces an exponential number of constraints in the integer program. Thus, we are able to compute the global optimum only for small networks.

We also consider a nearest-server baseline, in which the source client selects a server to which is closest to it, forming a star topology. Many current conferencing applications follow this policy.

\textbf{Conference configurations.} We consider various node location settings to illustrate the QoE provided by DecVi and its convergence behavior:

- \textit{(1) Structured node locations.} In this setting, we place clients and SFUs at chosen locations such that it is intuitively clear what the optimum should be. Specifically, we arrange the nodes as a tree with client 0 as the root, the SFUs as interior nodes and all other

\textsuperscript{2}The simulator will be made publicly available in the final version of the paper.
clients as leaves. We evaluate whether DecVi discovers this tree. The global optimum baseline also computes the planted trees as optimum.

(2) Random node locations. Next, we consider a setting where nodes are randomly placed on the 2D plane. As before we evaluate the small, medium and large conference sizes. The random node location setting is designed to mimic the distribution of nodes in real-world wide-area P2P networks.

(3) Real-world locations. To compare against popular centralized conferencing system today, we consider a setting where clients and SFUs are located at various cities around the world, with round-trip-times between cities obtained from a ping measurement dataset [2].

5.3 Results

DecVi’s QoS is comparable to the global optimum baseline in all the three scenarios. The time DecVi takes to converge depends on the size of setting and model hyper-parameters.

5.3.1 Structured Settings. Figure 2 shows the connections of structured settings. Figure 2(a) shows the connection given by the proposed model, Figure 2(b) shows the connection given by the global optimum baseline, respectively. The connections generated by DecVi closely match the optimal connections. The shade of edge color represents the number of layers actually sent through that streaming connection. Figure 3 shows the comparison of the latency and bandwidth of each client between DecVi and the baseline, given by CDF.

5.3.2 Connections under Different QoS Preferences. Figure 4 shows the different connection topology for the 7 SFUs, 11 clients structured settings under different QoS parameters (α = 0, α = 50). Figure 5 shows the different connection topology for the 10 SFUs, 50 clients structured settings when α = 0 and α = 50. A higher α means higher preference on bandwidth while a lower α indicates higher preference on latency. In both settings DecVi chooses to connect to receiver clients more directly with less intermediate SFUs involved when α = 0, while more layers are sent when α = 50, with more SFUs involved in the multicast tree.

5.3.3 Randomized Settings. Results of randomly generated settings for 50 clients are shown in Figure 6. DecVi achieves the global optimal in the small and medium settings, and gives a connection close to the global optimal in the large setting. Average latency of each receiver clients are shown in (c).

5.3.4 A Setting with Multiple Sources Streaming. To simulate a scenario where multiple users are talking at the same time, a setting is generated (Figure 7) with multiple sources sending streams. Two clients (client 0, client 6), are sending streams simultaneously, while all other clients being the receivers (including themselves). The color of the edges in Figure 7(a) represents the streams sent from different source clients. The performance of latency by each source client with respect to rounds is shown in 7(b).

5.3.5 Real World Video Conferencing Setting. A real-world video conferencing setting is shown in Table 1. The clients and data centers are located in the cities listed, which are geographically scattered on different continents so that a scheduling is needed to
Figure 6: A randomly generated setting with 10 SFUs and 50 clients. (a) Connection generated by DecVi. (b) Connection generated by baseline. (c) Average latency of DecVi with respect to rounds.

Figure 7: The multiple source streaming setting. (a) Connections generated by DecVi. (b) Latency with respect to rounds, the source is identified by gateway SFUs, which are SFU 1 and SFU 2.

Table 1: Location of each SFU and clients.

| Location  | Role          | Region    |
|-----------|---------------|-----------|
| Frankfurt | SFU/Data Center | Europe    |
| Hong Kong | SFU           | Asia      |
| San Jose  | SFU           | North America |
| Paris     | Client        | Europe    |
| Singapore | Client        | Asia      |
| Taipei    | Client        | Asia      |
| Seattle   | Client        | North America |

Table 2: Latency by DecVi and common conferencing application model baseline.

| Source | Receiver | Baseline’s Latency | DecVi’s Latency |
|--------|----------|--------------------|-----------------|
| Paris  | Seattle  | 154.65ms           | 169.03ms        |
|        | Singapore| 177.64ms           | 242.81ms        |
|        | Taipei   | 248.97ms           | 233.29ms        |
| Seattle| Paris    | 154.64ms           | 158.24ms        |
|        | Singapore| 312.20ms           | 183.86ms        |
|        | Taipei   | 383.53ms           | 174.34ms        |

We achieve a good QoE. Latency between every two nodes are acquired from global ping data [2]. The proposed model is compared against a centralized baseline model which is the model structure of currently widely used conferencing applications. For this baseline model, it is assumed to involve one data center which serves as the central node. For DecVi, we assume that for each client, there exists SFUs geographically close to it. Two clients, located in Seattle and Paris, are streaming at the same time, with all other participants being the receivers.

Table 2 shows the latency for each client to receive the streams from the two sources. The average latency for the baseline model is 238.61ms, while for DecVi it is 193.60ms.

6 CONCLUSION

We have presented DecVi a decentralized algorithm for efficient multicast tree construction in open, p2p video conferencing systems. Due to the trustless model in open, p2p systems, DecVi is a non-cooperative algorithm that makes actions purely based on its own past observations of the effects of past actions, without collaboration with media servers. The adaptive design followed by DecVi finds routing paths that are automatically tuned to the heterogeneity of media servers and clients in where they are located, their processing and bandwidth capabilities etc. without requiring any explicit manual input.

REFERENCES

[1] 2022. Elon Musk’s SpaceX bans Zoom over privacy concerns -memo. https://www.reuters.com/article/us-spacex-zoom-video-comm/elon-musks-spacex-bans-zoom-over-privacy-concerns-memo-idUSKBN21J71H.
[2] 2022. Global Ping Statistics. https://wondernetwork.com/pings.
[3] 2022. Huddle01. https://huddle01.com/.
[4] 2022. Zoom down: Video call app breaks in the middle of the working day. https://www.independent.co.uk/tech/zoom-down-video-call-app-b2106020.html.
[5] Emmanuelle Andre, Nicolas Le Breton, Augustin Lemesle, Ludovic Roux, and Alexandre Gouaillard. 2018. Comparative study of WebRTC open source SFUs for video conferencing. In 2018 Principles, Systems and Applications of IP Telecommunications (IP Telecom). IEEE, 1–8.
[6] Gonca Bakar, Riza Arda Kirmizioğlu, and A Murat Tekalp. 2018. Motion-based rate adaptation in WebRTC videoconferencing using scalable video coding. IEEE Transactions on Multimedia 21, 2 (2018), 429–441.
[7] Hyuneseok Chang, Matteo Varvello, Fang Hao, and Surit Mukherjee. 2021. Can you see me now? A measurement study of Zoom, Webex, and Meet. In Proceedings of the 21st ACM Internet Measurement Conference. 216–228.
[8] Boris Gostev. 2019. Efficient and scalable video conferences with selective forwarding units and webRTC. Ph. D. Dissertation. Université de Strasbourg.
[9] Chao Li, Xiao Zhao, and Yong Liu. 2011. Optimal bandwidth sharing in multirawarm multi-party p2p video-conferencing systems. IEEE/ACM Transactions On Networking 19, 6 (2011), 1704–1716.
[10] Kwok-Fai Ng, Man-Yan Ching, Yang Liu, Tao Cai, Li Li, and Wu Chou. 2014. A P2P-MCU approach to multi-party video conference with WebRTC. International Journal of Future Computer and Communication 3, 5 (2014), 319.
[11] Aleksandrs Slivkins et al. 2019. Introduction to multi-armed bandits. Foundations and Trends® in Machine Learning 12, 1-2 (2019), 1–286.