Client-Based Distributed Video Conferencing via WebRTC

Dominic Kern [1,*] and Matthias Teßmann [2]

[1] KURZ Digital Solutions GmbH & Co. KG, 90763 Fürth, Germany
[2] Nuremberg Institute of Technology Georg Simon Ohm, 90489 Nürnberg, Germany;
Email: matthias.tessmann@th-nuernberg.de (M.T.)
* Correspondence: dominic.kern@kurzdigital.com (D.K.)

***Abstract*—The most common video conferencing topologies are mesh and star topologies. The star topology requires a powerful server which leads to high costs. In the mesh topology, this is not the case, as each participant is directly connected to every other participant. However, due to the load caused by the numerous connections, the mesh topology is not suitable for larger video conferences. In this paper, we propose a video conferencing service that combines the advantages of the mesh and star topologies to enable larger video conferences without the need for powerful servers. This is achieved by distributing the video streams over the most powerful participants instead of a server. The resulting system achieves an improvement in video quality compared to a reference test in the mesh topology, which was determined based on the transmission rate and frame rate.**

***Keywords*—video conferencing, distributed, webRTC**

I. Introduction

Video conferencing services are often used for one-to-one conversations between two participants. In this scenario, a direct connection between them is the best way to perform the transmission. Video conferencing with significantly more than two participants is common, especially in the business environment. However, with the outbreak of the Corona pandemic, they also found their way into schools, universities, and private life.

WebRTC adds real-time communication capabilities to all web browsers. It allows video conferencing without the installation of an additional program [1]. The most common topologies used to build WebRTC conferences are the mesh and star topologies [2]. In the star topology, participants send their video streams to a server, that distributes them to all other participants [3]. One of the drawbacks of this topology is, however, that powerful servers must be deployed, which increases cost and limits scalability according to the power of the server infrastructure [4]. Also, this topology includes a single point of failure for the video conference [5].

In a mesh topology, each participant is directly connected to every other participant. Therefore, no powerful server is required. However, the scalability of the topology is also limited because the numerous connections with other participants put a heavy load on the bandwidth and CPU of each individual participant as the conference size increases. Therefore, this topology is not suitable for larger video conferences [6].

Nevertheless, to enable large videoconferences without a powerful server, another way must be found to distribute the video streams to the participants. A video conferencing service that distributes the video streams over the most capable participants could enable high performance as in the star topology at low infrastructure costs like mesh topology.

In this paper we propose a WebRTC service that does not require a powerful server infrastructure. This is achieved by distributing parts of the server's tasks among the most capable participants. To do so, two problems must be solved. On the one hand, a method is required to determine the performance of the individual participants. Additionally, an algorithm is needed that constructs the topology based on the performance of the participants. In this paper we describe a possible solution to both problems and show that a high-quality video conferencing service can be established without the need for a powerful infrastructure.

II. Related Work

In 2007, Horiuchi *et al*. [7] presented a method to implement a network for a scalable video conferencing system using a tree structure. The work includes concepts for a network tree mechanism, a tree reorganization mechanism, and a fault recovery mechanism. Their functionality has been demonstrated in simulations. Beyond simulation, however, no practical experiments have been conducted.

The work of Anitha and Rajkumar [8] describes the use of a tree structure. By using multilayer video transmission, they achieved high quality video transmission without disadvantages caused by participants with weak bandwidth for the rest of the conference. The signaling for setting up the videoconference network and an algorithm for selecting the root nodes are not discussed, though.

The work by Hamzaoui *et al*. [2] provides a conceptual design for a signaling topology for heterogeneous

Manuscript received August 27, 2022; revised October 21, 2022; accepted December 7, 2022; published March 14, 2023.

dynamically changing networks. In addition, the use of root nodes is proposed as a replacement for turn servers that would otherwise have to be provided externally.

A protocol that builds on WebRTC and enables distributed video conferencing is presented in Hallberg's work [9]. It incorporates a distributed algorithm for voice-activated quality control to reduce the computational and network resources used. A proof-of-concept web application is developed using the protocol and its performance is evaluated. The author notes that the maximum possible conference size is limited by the network capacity of the active speaker. As a solution, a distribution of the speaker's load among the other participants is proposed. It was not within the scope of the work, to implement this approach or to specify possible implementations in more detail.

In 2014, Grönberg and Meadows-Jönsson [10] also proposed a tree topology for use in video conferencing services. They implemented the service based on *appear.in*, a WebRTC video conferencing application that has been renamed to *whereby*[1] in the meantime. Based on their implementation, they show with experiments that the load on most nodes could be reduced, and the overall resolution of the image transmissions increased. With the reduced load on multiple nodes, a larger video conference can be conducted with the same quality as the reference implementation. However, their approach lacks the development of an adaptive method to build a tree. The selection of root nodes relies on manual selection by the developers.

III. Distributed Videoconferencing System

DCS was chosen as the abbreviation for the *Distributed Conference Service*. The basic functions are structured as follows. The DCS consists of a server and a web application that is running on the user's side. The user requests the web application from the server. A socket connection is established between the server and the web application to exchange messages. The server processes incoming messages itself or otherwise forwards them to the corresponding participant. A message protocol created for this purpose enables the correct assignment and processing of messages. The server is responsible for the mutual finding and coordination of the users. The WebRTC connections on the other hand are established directly between the participants. The transfer of image and sound data happens directly between the participants over the WebRTC connections. The necessary information for establishing the WebRTC connections is transmitted by messages that are forwarded by the server. The server is then no longer necessary for the operation of the connections.

A. Topology Adjustment Procedure

In order to enable distributed conferences with many participants low-performing participants must be relieved. This is achieved by strong participants forwarding the video streams of weak participants. Fig. 1 illustrates how the load can be relieved by forwarding. In part a) the initial situation is described. Five users are in a video conference using the mesh topology. User A must encode and send four video streams. Part b) shows the video conference when user B relieves user A by forwarding. Only one outgoing connection from user A to user B is necessary as the latter forwards the video stream to the other participants. The smaller number of outgoing transmissions reduces the workload for user B. Among all participants, those must be found who are able to perform this kind of stream forwarding. Participants who need the most support must be assigned to them.

The structure of the DCS is based on the following characteristics. A *supernode* is a powerful participant. It forwards video streams from one or more weak nodes to the others and thus supports them by reducing their workload. Participants that do not forward video streams are subsequently referred to as simple nodes. They are either supported by a *supernode* or participate as a neutral node that handle sending of their own video stream to all participants themselves.

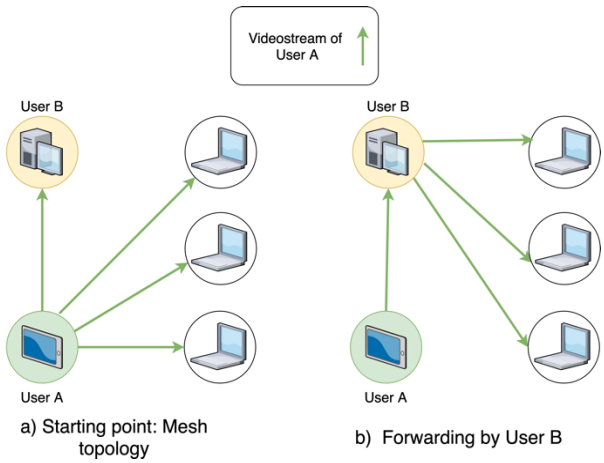

Figure 1. Comparison of the outgoing video streams of user A in an ordinary mesh topology against when forwarded by user B.

This depends on the availability of a *supernode* and their own workload. Each participant can take the role of a simple node or a *supernode* in a video conference.

The DCS must decide autonomously how the network should be structured. A mechanism is needed that determines which participant acts as a *supernode* and what tasks it must perform. An overview of the performance of all participants is required as a basis for decision-making. It is also necessary to record and keep up-to-date which tasks are currently being performed by a *supernode*. In the video conferencing solution presented in this paper, the network is controlled by the server.

The way the video conferencing service organizes its topology is as described below.

The participants determine their performance before joining the video conference and send it to the server in the form of a performance value. This value represents the maximum number of video streams that a participant can handle and is referred to as the connection score. How this score is calculated is described in further detail in Section III-C.

[1] https://whereby.com/

The Server has the task of maintaining the participant directory. This contains the connection score of each participant. Based on this data, the server calculates an optimized topology structure with the goal of enabling the largest possible video conference. It keeps a record of which video streams are forwarded in the participant directory. Then the server sends messages with instructions to all participants who are supposed to forward streams. Whenever the initial situation changes, the optimized structure of the topology is updated. This happens every time a user joins or leaves the conference.

B. Network Organization

The algorithm developed for network organization runs on the server and computes an optimized structure of the required topology of the network. The server uses the participant directory as the source of data. The aim of the procedure is to restructure the topology in such a way that participants with a low score are relieved from network load. In this way, the goal of making the video conferencing service suitable for larger conferences can be achieved. To achieve this goal, high-scoring participants are instructed to forward the video streams of low-scoring participants.

The procedure can be described as follows. The initial situation of the video conference is a mesh topology. Each participant is directly connected to every other participant. This setup is maintained for as long as possible. In this situation, the network latency is particularly low.

After more and more participants join, the video conference becomes too large and the network connection or the CPU performance of a participant is no longer sufficient to handle the number of connections. Then the participant should be relieved to keep the conference running. For this purpose, the algorithm must assign the appropriate forwardings. The process is as follows. As soon as a participant is identified as running out of network bandwidth and/or processing power and thus requires forwarding by other participants, it is completely assigned to a *supernode*. This makes the participant a child node of that *supernode*. The *supernode* is then responsible for forwarding *all of* the child nodes video streams. The node that has the most capacity left is always selected as the *supernode*. An allocation only takes place under the condition that the *supernode* itself is not running out of bandwidth or processing power by this operation.

To determine if a low performing participant is running out of power and how many forwardings a high performing participant can handle the remaining capacity is calculated. This is done by the server based on the participant directory which includes the connection score per participant. This score represents the total number of WebRTC connections with transmission in both directions that a participant can handle. The remaining capacity is the number of outgoing transmissions that a subscriber can additionally handle in the current state of the topology.

Just the outgoing transmissions are used, as only the number of these are relevant for the algorithm. The number of incoming connections that must be handled by the participants remains unaffected by the restructuring of the topology. At any given time, it corresponds to the number of other conferees as one participant must receive the video stream of all other participants. The difference with outgoing transmissions is that with the help of a supernode only one transmission is needed regardless of the number of other participants. The number of outgoing connections increases for supernodes with each transmission they forward and decreases at the same rate for their child nodes. Therefore, only the outgoing capacity is relevant for the number of additional forwardings that a supernode can handle.

The remaining capacity is calculated as follows. The burden of a participant in the current state of the topology is composed from the *meshBurden* and the *relayBurden*. The *meshBurden* is the number of times the participant must send its own video stream and equals the number of other participants in the conference:

$$meshBurden \leftarrow conferenceSize - 1\,. \tag{1}$$

If the participant is forwarded by a *supernode*, the *meshBurden* is 1. This is the case because the participant must send his video stream only to the *supernode* and not to all other participants.

The *relayBurden* is applicable to *supernodes* and equals the number of nodes the *supernode* is forwarding to. The burden of forwarding all video streams of one participant therefore is

$$relayBurden \leftarrow conferenceSize - 2\,. \tag{2}$$

This is because the video stream must be forwarded to everyone except the *supernode* and the source.

The two values are then summed and subtracted from the score

$$capacity \leftarrow score - meshBurden - relayBurden\,, \tag{3}$$

in order to get the remaining capacity.

The remaining capacity of each participant can then be used to assign the weak participants to the stronger ones in order to optimize the topology. Algorithm 1 shows the implementation.

Algorithm 1 Pseudocode showing the necessary steps for optimising the topology.

```
1:  procedure CALCULATEOPTIMIZEDTOPOLOGY(clientRegister)
2:      while true do
3:          remainingCapacityDict ← CALCREMAININGCAPACITYDICT(clientRegister)
4:          weakest ← minCapacityNode(remainingCapacityDict)
5:          strongest ← maxCapacityNode(remainingCapacityDict)
6:          if weakest.value < 0 and strongest.value ≥ (conferenceSize − 2) then
7:              clientRegister[strongest.id].relayedNodes.add(weakest.id)
8:          else
9:              break
10:         end if
11:     end while
12:     return clientRegister
13: end procedure
```

The *CalculateOptimizedTopology* function is passed the *clientRegister*. It makes changes to it and returns it. These changes are the optimizations to the topology in the form of the selection of *supernodes* and their associated children.

In Line 2 of Algorithm 1, the main loop of the function starts. In Line 3, the remaining capacity of all participants is calculated and saved in a dictionary. Then *minCapacityNode* and *maxCapacityNode* are used to determine the two participants with the highest and lowest remaining capacity. They are stored in the two variables *weakest* and *strongest* from where these can be easily accessed.

The condition in Lines 6–10 expresses the following ideas. The first check *weakest.value* < *0* determines whether the weakest node is running out of capacity. If its remaining capacity is less than 0, it needs support. The second check ensures that the strongest node has enough remaining capacity to support the weakest node. Therefore, its capacity must correspond to at least the *conference size* – 2. If both conditions are met, the entry of the strongest node is changed, and the weakest node is added to the list of child nodes of the strongest node. Then the whole loop is executed again.

When the loop is executed again, the remaining capacity is recalculated for each participant while the changes to the topology from previous runs are considered. Another node will now take the position of the weakest node as their remaining capacity has improved due to the support. The position of the strongest node changes depending on which node has the sparest capacity under the new circumstances.

The loop terminates as soon as either the weakest node is no longer overwhelmed, or the strongest node no longer has enough capacity for further support. The adjusted *clientRegister* is then returned. The server finally begins to communicate the topology changes to the network by issuing instructions to the participants.

C. Determination of the Connection Score

The developed software performs a restructuring of the topology with the goal of relieving the low-performing participants at the expense of the strong participants. For this purpose, it is essential to be able to determine how powerful a participant is. This cannot be easily determined by a web application running in a browser.

Therefore, the web application performs a self-test. The result is the connection score of a participant and this score is sent to the server after joining a conference.

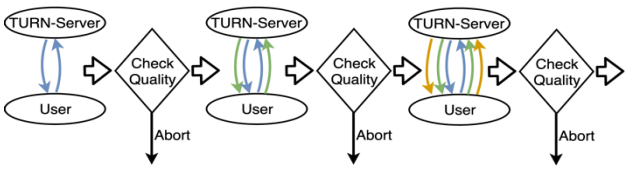

Figure 2. Procedure of the self-check to determine the score. The user establishes WebRTC connections to himself until their transmission quality falls below a certain level.

The principle behind the developed self-check is shown in Fig. 2. The client web-application establishes multiple WebRTC connections to itself relayed over a TURN-server. It sends a video stream over each of these connections. The streams pass through the complete WebRTC media pipeline on the users device. After establishing a WebRTC connection the client waits a predefined time interval of two seconds before the next connection is initiated. After a certain, system specific number of simultaneous transmissions, the system runs into an overloaded state. This becomes noticeable as the quality of the transmissions degrade. To detect the overload of the system, the status values of the WebRTC connections are collected (see below). This is achieved through the WebRTC statistics API [11]. The self-check is aborted if the quality of the transmission is considered insufficient for a videoconference. How this is determined based on the collected data is described below. The number of connections established until the abort is the resulting connection score. It corresponds to the maximum number of transmissions a participant is capable of handling.

The self-check is performed only once before the start of the video conference. The value determined remains for the entire conference. The reason for this is this purposeful overload would lead to a severe degradation of quality in the running conference. In addition, a video conference running in parallel would also falsify the measurement.

The overload of the host system is detected based on status values of the WebRTC connections. The following three values are calculated from these and were chosen as indicators for the transmission quality.

Frame rate refers to the number of frames that are played back per time span. This indicator is read on the receiver side to measure how smoothly the video stream appears to the user during playback. If it falls below the threshold of 10 fps, the transmission is disturbed and the video stutters. In this case, the quality is considered insufficient.

Delay is composed of several values and here describes the delay over the entire path from the recording to the playback of a video frame. In addition to the transmission time via the network, this also includes the processing time on the transmitter and receiver side. Among other things, this includes the duration of encoding, decoding, and the delay due to jitter buffer until a video frame is played. If the delay exceeds a threshold of 250ms, bidirectional video communication is considered unfeasible.

The transmission rate is taken on the receiver side and measures the amount of data transmitted over a certain period of time. It drops when the internet connection is overloaded. If its value falls below 350 Kbit/s, the quality of the video stream is considered insufficient.

The thresholds used were determined empirically with the help of three colleagues. They were asked to watch the video streams of an ongoing self-check and to determine the point at which they perceived the video quality to be insufficient for a conversation. The value of the indicators at this point were used as a baseline. The test was repeated under different conditions. On each run, the thresholds were manually adjusted to match the perception of the video until the measured score matched the subjective rating.

Using this approach, a threshold could be defined for each of the indicators individually. As soon as one of these values falls below the threshold, the quality is considered being below an acceptable level. The system has run out of resources to handle more connections and the self-check is aborted.

IV. Experiments

This section presents the results of the conducted experiments. They are intended to answer to what extent the objective of the work could be fulfilled. For this purpose, DCS and its components are tested to make statements about a video conferencing service with the described concept. The DCS was examined regarding the following aspects

- Does the developed self-check deliver meaningful results?
- Do the forwarding actions initiated by the system lead to an improvement?

A. Self-Check on Different Bandwidths

The score must be reasonably close to the number of possible connections that the participant is capable of. If the self-check does not deliver appropriate results, no meaningful optimizations of the topology can be made. To find out to what extent the determined score corresponds to the performance of the system, the self-check was performed under different conditions and the results were compared.

In the first step, test runs were performed under different network conditions. For this purpose, the bandwidth of a test device was limited by the "Network Link Conditioner" software tool to simulate a poor Internet connection. The test series was intended to show how the self-check reacts to different network conditions and how this affects the calculated score. As TURN-server the open-source project *Coturn*[2] was installed on a rented virtual server.

A MacBook Pro 15"2017 was used as the test device to run the self-check. The bandwidth of the test device was limited to 3 Mbps, 5 Mbps and 10 Mbps. For each of these settings, the self-check was run six times. Two minutes wait time between each run allowed the temperature of the CPU to stabilize and prevented the result from being affected by heat-induced CPU throttling. Only the upload bandwidth was limited. Since the videos are transferred to a turn server and back during the self-check, the required up- and download bandwidth are the same. Therefore, only the lower of the two is relevant and usually most residential internet connections have more download than upload bandwidth.

The scores calculated during the self-checks are listed in Table I. From the data, it is evident that there is a direct proportionality between the available bandwidth and the measured score. A video transmission of the DCS requires 500 kbit/s bandwidth. An upload bandwidth of 3 Mbit/s should therefore be sufficient for six connections. For this bandwidth, the self-check resulted in an average score of 6, which is in line with expectations under these conditions. A bandwidth of 5 Mbit/s should be sufficient for ten transmissions. Under these conditions, the average score was 9.8 and is thus close to the theoretical estimate of performance. A bandwidth of 10 Mbit/s should be sufficient for 20 transmissions. The average score in this test series was 18.3. The determined scores only deviate slightly from each other within the test series.

TABLE I. Scores Calculated by the Self-Check at Different Upload Bandwidths

Bandwidth	Test series						Average
3 Mbit/s	5	6	7	6	6	6	6.0
5 Mbit/s	10	10	9	12	9	9	9.8
10 Mbit/s	19	18	16	20	18	19	18.3

The experiment shows that the self-check can reliably determine the performance of the Internet connection under the testing conditions. The determined score is very close to the number of transmissions that should be possible with the respective bandwidth. Therefore, it is plausible that the determined score corresponds to the number of transmissions. The small deviation between the scores measured at the same bandwidth shows that the results of the self-check are reliable.

B. Self-Check on Different Devices

In addition to the tests under different network conditions, the self-check was also run on different devices. This is to determine whether the self-check can detect the performance of the hardware.

The conditions for this test were almost the same as for the previous one. However, redirection of video transmissions via the turn server was disabled. The reason for this was the limited upload bandwidth at the test site. This would have meant an upper limit for the result of the self-check testing different hardware. Therefore, this limit was removed for this test.

The following devices were used for the test. A Lenovo Yoga 720 convertible with Intel Core i5-8250U processor and 15W TDP as a laptop with lower performance. A MacBook Pro 15"2017 with i7 and 45W TDP as a laptop with higher performance. Lastly, a desktop computer with Ryzen 5 1600 and 60W TDP as the most powerful device.

TABLE II. Scores Calculated by the Self-Check with Different Hardware

Device	Test series						Average
Convertible	6	25	24	24	21	23	20.5
MacBook	37	41	37	38	40	38	38.5
Desktop	64	63	62	60	64	63	62.7

Table II lists the scores determined for the individual devices during the self-checks. The average score achieved by the devices over the measurement series differs noticeably between the devices. The smallest average score was achieved by the convertible with a value of 20.5. The MacBook achieved a higher average score of 38.5. The desktop computer, as the most powerful test device, achieved the highest average score

[2] https://github.com/coturn

of 62.7. The determined scores also only deviate slightly within the test series. The more powerful devices received a higher score than the weaker ones which is according to our expectations. This indicates that the self-check takes the processing performance of the devices into account. The small deviation between different test series indicate that the score can be determined reliably.

C. Videoconferencing with Six Participants (no DCS)

Within the scope of this paper, the DCS was evaluated in a video conference with six participants. The bandwidth of one participant was limited to simulate a weak internet connection. The test was performed with the full functionality of DCS as well as with deactivated load redistribution as a reference test. Status values of all WebRTC transmissions were recorded, evaluated, and compared.

A MacBook Pro 15"2017, a desktop computer and four Lenovo Yoga 720 convertibles were used as test devices. The upload bandwidth of the MacBook was limited to 1500 kbit/s using the software tool "Network Link Conditioner". The network connection of the other test devices remained unaffected. The test devices joined the video conference at intervals of 20 seconds. The test setup is shown in Fig. 3.

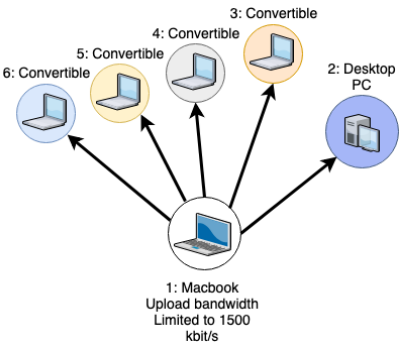

Figure 3. Experimental setup of the reference test in the mesh topology. The six participants join the video conference in the order noted. The first device's transmissions to the others, are shown as black arrows.

First, the reference test was conducted. Therefore, the functions for relieving weak participants were deactivated. As a result, the video conference takes place in the mesh topology.

The transmission rate at which the other participants receive the video stream from participant 1 is shown over time in Fig. 4. The time points at which participants joined are inserted as vertical lines. The first transmission starts when user 2 joins. The line shown in dark blue indicates the transmission rate at which he receives the video stream from participant 1. The transmission rate is 500 kbit/s until the next participant joins. When user 3 joins, the transmission rate of the two participants initially drops, but stabilizes again and rises back to the previous 500 kbit/s. The transmission rate of user 2 drops to 500 kbit/s. After user 4 joins, the transmission rate of all participants fluctuates increasingly and continues to decrease as more users join the conference. The graph also shows that the transfer rates differ greatly between the participants. While the transfer rate of user 5 increases up to 500 kbit/s for a short time directly after joining, the transfer rate of participant 1 is below 100 kbit/s at this time.

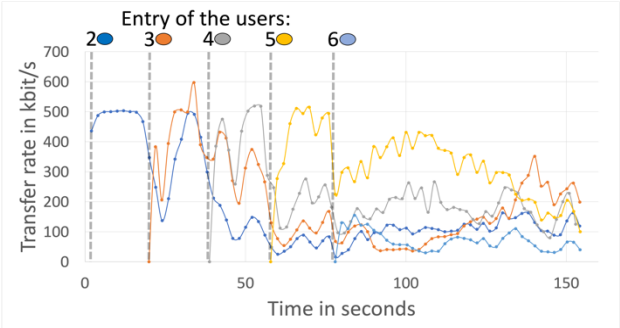

Figure 4. The transmission rate at which user 1 transmits his video stream to the other participants in the reference test. The users join the video conference one by one.

The trend of the transmission rate during this test run is also shown by Table III. In this table, the average transmission rate of all users in the respective phases is listed. It is noticeable that the average transmission rate decreases steadily as the number of participants increases. While the average transmission rate for two participants was 488 kbit/s at the beginning of the conference, it is only 157 kbit/s after six participants joined.

TABLE III: THE AVERAGE TRANSMISSION RATE IN THE REFERENCE TEST FOR THE RESPECTIVE CONFERENCE SIZE

Number of participants	2	3	4	5	6
Average transmission rate on kbit/s	488	379	291	196	157

Fig. 5 shows the frame rate at which user 1's transmissions are played back on the other devices. At the beginning with only two participants, this is constant at 30 frames per second. As the number of participants increases, the number of dropouts also increases.

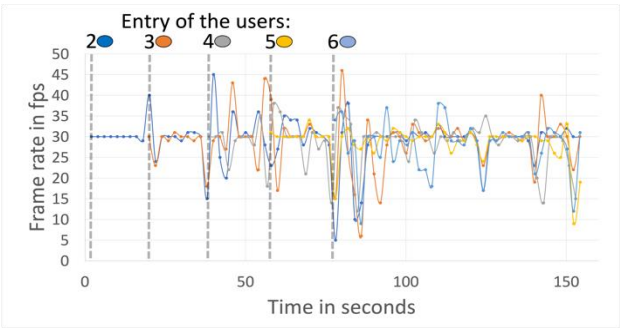

Figure 5. The frame rate over time of the videoconference in the reference test. The fluctuations increase as the conference size increases.

D. Videoconferencing Using DCS with Six Participants

After the reference test, the load distribution functions were reactivated. The test was repeated with the same test setup to compare the measured values. The participants performed the self-check automatically prior to participation as they would under real conditions. The resulting scores are shown in Fig. 6.

In the specified joining order, the course of the experiment is as follows. With its score of 3, user 1 is overloaded as soon as there are more than three other participants in the conference. This is the case when user 5 joins the conference. From that moment on, his video stream is forwarded by user 2 as a *supernode*.

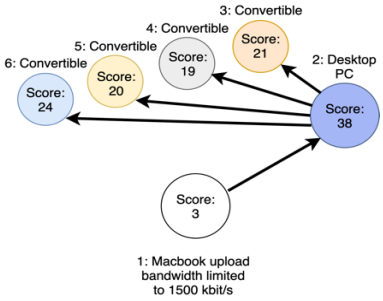

Figure 6. Test with load balancing enabled. The test devices are shown with the scores they calculated.

This forwarding is visualized by the black arrows in the figure. Before user 5 joins the conference, the topology does not differ between the two test runs performed.

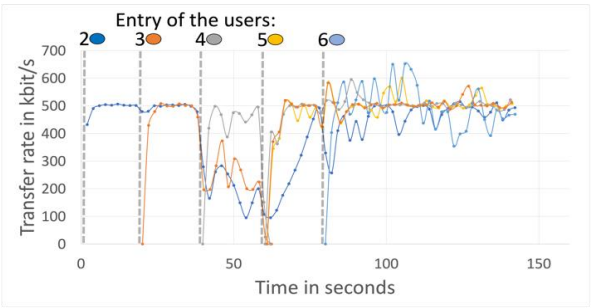

Figure 7. The transmission rate at which user 1 transmits its video stream to the other participants. When user 5 joins, the forwarding takes effect and the transmission rates increase again.

The transmission rate at which the other participants receive the video stream from participant 1 is shown in Fig. 7. Until user 5 joins, the course here is similar to the reference test. In both cases, when the fourth participant joins, the transmission rate for users 2 and 3 drops significantly. With the joining of user 5 the forwarding becomes active. From here on, the topology differs between the two trials. As visible in the figure, the transmission rates of user 3 and user 4 initially drop to zero. Direct transmissions have been terminated at this point. They are replaced by new connections via the *supernode*. The graph shows the transmission rate of the new connections from this point on. After the switch, the graph shows that the transfer rates of all users increase sharply. Even after user 6 joins, the transmission rate remains stable - close to the maximum transmission rate per transmission of 500 kbit/s.

The effect that forwarding has on the transmission rate can also be seen in the average transmission rate in Table IV. Similar to the situation during the reference test, the average transmission rate initially decreases as the number of participants increases. When the fifth participant joined and forwarding started, the results clearly differed. While the transfer rate in the reference test continues to drop from participant 5 and is only around 196 kbit/s for 5 participants, it increases from about 286 to 417 kbit/s when load balancing is enabled. The difference is even greater when the full conference size of six participants is reached. The average transmission rate in the reference test is about 157 kbit/s while when using DCS the rate increases to about 495 kbit/s.

TABLE IV: THE AVERAGE TRANSMISSION RATE IN THE TEST WITH ACTIVATED LOAD DISTRIBUTION

Number of participants	2	3	4	5	6
Average transmission rate on kbit/s	492	491	286	417	495

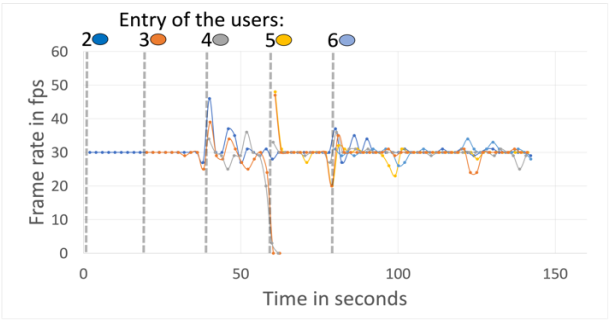

Figure 8. The frame rate over time during the video conference. After forwarding from user 5 joining, the frame rate is more stable.

If we look at the frame rate in Fig. 8, we see that there are significantly smaller fluctuations here. Like the transmission rate, the fluctuations increase from the 4th participant on. When the fifth participant joins, the change takes place and the frame rate shows less fluctuation from this point on than before. Compared with the reference test, the dips in frame rate from the fifth participant onward are smaller and less frequent.

The results demonstrate that the DCS can perform topology changes based on the state of the current video conference. In the given experimental setup, it reacts correctly. The user with limited bandwidth was identified and his video streams are automatically forwarded by the best performing participant in the conference.

From the gathered data, it can be concluded that the performed forwarding has improved the quality of the video transmission. The transmission rate is significantly higher with activated forwarding than in the reference test with the same conference size. If only little bandwidth is available, WebRTC automatically reduces the transmission rate at the expense of image quality. For this purpose, the resolution is reduced or the video stream is encoded in lower quality. A higher transmission rate can therefore be assumed to also yield a higher image quality.

In addition to the transmission rate, the frame rate shows that the quality is improved by the forwarding. Fig. 5 shows that the frame rate in the reference test is subject to increasingly frequent fluctuations as the conference size increases. If the frame rate is irregular and drops frequently, stuttering is to be expected. In such a case, it can be assumed that this lowers the perceived

quality of the video transmission. Fig. 8 shows that in the test with activated load balancing, the dips in frame rate become less with activation of forwarding. With a more constant frame rate with fewer dips, a smoother display can be assumed.

V. Conclusion

In this work, we proposed a WebRTC based videoconferencing service that autonomously optimizes the network topology named DCS. The goal of DCS was to enable larger video conferences without maintaining a powerful server infrastructure. This was implemented by identifying the most powerful participants in the video conference in terms of network bandwidth and computing power in order to support the weakest participants by forwarding and distributing their video streams. The paper presented the concepts on which DCS is based. These included how the videoconferencing service is organized, how the performance of the participants is determined, and how, on this basis, it is decided what changes to the network topology should be made to find the optimal solution for the distribution of streams. Subsequently, a series of experiments were carried out to determine the applicability of the system for the desired purpose. The experiments showed that DCS can make changes to the network topology based on the information about the current videoconference. The changes resulted in an improvement in video quality, as determined by transmission rate and frame rate. As an important precondition, it was shown that the performance of the participants could be reliably determined. In experiments with different bandwidth and hardware, the self-check yielded plausible results. The number of participants in the experiments was limited to 6. The optimization of the topology should also be effective for larger videoconferences, as long as there are enough strong participants to support the weak ones. From the perspective of a weak participant that gets support, the performance should be comparable to that of a star topology. An overload of the download bandwidth in large conferences could become problematic, however, but this problem would also apply to star topologies with a server carrying the load. As further work we plan to evaluate our software on a larger number of devices under varying network conditions, including mobile networks, in order to undermine our findings. Overall, it could be shown that our implementation of a client-based distributed conferencing service, DCS, is a valuable tool for the provision of conferencing services without the need for an expensive server infrastructure.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

Dominic Kern conducted the primary research of the subject, provided the implementation, and has written the major parts of the paper. Prof. Teßmann supervised the research project and contributed text, reviewed, and edited parts of the paper.; all authors had approved the final version.

References

[1] A. B. Johnston and D. C. Burnett, "APIs and RTCWEB protocols of the HTML5 real-time web," in *Digital Codex LLC*, St. Louis, MO, USA, 2012.

[2] A. Hamzaoui, H. Bensaid, and A. En-Nouaary. "A new signaling topology for multiparty web real-time video conference networks," in *Proc. SITA'18. Rabat, Morocco: Association for Computing Machinery*, 2018.

[3] M. Wenzel and C. Meinel, "Full-body WebRTC video conferencing in a web-based real-time collaboration system," in *Proc. IEEE 20th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, 2016, pp. 334–339.

[4] A. A. Lozano, V. Singh, and J. Ott, "Performance analysis of topologies for Web-based Real-Time Communication (WebRTC)," M.S. thesis, School of Electrical Engineering, Espoo, Finland, 2013.

[5] F. D. López-Fuentes, "Video multicast in peer-to-peer networks," Dr. Eng. dissertation, Technical University of Munich, 2009.

[6] B. Grozev, G. Politis, E. Ivov, T. Noel, and V. Singh, "Experimental evaluation of simulcast for WebRTC," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 52–59, 2017.

[7] H. Horiuchi, N. Wakamiya, and M. Murata, "A network construction method for a scalable P2P video conferencing system," in *Proc. the Third IASTED European Conference on Internet and Multimedia Systems and Applications, EurolMSA '07*, ACTA Press, 2007, pp. 196–201.

[8] M. Anitha and K. Rajkumar, "P2P multipoint video conferencing using layered video and multi-tree structure," *International Journal of Engineering and Technology*, vol. 5, pp. 857–861, April 2013.

[9] A. Hallberg, "A protocol for decentralized video conferencing with WebRTC: Solving the scalability problems of conferencing services for the web," Ph.D. dissertation, KTH Royal Institute of Technology, Stockholm, 2016.

[10] J. Gronberg and E. Meadows-Jönsson, "Tree topology networks in WebRTC: An investigation into the feasibility of supernodes in WebRTC video conferencing," M.S. thesis, Chalmers Univ., Göteborg, Sweden, May 2014.

[11] H. Alvestrand, B. A. Singh, and H. Boström. (2022). Identifiers for WebRTC's statistics API. W3C Candidate Recommendation Draft. [Online]. Available: https://www.w3.org/TR/2022/CRD-webrtc-stats-20220517

Dominic Kern was born in Erlangen, Germany, in 1996. He received the B.S. degree from the Nuremberg Institute of Technology Georg Simon Ohm, in 2019 and the M.S. degree from the same Institute in 2022, both in computer science. Currently he is working as a software developer for the company Kurzdigital.

Matthias Teßmann is a professor of computer science at the Nuremberg Institute of Technology. His research interests include, amongst others, web application development and real-time audio and video communication over IP networks.