Teledrive: A *Multi-master* Hybrid Mobile Telerobotics System with Federated *Avatar* Control

Ashis Sau(✉), Abhijan Bhattacharyya, and Madhurima Ganguly

TCS Research, Tata Consultancy Services, Kolkata, India
{ashis.sau,abhijan.bhattacharyya,ganguly.madhurima}@tcs.com

Abstract. This paper addresses the issue of inherent delay in cloud-centric architecture a scenario where multiple remote human users join a tele-presence/tele-operation session with a robot such that any one of the users may take exclusive control of the robot and remotely maneuver it as the Avatar of the current Master. It presents a novel WebRTC based signaling protocol which allows a hybrid topology for the multi-user session. While the audio/video conferencing happens over cloud, each Master can create an on-demand peer-to-peer channel with the Avatar for the desired duration of maneuvering the Avatar. Thus, it allows low-latency delivery of command resulting in better experience for the end-user. The system is deployed in public cloud and is operational. The efficacy of the system is established through benchmarking against cloud-centric architecture in real-life field trials.

Keywords: Telepresence · Cloud-robotics · P2P · WebRTC

1 Introduction

1.1 Motivation

The prevalent pandemic situation requires the human civilization to continue 'social' cooperation while maintaining physical 'distancing'. Hence, the unprecedented hindrances in traveling, physical assembling, etc. has reinforced the need for mobile telerobotics based applications. In mobile telerobotics a human Master interacts with a remote environment through a robot. The robot in the remote environment acts like an Avatar of the human Master. In the simplest form, the Master establishes a real-time audio-video chat with the Avatar over the Internet and, based on the visual feedback from the Avatar, the Master sends control commands to remotely maneuver the Avatar in real-time. In a Telepresence scenario the maneuvering is limited to moving the robot around the remote environment as desired by the Master. The off-the-shelve telerobotics systems are mostly one-to-one. A single Master connects to an Avatar remotely and maneuvers it while having audio/video exchange with the remote environment.

T. Hara and H. Yamaguchi (Eds.): MobiQuitous 2021, LNICST 419, pp. 102–114, 2022.
https://doi.org/10.1007/978-3-030-94822-1_6

However, in a practical collaborative situation, it may be required that multiple human users need to connect to the robot simultaneously in a session to perform a composite task through the robot. While having a real-time multiparty A/V conference, each user may need to perform a part of the task. Thus, each user needs to act as a Master for a certain duration with exclusive access to the robot and the robot acts as the Avatar of that particular user for the stipulated period. Once, the task is completed, current Master may relinquish the control. Another user may take control and elevates as a Master. However, irrespective of which user is the current Master, all the users are able to continue participating in the A/V chat and receive visual feedback from the Avatar as well as other users. For example, in a telemedicine scenario (Fig. 1), a medical assistant and a specialist doctor may join a session with a telerobot from two distant locations to provide consultation to a patient in isolation ward. In the beginning the medical assistant, who is aware of the geography of the patient's premise, remotely maneuvers the robot to navigate near the patient. After this she relinquishes the control and the specialist doctor takes control and becomes the Master to pursue further interactions with the patient. The medical assistant continues in the A/V conference to follow the doctor's instructions.

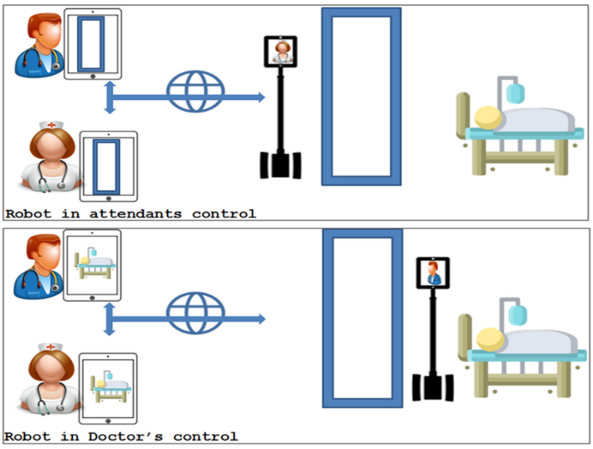


Fig. 1. The telemedicine application with users collaborating with the robot.

1.2 WebRTC in Telerobotics

WebRTC [1] is the dominant technology for real-time telerobotics applications like [2]. It is primarily designed to use the Internet browser as the user interface (UI) for multimedia conferencing. The audio/video is exchanged over the media channel on SRTP [1] and any other data is exchanged over a reliable data-channel on SCTP. The main reason of WebRTC's popularity is its inherent capability to establish low-latency end-to-end (E2E) channel through peer-to-peer (P2P) connection between the browsers of the participating Master and Avatar (Fig. 2a) [3]. However, the P2P connection may have to pass through a TURN (Traversal Using Relays around NAT) server when a direct P2P is not possible for nodes behind restricted NATs (Fig. 2b) [4, 5]. Most telerobotics

solutions work in a P2P setting with only one Master connecting the Avatar. However, in a multiuser scenario all the participating parties, including the robot, would require to connect through a conferencing server in cloud in a star-like topology. This would break the true P2P paradigm. The cloud centricity causes delay in both transfer of control commands as well as in sharing the visual feedback which is critical for the end-user Quality of Experience (QoE) for telerobotics.

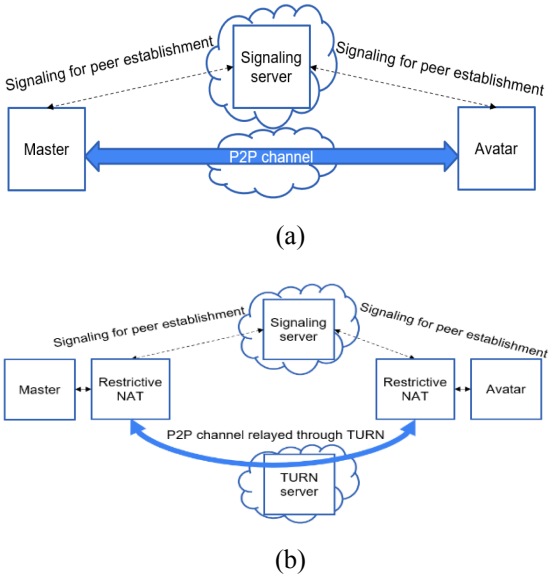


Fig. 2. a) WebRTC with direct P2P connection. b) WebRTC P2P connection via TURN server.

1.3 Salient Contribution and Organization of the Paper

This paper primarily addresses the latency issue of such cloud-centric multi-user telerobotics system. The key contribution of this paper is, *Teledrive*, a practical production-grade deployment of the telerobotics architecture such that, while remaining cloud-centric, the delay-critical control commands can be transferred over a P2P data-channel between the robot and the current Master on demand. Thus, *Teledrive* enables a dynamic configuration with hybrid topology where the media is exchanged over the server in the cloud and the control is transferred over the on-the-fly P2P connection. Thereby *Teledrive* aims to improve the latency in maneuvering the Avatar. Such an architecture is unique to best of our knowledge.

Furthermore, since multiuser telepresence system, as described, is not available, we modified *Teledrive* to a completely cloud-based system where the control is also transferred through the cloud server. We compare the performance of the cloud version and hybrid version of teledrive in both Quality of Service (QoS) and QoE measures in a practical deployment with distant users connecting to a custom robotic base and prove the efficacy of the hybrid *Teledrive*. QoE measures are performed based on user

experience on completion of a task by the Avatar. Also, since the media always travels through the cloud, we compare the impact of reduced delay in the control path against the video latency as revealed in the QoE experiments.

Even further, we check the performance of hybrid *Teledrive* in two different P2P settings: with TURN and without TURN. We show that the performance of the hybrid version is further improved when the P2P is established without TURN.

To best of our knowledge, such an extensive practical deployment based benchmarking and architectural innovation is not present in the existing literatures. We hope that this work would open up new research directions towards delay compensation in cloud-centric telerobotics.

The rest of the paper is organized as follow. The next section describes the detail architecture of *Teledrive* for both cloud centric and hybrid versions. Section 3 describes the S/W and H/W deployment of the system for the required testbed. Section 4 describes the experiment methods for QoS and QoE aspects and the results with relevant analysis. Section 5 presents description of the related state-of-the-art. Finally the paper concludes with future research directions.

2 System Architecture

2.1 Generic Architecture

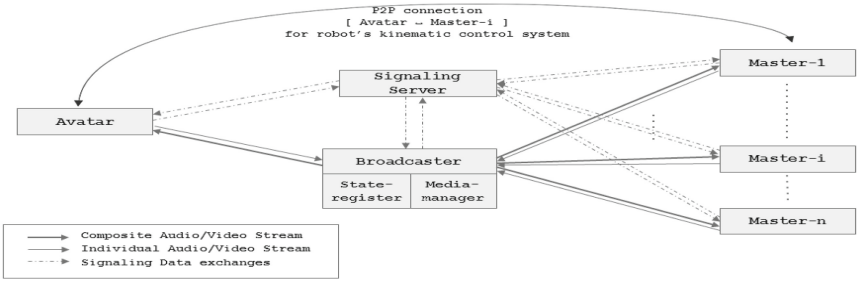
The multi-user connectivity is established through a special node called *broadcaster* (Fig. 3a). The *broadcaster* maintains separate P2P relationship with each human user nodes and the robot. Thus the 'star' is formed. The *broadcaster* receives A/V feedback from all the individual nodes over the P2P media channel and shares the composite A/V combining all the feeds all the other users connected to the *broadcaster* over the individual P2P channel. The *broadcaster* maintains the states of all the peers in each P2P channel and a *media-manager* inside the *broadcaster* is responsible for combining the video in a customized manner for each peer. The *state-register* maintains the state for each peer's connection state, especially the state of the robot. Figure 3b depicts the state transition between different robot states. The state also contains which user is the current Master (as discussed next). The *signaling server* helps establish the peering of the broadcaster with each peer. The *signaling server* and the *broadcaster* both are in public cloud. The Master users and the Avatar can be behind different types of NATs or contain public IPs. The P2P connections may traverse through a TURN server or be direct depending on whether the respective peer is behind restrictive NAT.

We have used a custom messaging for signaling. The basic application message structure in JSON format is:

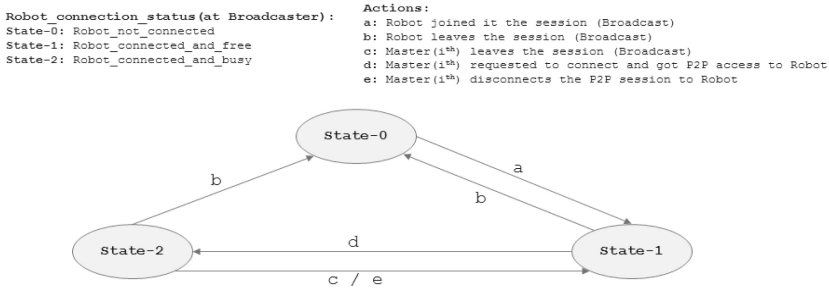
```
{"connection-type":< >, "room-id":< >, "message-type":< >,
"message":< >}.
```

Interpretation of each parameter is given in Table 1. The individual message semantics are given in Fig. 3c. Each node has to specify its intended roleplay, i.e., Master (*peer type 'M')* or Avatar *(peer type 'A')*. Type *'B'* is dedicated for the *Broadcaster*. Based on

that it is decided whether the control interfaces are going appear for the user or not. The i-th Master would have the control interfaces enabled only when it is able to establish a control channel with the robot successfully.



(a)



(b)

i. **Request connection/disconnection:** { "request": < 0 (disconnect) / 1 (connect) >, "peer_id": <>, "peer-type": < 'B' (broadcaster) / 'M' (master) / 'A' (avatar) > }

ii. **Send connection request acceptance:** { "acceptance-status": < 0 (reject) / 1 (accept) >, "robot-id": <>, "master-id": <>, "robot-connection-status": < state- 0/1/2 > }

iii. **Initiate connection:** { "peer_id": <>, "peer-type": < 'B' (broadcaster) / 'M' (master) / 'A' (avatar) > }

iv. **Send connection establishment status:** { "peer_id": <>, "status": < 0 (channel not established) / 1 (channel established) >}

v. **Send Robot-connection-status:** { "master_id": <>, "robot-connection-status": < state- 0/1/2 >}

vi. **Send Offer:** { "peer_id": <>, "sdp": <RTC Session Description> }

vii. **Send Answer:** { "peer_id": <>, "sdp": <RTC Session Description> }

viii. **Send ICE Candidate:** { "peer_id": <>, "candidate": <ICE Candidate> }

(c)

Fig. 3. (a) The generic WebRTC based architecture for Teledrive; (b) the robot state machine maintained at the broadcaster; (c) the semantics of individual message types exchanged.

In the general cloud-only configuration, each potential Master willing to control the Avatar sends a data channel request to the broadcaster through the signaling server. The

broadcaster always maintains a data channel connection to the Avatar. If the broadcaster finds the status of the Avatar as 'free', it intimates connected status to the ith user and the user is elevated as Master. The Avatar state is changed to 'busy' and all other peers are notified the same. Henceforth, until disconnection, the ith Master sends all the control commands to the broadcaster peer, which in turn relays it to the Avatar. The execution status is likewise relayed back to the Master. Thus, both the A/V information and control information are relayed through the broadcaster in the cloud.

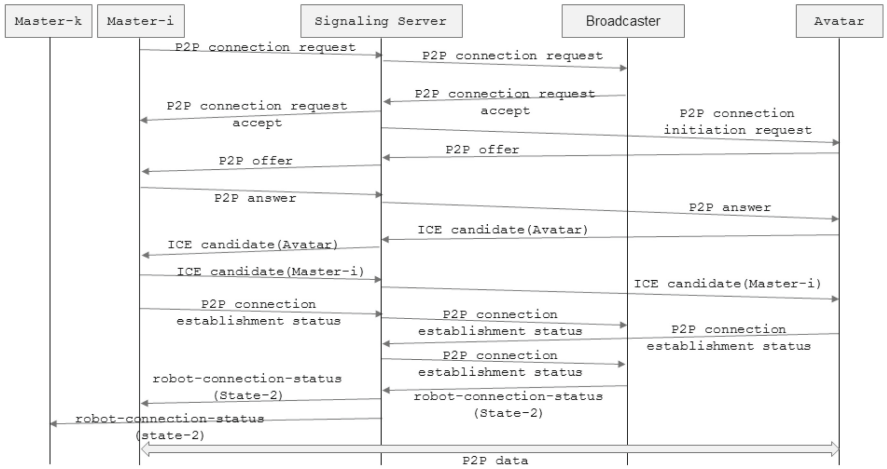
Table 1. Description of parameters in Teledrive message formats.

Parameter	Interpretation
Connection-type	0: Connection for broadcaster 1: Connection for any other peer
Room-id	<any string>: The meeting room ID for the session
Message-type	000: Request connection/disconnection 001: Send connection request acceptance 010: Initiate connection 011: Send connection establishment status 100: Send Robot-connection-status 101: Send Offer 110: Send Answer 111: Send ICE Candidate
Message	Type-specific message in JSON format (Ref. Fig. 3c)

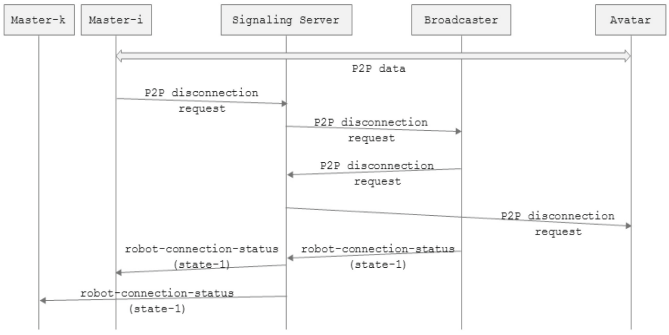
2.2 Protocol for the Hybrid Topology

For the hybrid configuration, which is the key architectural contribution of this paper, the *broadcaster* involves a few more signaling to facilitate a P2P connection between the i^{th} *Master* and the *Avatar* (Fig. 4a). In this case, when the i^{th} *Master* wants to connect to the *Avatar*, *signaling server* that to the *broadcaster* which maintains all the states of connected peers. In response, the *broadcaster* notifies the ICE ID [1] and current state of the *Avatar* to the *signaling server* and it relays that to the i^{th} user. The *signaling server* also requests the *Avatar* to offer a peer connection to the ICE candidate corresponding to the i^{th} user. Once the ICE candidate states are exchanged between the Master and the Avatar, the *signaling server* notifies the same to the *broadcaster* and a P2P data channel is established between the i^{th} *Master* and the *Avatar*. Thus, while the A/V is still exchanged via the *broadcaster* through star topology, the control is transferred over a P2P connection to ensure low-latency as will be proven in the next sections.

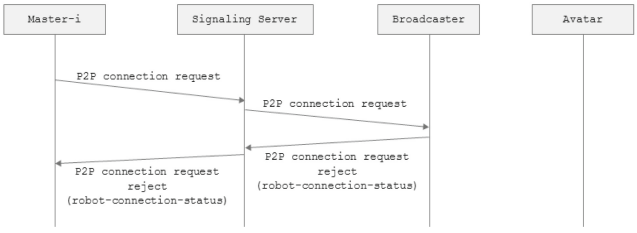
Once the i^{th} *Master* disconnects the control (Fig. 4b), the *signaling server* ceases the P2P *data channel* and intimates the same to the *broadcaster*. The *broadcaster* marks the *Avatar* as *free* and notifies all the other users. Figure 4c illustrates the protocol to decline the P2P control connection request when the Robot is altogether disconnected from the ongoing session or is acquired by any other *Master*.



(a)



(b)



(c)

Fig. 4. The timing diagram of the protocol to (a) establish P2P control channel between the i^{th} Master and the Avatar when the Avatar is in free state; (b) decline a connection attempt when the robot is busy, (c) decline a connection request when the robot is not in the ongoing session or already acquired by other *Master*.

3 Deployment and Test-Bed

We have created a custom built WebRTC stack using the standard WebRTC APIs using Java Script and HTML 5. The *signaling server* runs on node.js. The nodes acting as Master/Avatar uses Chrome or Mozilla browsers for the user interface. The *broadcaster* uses a headless chrome implementation called *puppeteer* [6]. The media manager within the *broadcaster* is built using the *video stream merger* APIs from *NPM registry* [7]. The *Avatar* is an Intel Core i5 laptop running Ubuntu 16.04. The *Avatar* laptop is fixed on a custom-built robotic base created using "Arduino Uno R3" microcontroller. The microcontroller is connected to the laptop through USB port. The microcontroller is augmented with L293D motor driver shield, 4 gear motors running 4 wheels in front, rear, left and right directions. The system is powered by a 2200 mAh battery. The Avatar machine is installed with Python 3.9. The interfacing between the controller and the Avatar machine is done through serial port using *PySerial* library APIs. All commands are passed through this. The connection between the *Avatar* browser and the Python module is done through a WebSocket within the scope of the local host. The browser code runs a WebSocket client which connects to local Python WebSocket server on the *Avatar* laptop. The WebSocket server houses the *PySerial* APIs. Thus all the control commands received by the *Avatar* browser are transferred to the microcontroller through the WebSocket connection. This way the i^{th} *Master* can use the browser UI to remotely maneuver the Avatar.

The broadcaster and the signaling servers are hosted in AWS cloud on two EC2 *t3.xlarge* instances. The TURN server is built using COTURN [8] on one of the instances. The cloud instances are located in the *US-EAST region.*

Figure 5(a) and (b) show the user interfaces for the active *Master* user and the *Avatar* respectively in a real-life experiment with three potential *Master* users and the *Avatar* joining from different locations around the city of *Kolkata, India* and the distance between the Avatar and other users range between *50 to 100 km.*

4 Experiments, Results and Analysis

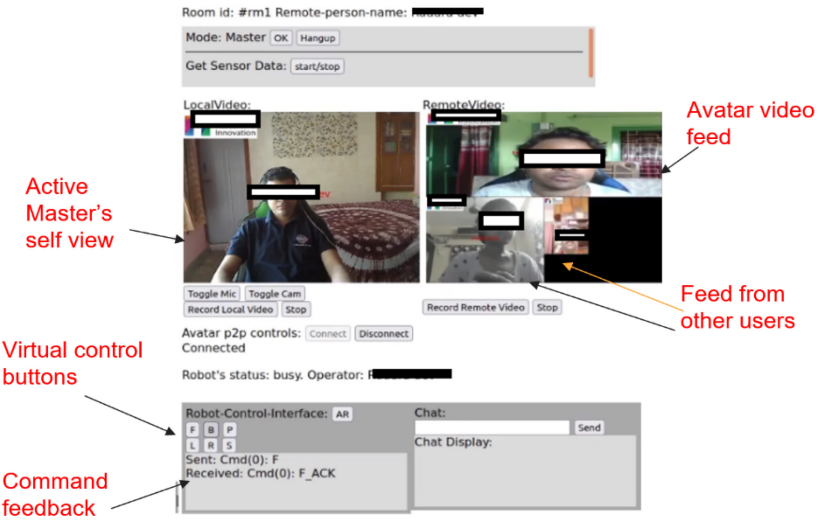
4.1 On Network Aspects

We first compare the latency performance in terms of the closed loop response time in terms of:

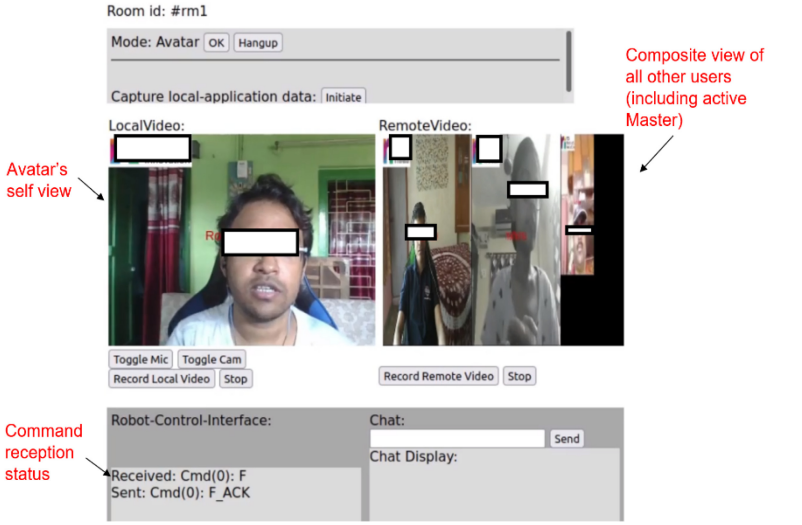
$$RTT = t2 - t1,$$

where, t1 = time recorded at the Master browser when a command button is pressed and t2 = time recorded when an acknowledgement is received at the Master browser.

The Java Script is modified to report *RTT* on the *Master* browser after every command. We note the values for 3 different users from three different locations as specified above at four different segments of the day. The segments are *morning (7–7:30 am), afternoon (2–2:30 pm), evening (6–6:30 pm), night (9–9:30 pm).* The five-day average of *RTT* for all the users for each segment of the day and the overall average is shown for both cloud-only configuration and the hybrid configuration in Fig. 6. The *Avatar* is



(a)



(b)

Fig. 5. (a) UI of the active Master; (b) UI of the Avatar.

connected to an ISP via WiFi AP such that the ISP forces *server reflexive address* [4] for all nodes connected to it. Thus, all the WebRTC peering are over the TURN server. It is observed that in India most of WiFi service provider ISPs force the use of NAT.

From the result we see that in the hybrid mode the command response is received much faster than the cloud-only mode. This is because, in hybrid mode, the control commands pass through the P2P channel, rather than through the *broadcaster* cloud. Although it passes through the TURN relay, still we get a reduction in the order of several hundreds of milliseconds.

We repeat the above experiment with all users and the *Avatar* connecting using 4G SIM from smartphones, rather than WiFi APs. In this case TURN relay is not required. We compare the command response for hybrid-mode with the situation where P2P is established through TURN server. The result is shown in Fig. 7. We observe that there is a further reduction by hundreds of milliseconds in the latency when the TURN relay is not used. One of the reasons of such a huge reduction is that the TURN server's location is in US, while P2P is formed within India. It is also to be seen that in the peak hours the difference in RTT for the two configurations minimizes. This also due to the load in the long backhaul during the peak usage in India and US respectively. Relocating the TURN server in India might have reduced the delay in TURN. Also, the effect of time axis is not significant for direct P2P. Hence, the results strongly establish the efficacy of hybrid Teledrive.

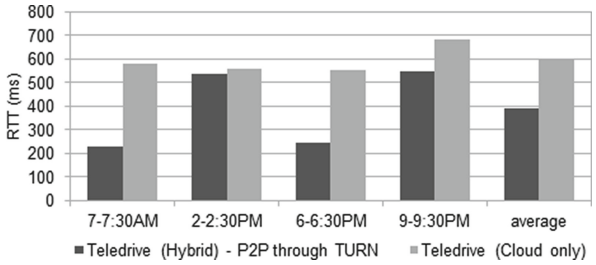


Fig. 6. RTT comparison between cloud only and hybrid configurations of Teledrive when the P2P command control channel passes through TURN server.

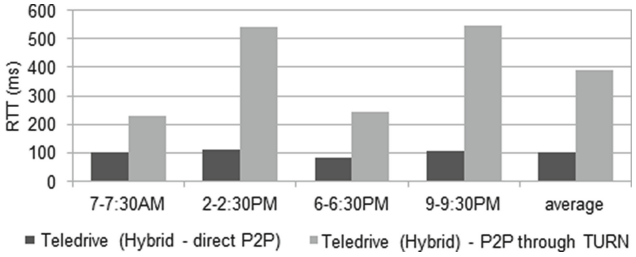


Fig. 7. RTT comparison between two different types of P2P command control channel for the same hybrid configuration: One passes through TURN server and the other is direct.

4.2 On Visual Aspects

For application user's paramount importance for QoE would be when the user sees the desired visual feedback from the Avatar after pressing a command button. To accomplish this, we create a set up as shown in Fig. 8. We place a blue bottle in camera-front at the original position of the Avatar and put a red bottle at a perpendicular position such that the Avatar faces this bottle when it rotates left by ninety degrees. We use the H/W platform described in Sect. 3. We repeat the similar exercise as described above. However, every Master presses only the L (rotate left) button. We developed a time calculation tool for this by enhancing Java Script code running at the Master browser. Whenever the user presses L a timer starts on the browser. As soon as the Master sees the complete view of the red bottle on the console, he/she presses a stop button and the timer stops. We call this time *Response time* defined as:

$$Execution\ time = t2 - t1,$$

where, t1 = time recorded at the Master browser when L command button is pressed and t2 = time recorded when Master presses stop button.

The results of this exercise are shown in Fig. 9 as average in time buckets and the overall average. We see a faster average response for the hybrid configuration. This is because the execution starts much earlier in case of hybrid configuration since the desired command is communicated to the *Avatar* much faster due to the P2P between the *Master* and *Avatar*.

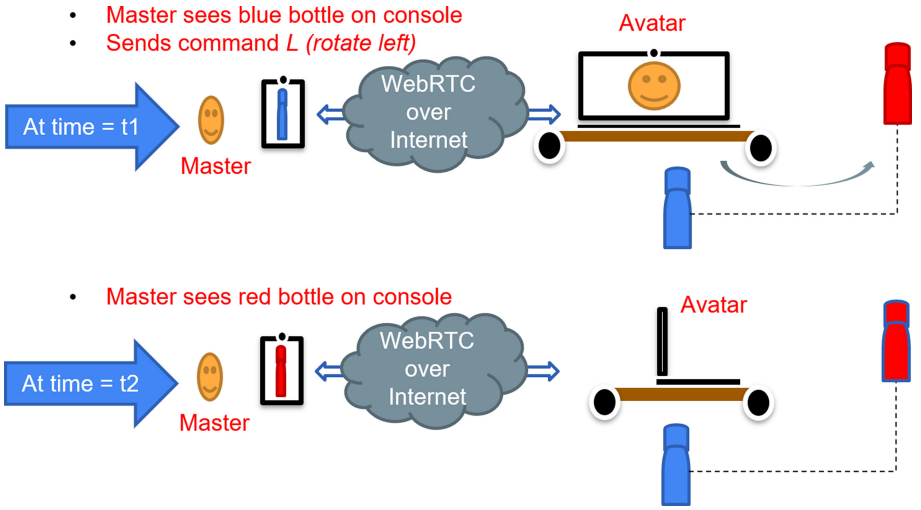


Fig. 8. Experimental setup to observe the visual aspects

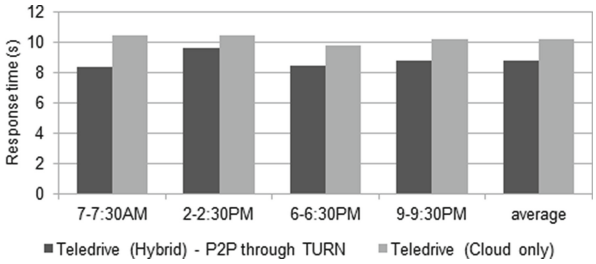


Fig. 9. Result of experiments on visual aspects

5 Analysis on the State-of-the-Art

To best of our knowledge we have not encountered any multiuser federated Telerobot controlling architecture as specified in this paper. [9] discloses an application which would allow doctors and nurses to join a telemedicine session with robot. However, this primarily deal with GUI design for the solution and does not disclose any Internet based real-time architecture as *Teledrive*. [10] proposes a multiuser robot control, but they deal with modelling the human behaviour when both user controls the robot simultaneous to access the different sensors of the robot to perform parts of a task collaboratively. It also does not deal with any Internet based architecture for federated Tele-presence or Tele-operation with a hybrid topology as *Teledrive*.

There are many literatures like [11] which have dealt with delay compensation in cloud robotics. But they deal with AI based solutions in specific environments.

WebRTC has been widely used in commercial Telepresence systems as [2] but those do not support a multiparty system. Literatures like [3] emphasize the importance of WebRTC in futuristic haptic communication and propose standardization of haptic data exchange, but it does not deal with any multi-user federated control scenario. Even literatures like [12, 13] also deal with the simple peer-to-peer scenario.

6 Conclusion

Teledrive consists of a unique Telerobotics communication architecture with custom signaling protocol built using WebRTC APIs. The hybrid topology allows multiple human users join a session along with a telerobot. While the users may continue A/V conferencing through a cloud infrastructure, any one of the human users may acquire exclusive control on the robot and make it his/her Avatar. The commands to control the Avatar would no longer travel through the cloud, rather, in the interest of reduced latency in remote execution, would be transmitted over an on demand P2P channel created between the active *Master* and the *Avatar*. The practical on-field deployment of the architecture is proven to have better performance compared to cloud centric architecture. To best of our knowledge, there are no off-the-shelve solution available that allow such multi-user federated fast maneuvering of a telerobot.

We look forward to deploy this on a standard platform as [2]. The future plan is to extend this hybrid topology to multirobot systems. Also, it remains to be seen whether the latency saving in sending the command is reciprocated by the visual feedback which still travels through the cloud. We would need to observe the time taken by the video to reach the *Master* after the execution has finished at the *Avatar* side. The resulting investigations may reveal new dimensions of research in compensating the delay in the video to reciprocate the low-latency command path. We will explore this in our next publication.

References

1. Alvestrand, H.: RFC 8825, Overview: Real-Time Protocols for Browser-Based Applications, IETF (2021)
2. Double Robotics - Telepresence Robot for the Hybrid Office. https://www.doublerobotics.com/. Accessed 18 July 2021
3. Iiyoshi, K., et al.: Towards standardization of haptic handshake for tactile internet: a WebRTC-based implementation. In: 2019 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE), Malaysia (2019)
4. Keranen, A., Holmberg, C., Rosenberg, J.: RFC 8445, Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal, IETF (2018)
5. Ford, B., Srisuresh, P., Kegel, D.: Peer-to-peer communication across network address translators. In: Proceedings of the Annual Conference on USENIX Annual Technical Conference, ACM, USA (2005)
6. Puppeteer. https://github.com/puppeteer/puppeteer. Accessed 18 July 2021
7. Video-Stream-Merger – npm. https://www.npmjs.com/package/video-stream-merger. Accessed 18 July 2021
8. Coturn. https://github.com/coturn/coturn. Accessed 18 July 2021
9. Wang, Y., Jordan, C. S., Pinter, M.: Tele-presence robot system with multi-cast facility, United States Patent No. US 984219B2 (2017)
10. Lee, D.G., et al.: Human-centered evaluation of multi-user teleoperation for mobile manipulator in unmanned offshore plants. In: 3 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Japan (2013)
11. Tian, N., Tanwani, A.K., Goldberg, K., Sojoudi, S.: Mitigating Network Latency in Cloud-Based Teleoperation Using Motion Segmentation and Synthesis, International Symposium on Robotics Research, Vietnam (2019)
12. Melendez-Fernandez, F., Galindo, C., Gonzalez-Jimenez, J.: A web-based solution for robotic telepresence. Int. J. Adv. Robot. Syst., SAGE (2017)
13. Tan, Q., et al.: Toward a telepresence robot empowered smart lab. Smart Learn. Environ. **6**(1), 1–19 (2019). https://doi.org/10.1186/s40561-019-0084-3