A P2P-MCU Approach to Multi-Party Video Conference with WebRTC

Kwok-Fai Ng, Man -Yan Ching, Yang Liu, Tao Cai, Li Li, and Wu Chou

***Abstract*—WebRTC enables web browsers with real-time communications capabilities via JavaScript APIs. But when the number of the participants increases, the bandwidth and CPU requirements have become a serious issue in a push based mesh network. In this paper, we propose a P2P-MCU approach for multi-party video conferencing that efficiently supports both ordinary smart mobile phones and PCs. In our approach, a MCU module is integrated into the browser to mix and transcode the video & audio streams in real time. And when the browser acts as the MCU node leaves the conference session without notice, another candidate browser can take over the control immediately, and the ongoing WebRTC conference can be seamlessly recovered with our MCU selection algorithm. In addition, our approach works under the 3G symmetric NAT networks by using some UDP hole punching method. Our P2P-MCU solution reduces 64% CPU usages and 35% bandwidth consumptions for each participant compared to the mesh-network solution in our eight-party WebRTC conference experiments. Although the P2P-MCU module may introduce some delay (<500ms), the delay is stable and perceptually almost neglectable.**

***Index Terms*—MCU, P2P, video conference, WebRTC.**

I. Introduction

Driven by the widespread fixed and mobile broadband networks, there is a trend to have real time multi-party video conferences at any time/place. To meet the emerging requirements, WebRTC [1] (Web Real-Time Communications) received a great interest since the API is inherently supported by many new versions of popular browsers, i.e. Google Chrome and Mozilla Firefox. However, since WebRTC is initially designed for browser to browser communication, even for a small scale group, the multi-party conference model may be either complicated or expensive.

In particular, to support N conference participants with a pure Mesh network, there will be N*(N-1)/2 links. The bandwidth/device capability requirements will increase quadratically to the number of the participants in the conference. Accordingly, a MCU [2] (Multi point Control Unit) server is introduced to reduce bandwidth consumption by mixing the media received from users in the conference into a single stream to each participant. However, MCU server, typically based on a fixed and pre-configured hardware, is often quite costly and it consumes significant amount of bandwidth.

Manuscript received January 25, 2014; revised March 23, 2014.

Kwokfai Ng, Manyan Ching, and Yang Liu are with Hong Kong Applied Science and Technology Research Institute Company Limited, HK (e-mail: micng@astri.org, rachelc@astri.org, yangliu@astri.org).

Tao Cai, Li Li, and Wu Chou are with Huawei Technologies Co., Ltd, Shenzhen, China (e-mail: t.cai@huawei.com, Li.NJ.Li@huawei.com, Wu.Chou@huawei.com).

In this paper, we describe our approach to peer-to-peer MCU (P2P-MCU) that tackles the abovementioned issues. Moreover, in our approach, the MCU is integrated in a browser at the client side, and this specific client is called MCU host. Accordingly, the media flows in the conference run in a P2P manner between the MCU host and web browsers. The proposed approach is implemented and we demonstrate the web applications that we developed for an eight party WebRTC conferencing including mobile clients.

The contributions of this paper are: firstly, we design a P2P-MCU architecture working with current WebRTC protocols; secondly, we propose a MCU host determination strategy to dynamically and optimally place the MCU host at the web browsers; finally, we implement an efficient UDP punching [3] method for mobile users behind firewall/NAT to participate in the conference. Experiments with a mixture of mobile and PC users under various configurations are conducted. The results indicate that our approach is feasible and efficient for multi-party conferencing through WebRTC.

The rest of this paper is organized as follows. Section II presents our P2P-MCU approach to multi-party conference application. Section III illustrates various communication models with our P2P-MCU approach. In Section IV and Section V, we study the strategy for MCU host determination and UDP punching to support mobile clients. Section VI, the implementation details and experimental results are presented. Section VII summarizes the work and concludes with some future directions.

II. Deployment Multiple User Conference Application

A. General View

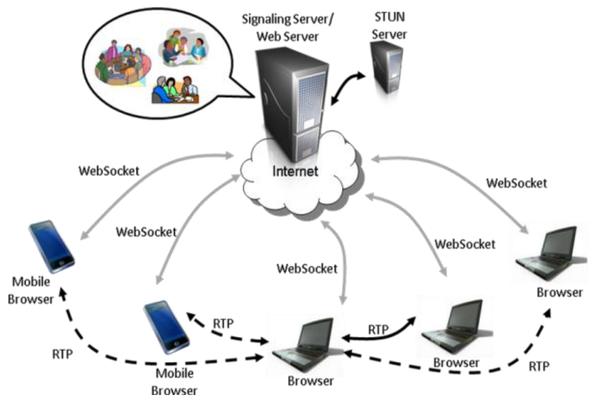


Fig. 1. Centralized server for video chat.

WebRTC browsers that want to join a conference can be connected in a variety of ways. For simplicity, we use a centralized web server to handle signaling between WebRTC capable browsers as illustrated in Fig. 1. The connections between signaling server and browser are based on WebSocket [4]. However, XHR/JSONP Polling can replace WebSocket connection in restricted environment such as 3G environment. Business logics are programmed in JavaScript.

In a full mesh conference, peer connections are established between each pairs of participants in the conference room. In our proposed solution, one participant only has to establish a connection to the MCU Host which maintains the MCU session. Consequently, it is not necessary for the remaining participant browsers to run any MCU. The maximum number of participants is subject to the processing power of MCU host.

B. Signaling Server

In this section, we summarize the functionality and characteristics of signaling server used by multiple user video chat. A signaling server provides administration of individual conference room. A conference room has a participant list. A participant can create a new room, join a room and leave a room. Every time the participant presence is changed, all the participants in the room will be notified by message.

Because both browsers may be behind NAT, they may not be able to communicate directly. A signaling server works by routing messages between participants in a conference room. The examples of such messages are browser capabilities, type of NAT it has, video frame rate preference, SDP [5] (Session Description Protocol) message, WebRTC ICE [6] candidates and so on. Each message includes unique identity of participant. Participant can send private to each other or group messages to all participants in the same room.

C. Peer-to-Peer MCU

In our proposed approach, we introduce P2P-MCU. P2P-MCU is the implementation of a distributed MCU using a peer-to-peer (P2P) architecture. P2P eliminates centralized media servers and the complex infrastructure investments. Unlike traditional video conferencing server, MCU host runs in a desktop browser which is normally behind a NAT. To overcome NAT connectivity restrictions, P2P-MCU module uses Session Traversal Utilities for NAT (STUN) [7] to discover correct public address that NAT allocates for UDP traffic between the local and external hosts.

MCU provides encoding/decoding of individual RTP [8] streams and mixing of RTP streams. MCU assigns audio/video encoder and decoder which are capable to decode RTP streams from the participant.

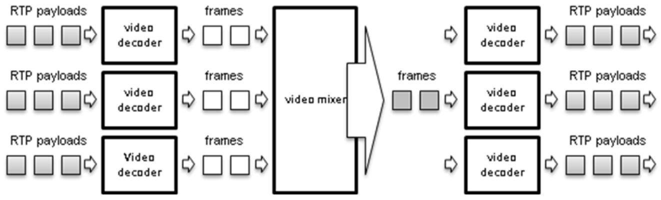


Fig. 2. Mixing video streams into a video stream.

In video mixing, all participants would see the same mixed video. The process starts with video decoding which decodes RTP payloads into video frames. Then, a video mixer mixes all video frames from different participants into one stream of video frames as illustrated in Fig. 2. Finally, individual video encoder encodes mixed video frames back into RTP payloads for each participant.

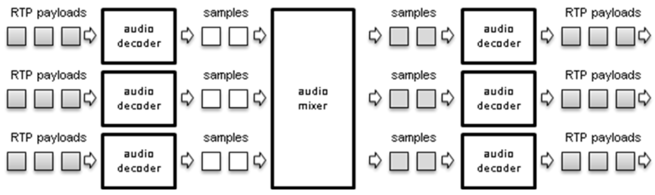


Fig. 3. Mixing audio streams into individual audio streams.

Mixing all participant audios into one stream would result in voice echoing back. To avoid this, each participant would have his own mixed audio stream which does not contain his own audio. First, audio decoders decode RTP payloads into audio samples. Then, an audio mixer mixes audio samples into individual streams of audio samples as illustrated in Fig. 3. Finally, individual audio encoder encodes corresponding audio samples back into RTP payloads.

D. Brower Integration

In this section, we summarize integration of MCU into the WebRTC architecture to enable rapid development and deployment of video conference application via JavaScript API. In our proposed solution, we expose P2P-MCU and STUN client library ability to JavaScript for easy manipulation by web applications (Fig. 4). They are MCU session controller and NAT detector.

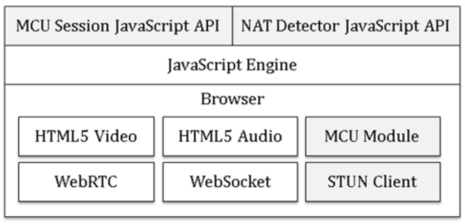


Fig. 4. Architecture of modified browser.

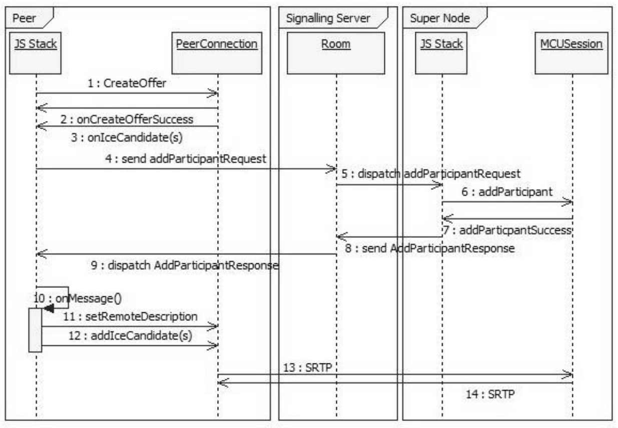


Fig. 5. Example of communication between MCU and participant via a signaling server.

MCU session controller is exposed as JavaScript API library to facilitate extension and modification of application logics. It controls the P2P-MCU module and encapsulates functionality of MCU conference room, include creating a conference room, adding participant to room and removing participant from the room.

It works by processing SDP offer and answer messages generated by and exchanged between WebRTC browsers via a signaling server as illustrated in Fig. 5.

NAT detector encapsulates functionality of the STUN client and exposed as JavaScript API library. A STUN client discovers the presence of a NAT and detects the type of NAT behind which the browser is hosting.

WebRTC Peer Connection object use ICE mechanisms to traverse layered NAT devices between the web browsers. If a direct connection is impossible because the browsers are behind symmetric NATs, traffic is routed via a TURN [9] relay server as a fallback, which may increase latency. By determining the type of NAT, we can guarantee MCU session not hosting behind symmetric NAT, and eliminate the involvement of TURN relay.

III. Communications

A. One-to-One Video Chat

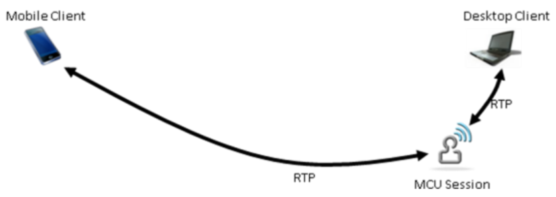


Fig. 6. Scenario involving one desktop client and mobile client.

In our proposed model, we assume a conference room has at least one browser that is integrated with MCU and is not behind symmetric NAT. We develop logic responsible to select the most appropriate participant to host a MCU session. The MCU host invites all participants in the room to establish peer connections to the MCU session. Once peer connections are established, video and audio streams from both peers are routed to MCU session. Multiple video and audio streams are mixed into a video stream and individual audio streams and routed to all participants (Fig. 6).

B. 3-Way Video Chat

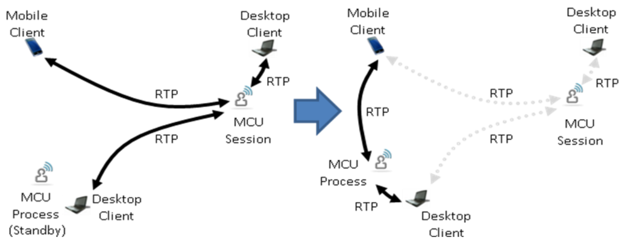


Fig. 7. Scenario involving two desktop client and mobile client.

When a participant joins a room at any time, the browser hosting the MCU session will invite the new participant to establish a peer connection (Fig.7 left). When a MCU host leaves the room at any time, the next appropriate participant will be elected to host a new MCU session and re-invite all participants in the room (Fig. 7 right). If there is no appropriate participant to host the session, the video chat is in pending mode. Multiple video and audio streams are always mixed in a MCU session.

C. Eight-Peers Video Chat

In the real word, different devices would have different capabilities. Mobile devices have less processing power, bandwidth, memory capacity, and screen size than desktop computers. Standard definition video suited for desktop computers is too demanding for mobile devices. In order to support heterogeneous devices, different devices should receive video and audio streams in their preferred profile.

In WebRTC, the basic method to support multiple participants is to use multiple Peer Connection objects, one for each participant. But this method is difficult to mix the streams into one and encoded it for different media profiles. In our approach, MCU host has sufficient processing power and network bandwidth to mix and deliver media streams for each participant in preferred profile directly as illustrated in Fig. 8.

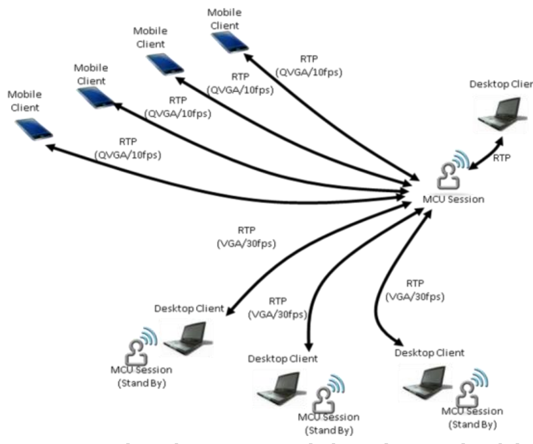


Fig. 8. Scenario involving heterogeneous desktop clients and mobile clients.

IV. MCU Host Determination

After the MCU is integrated in the browser as described in previous section, the important question is how to select a proper MCU host in the system to coordinate the conference effectively.

In theory, any browser in the system can be selected to perform MCU as long as it satisfies the MCU host requirements. But in practice, to make the selection efficiently, we assume MCU belongs to one browser in the conference in this paper.

In addition, a network control unit is needed to coordinate the signaling and information during the MCU host determination procedure. In our practice, the signaling server is one good candidate for the control unit.

Accordingly, there are three steps to determine the MCU host:

Step 1: the signaling server collects the information to determine the MCU;

Step 2: the signaling server determines which browser in the conference should be the MCU host based on the information collected in Step 1.

Step 3: the signaling server notifies all the browsers the selection result.

In details, for Step 1 and 2, the information includes network information and user information. The network information may include the network load information and topology information if necessary, while the user information may include the terminal type, whether or not the user is behind a firewall, and the terminal capability if necessary. Additionally, for Step 2, the signaling server quantifies the metrics indicated by the information, scores the metrics with predefined weights, and selects the client with best score as MCU. Finally, the signaling server should notify all the clients which one is the MCU host and the conference may start.

Specifically, the MCU host may be updated when one user enters the conference or the MCU host itself leaves the conference. Signaling server collects the user information when one client enters/leaves the conference or periodically, determines the MCU host and updates the MCU host information for every client if necessary.

An MCU host selection strategy example is shown in Table I below. Client 2 with the highest score is selected as MCU in the case.

TABLE I: AN MCU HOST SELECTION STRATEGY EXAMPLE

Client	Network Load/score	Terminal Type/score	Behind firewall?/score	Final score
1	Light/2	Mobile/0	Yes/0	2
2	Light/2	PC/2	No/2	6
3	Middle/1	PC/2	Yes/0	3
4	Heavy/0	PC/2	No/2	4

V. UDP HOLE PUNCHING FOR 3G NETWORK AND NETWORKS OF ENTERPRISE

In this section, we will describe how to eliminate TURN relay involvement in 3G environment. Symmetric NAT is commonly used in 3G networks and also in networks of enterprise. Symmetric NAT allocates different ports for outgoing UDP connection from same local host to different external hosts. STUN does not work in Symmetric NAT. If the device is in 3G mobile network and another device hosting MCU session is behind a port-restricted cone NAT, traffic between two devices will be blocked out by NAT in counter parties. TURN is the only solution.

In our model, we turn the behavior of port-restricted cone NAT into restricted cone NAT. MCU session gets the external host public IP embedded in SDP Offer and ICE candidates generated by participant. MCU session sends empty UDP packets with short TTL (Time to live) to all user ports of external host. Every time a UDP traffic from same local host to new external address pass through a NAT, the NAT adds an extra mapping rule for the same port allocation. As a result, UDP packets from external host can reach the MCU session.

VI. EXPERIMENTS

We evaluated the design of P2P-MCU solution through a field experiment and MCU host was realized by integrating the Medooze MCU Media Server [10] into Chromium [11] browser. We firstly evaluated the connection success rate, the setup time and the effectiveness of rejoining the conference room on Windows and Android clients. The result shows that the connection success rate is nearly 100% with a steady setup time and effective to re-connection the room.

A. Performance on Windows

We evaluated the performance of P2P-MCU and WebRTC mesh-network on Windows. For P2P-MCU experiment, the first desktop client was served as MCU host, which ran on Windows 8 Pro, Intel® Core™ i7-3520M CPU @ 2.90GHz, and other clients ran on Windows 7 Professional, Intel® Core™ i5-3210M CPU @ 2.50GHz. All video and audio streams were mixed by this MCU host. CPU usage and network utilization were captured on both sides and we repeated the steps with 2P, 5P and 8P. Fig. 9 shows 8P video chat with MCU between desktop clients. For WebRTC mesh-network experiment, see Fig. 10, all clients ran on Windows 7 Professional, Intel® Core™ i5-3210M CPU @ 2.50GHz, and CPU usage and network utilization were captured.

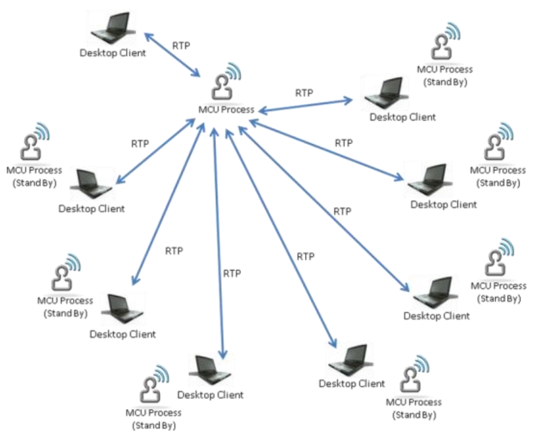


Fig. 9. Scenario involving desktop clients with P2P-MCU.

Fig. 11 summarizes the network performance of the experiments. We observed that the increase of number of peers did affect the performance of MCU host seriously, and the network utilization was steadily at low bandwidth consumption. It was in 7 to 8 Mbps range, when CPU usage was around 23% to 25%, as in Fig. 12. The success rate was nearly 100% with steady setup time. It was around 2 seconds per peer, and MCU host could see himself in a clip of mixed video.

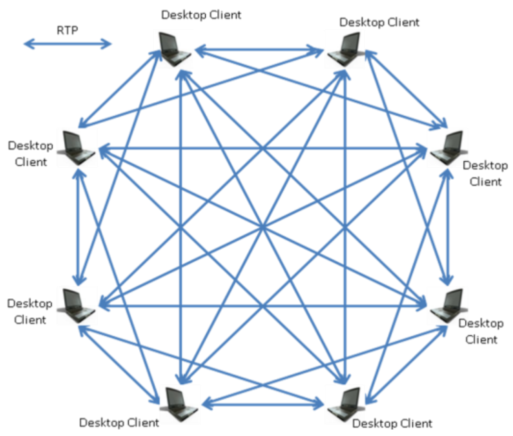


Fig. 10. Scenario involving desktop clients with WebRTC mesh-network (prefect world).

For the testing of WebRTC on mesh-network, we examined the results for both network utilization and CPU usage rise, where the CPU usage was stretched to the boundary of 93%.

We estimated the network utilization of 8P because of the limitation of the WebRTC mesh-network. We observed that the network utilization was not linearly proportional to the number of peers. The contributing factors may include that the process of trans-coding is slowed down a bit, some frames are dropped because of low bandwidth, or parts of signals are broken by firewall, see Fig. 13. There is a limit to add too hand, SPIN analyses the model against the given properties considering all possible executions by performing an exhaustive search on the state space. It can also perform partial search on the state space, which is quite useful in case of very large models or insufficient computational resources. If SPIN finds a violation, it produces an error trace. Using this error trace, a user can run a simulation of the execution that leads to the violation.

Our primary aim in this work is to verify the properties of the security adaptive protocol suite. We employ SPIN to check the protocol model against some properties that we formallyspecify as never claims in PROMELA and list any flaws, if any, as violations.

IV. Protocol Implementation Using SPIN

Model checking in SPIN is often bounded by the amount of physical memory available to the computer. To alleviate this problem it is required to reduce the complexity of the model. A simplified version of security adaptive protocol is being used in our experiment to avoid unnecessary details irrelevant to our verification. When modeling an ad hoc network protocol, apart from the usual consideration of limiting state space, it is required to pay attention to the way of modeling broadcast, connectivity as well as the dynamics of topology.

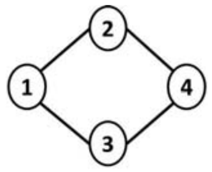
Broadcast is heavily used in most ad hoc networking protocol and it can be modeled by unicasting to all nodes with whom the sending nodes has connectivity. A HELLO message is used to maintain contact with its neighbors and also to contact new neighbors. Hello messages indicate the presence of a node.

Hello message are frequently sent by each node via channel. HELLO message is supposed to have only one piece of information: HELLO massage, source id number which identifies the node from where the hello message is coming from

PROMELA doesn't provide any time features except a timeout function. Timeout keyword is a modeling feature that provides an escape from a hang state which does not correspond to the real timer definition. So in this tools use time as variable to maintain the clock time.

Link update is required to maintain the ranks of the neighbors. Because a link update travels through the network, it represents the most up to date network topology information. When a route is requested from a source node, A, to a destination node, B, in AODV, a route request is broadcasted. For the security protocol suite, the basic principle will be same, the only difference lies in the fact that, before a route request is broadcasted, the security level requirement has to be defined, being called here as the Minimum Security Level(MSL). Here node A checks the trusted neighbors to send message. So this node checks the security level of all its neighbors. The source node will be notified of the fact that the route request that has been sent is not returning a path with the defined MSL. In this scenario, the source will now define a new MSL, by decrementing the MSL value, and will rebroadcast the Route Request to a new set of nodes.

A routing table is maintained for each node. Whenever a packet is to be transmitted from one node to another.



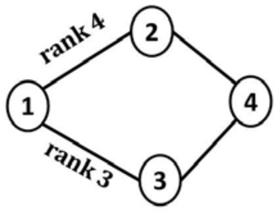
Routing table for node 1

Destination	Next-hop 1	Next-hop 2	Hop
4	2	4	2
4	3	4	2
3	3	-	1
2	2	-	1

Routing table for node 3

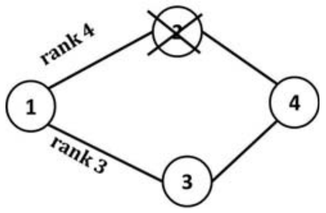
Destination	Next-hop 1	Next-hop 2	Hop
2	4	2	2
2	1	1	2
4	4	-	1
1	1	-	1

Fig. 1. Routing information for node 1 and node 3.



Destination	Next-hop 1	Next-hop 2	MSL Level
4	2	4	Rank 4

Fig. 2. Route table with node rank (MSL).



Destination	Next-hop 1	Next-hop 2	MSL Level
4	3	4	Rank 3

Fig. 3. Route update when MSL changes.

Routing table is consulted along with the rank information of each node and keeping the MSL value. The process of message broadcasting reference to MSL will help the route request reach the destination with trusted neighbor. Fig. 1 illustrates the route table of two nodes.

Suppose Node 1 in Fig 2 is broadcasting a message destined to Node 4. The MSL level is kept at rank 4. In case a route is not able to be established with the initial MSL, with which the *Route Request* was broadcasted, an *Error* packet will be generated. Then source will define new minimum MSL, by decrementing the MSL value and will rebroadcast the *Route Request* to new set of nodes (Fig 3).

A. Property Specification

Among the various properties related to any SAODV protocol, we are interested in two properties crucial to the earlier mentioned protocol suite. The first one is the loop freedom and another is the maintenance of correct rank of the neighbours based on their distances. It is also need to be

VII. Conclusion

We described a P2P-MCU approach to support multi-party WebRTC conference even with a general Android mobile. Our P2P-MCU solution is transparent for the mobile users as they can just install the official Chrome for Android on their smart phones. Although the new added P2P-MCU module may introduce some delay (< 500ms), the delay is stable and perceptually almost neglectable for the participants. Our MCU host determination strategy guarantees the conference be established and recovered seamlessly. Accordingly, the performance of P2P-MCU is quite stable and the success rate of establishing connection in 3G networks is almost 90%, which is much higher than that in normal WebRTC. Experimental results from an eight party video conference experiment indicated that our solution can reduce 64% CPU usages and 35% bandwidth consumptions for each participant compared to a pure WebRTC mesh network.

Acknowledgment

The authors would like to thank ASTRI Program Director Dr. James Zhibin Lei and ASTRI Principal Engineer Dr. Kent Kangheng Wu for their support and suggestions in this project.

References

[1] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan. WebRTC 1.0: Real-time Communication between Browsers. [Online]. Available: http://www.w3.org/TR/webrtc

[2] M. H. W. LeMair, D. D. Kandlur, and Z. Y. Shae, "On multipoint control units for videoconferencing," in *Proc. 19th Conference on Local Computer Networks*, 1994, pp. 356–364.

[3] P. Srisuresh, B. Ford, and D. Kegel. State of Peer-to-Peer (P2P) Communication Across Network Address Translators (NATs). [Online]. Available: http:/www.rfc-editor.org/rfc/rfc5128.txt.

[4] I. Fette and A. Melnikov. The WebSocket Protocol. [Online]. Available: http:/www.rfc-editor.org/rfc/rfc6455.txt

[5] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. [Online]. Available: http:/www.rfc-editor.org/rfc/rfc4566.txt

[6] J. Rosenberg. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. [Online]. Available: http:/www.rfc-editor.org/rfc/rfc5245.txt

[7] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). [Online]. Available: http:/www.rfc-editor.org/rfc/rfc3489.txt

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. [Online]. Available: http:/www.rfc-editor.org/rfc/rfc3550.txt

[9] R. Mahy, P. Matthews, and J. Rosenberg. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). [Online]. Available: http:/www.rfc-editor.org/rfc/rfc5766.txt

[10] MCU Open Source Version. [Online]. Available: http://sourceforge.net/projects/mcumediaserver/

[11] Chromium. [Online]. Available: http://www.chromium.org

Kwok-Fai Ng was born in Hong Kong in 1978. He got his bachelor's degree in system engineering and engineering management from The Chinese University of Hong Kong, Hong Kong in 2000.

He worked in TradeCity Cybersoft for J2EE application server development in Hong Kong from 2000 to 2003. Currently he is principal engineer for enterprise and consumer electronics at Hong Kong Applied Science and Technology Research Institute. He is an expert in J2EE technologies and is oracle certified SCJP, SCWCD, SCBCD and SCDJWS. He developed a variety of technologies and software programs, including digital asset management system, multi-platform eBook reader, real-time social collaboration system, NAT traversal solution, web service collaboration platform, data persistence framework, etc

Man-Yan Ching earned her B.Eng. degree from the the Hong Kong University of Science & Technology. She got her M.S. degree in electrical and electronic engineering from The University of Hong Kong in 2003.

She worked in InfoTalk for speech recognition development in Hong Kong from 2001 to 2002. Currently she is working in Hong Kong Applied science and technology research institute and she developed a variety of software programs, including embedded system, multimedia technologies and multi-platform eBook reader.

Yang Liu got his B.Eng. and M.S. degrees from the Southeast University, Nanjing, China in 1997 and 2001, respectively. He got his Ph.D. degree in electrical and electronic engineering from the University of Hong Kong in 2006.

He worked in ZTE corporation for 4G/5G communication standard development in Shenzhen, China from 2007 to 2013. Currently he is working in Hong Kong applied Science and technology research institute (ASTRI). His research interests include communication protocol, network security and related applications.

Tao Cai got his B.S. and M.S. degrees in applied mathematics from Beijing Institute of Technology in 1999 and 2002 respectively.

He has over 10-years software R&D experiences and is certified Project Management Professional (PMP). Currently he is a researcher at Huawei Shannon Lab and focuses on WebRTC, HTML5, SDN and web browser.

Li Li received his Ph.D. in computer sciences from University of Alabama at Birmingham in 1995, and M.S. in computational linguistics from Huazhong University of Sciences and Technology in 1987.

In 2001, he became a research scientist of Avaya Labs, and in 2012 he joined Huawei Shannon Lab. Dr. Li has published over 50 conference and journal papers and 1 book on Artificial Intelligence. He holds 4 US patents and is the co-inventor of over 20 pending US patent applications. He was on the Advisory Committee for IEEE Service Cup Competition, invited panelist and technical committee member for several international conferences on Web Services. He was the editor of 2 ISO/ECMA CSTA standards and made significant contributions to W3C WS-RA standard suite.

Wu Chou graduated from Stanford University in 1990 with four advanced degrees in science and engineering.

He is a VP, chief IT scientist, and the head of Huawei Shannon Lab, USA. He joined AT&T Bell Labs after obtaining his Ph.D. degree in electrical engineering and continued his professional career from AT&T Bell Labs to Lucent Bell Labs and Avaya Labs before joining Huawei. He published over 140 journal and conference papers, holds 29 US and international patents with many additional patent applications pending. He is an IEEE Fellow and serves as an editor for multiple standards at W3C, ECMA, ISO, ETSI, etc. He was an editor of IEEE Transactions on Services Computing (TSC), IEEE TSC Special Issue on Cloud Computing, IEEE Transaction on Audio and Language Processing and Journal of Web Services Research.