# Evaluation of WebRTC in the Cloud for Surgical Simulations: A case study on Virtual Rotator Cuff Arthroscopic Skill Trainer (ViRCAST)

William Kwabla[1], Furkan Dinc[1], Khalil Oumimoun[1], Sinan Kockara[2] [0000-0002-5881-1653], Tansel Halic[3] [0000-0002-2558-4001], Doga Demirel[4,*] [0000-0002-8270-1163], Sreekanth Arikatla[5], and Shahryar Ahmadi[6]

[1]University of Central Arkansas, Conway, Arkansas, USA
[2]Lamar University, Beaumont, Texas, USA
[3]Intuitive Surgical, Peachtree Corners, Georgia, USA
[4]Florida Polytechnic University, Lakeland Florida, USA
[5]Kitware Inc., Carrboro, North Carolina, USA
[6]Memorial Orthopaedic Surgical Group, Long Beach, California, USA

*ddemirel@floridapoly.edu
william.k.kwabla@gmail.com, furkandinc71@gmail.com, khalil.oumimoun@gmail.com, skockara@lamar.edu, Tansel.halic@intusurg.com, ddemirel@floridapoly.edu, sree.arikatla@gmail.com, ahmadismd@gmail.com

**Abstract.** Web Real-Time Communication (WebRTC) is an open-source technology which enables remote peer-to-peer video and audio connection. It has quickly become the new standard for real-time communications over the web and is commonly used as a video conferencing platform. In this study, we present a different application domain which may greatly benefit from WebRTC technology, that is virtual reality (VR) based surgical simulations. Virtual Rotator Cuff Arthroscopic Skill Trainer (ViRCAST) is our testing platform that we completed preliminary feasibility studies for WebRTC. Since the elasticity of cloud computing provides the ability to meet possible future hardware/software requirements and demand growth, ViRCAST is deployed in a cloud environment. Additionally, in order to have plausible simulations and interactions, any VR-based surgery simulator must have haptic feedback. Therefore, we implemented an interface to WebRTC for integrating haptic devices. We tested ViRCAST on Google cloud through haptic-integrated WebRTC at various client configurations. Our experiments showed that WebRTC with cloud and haptic integrations is a feasible solution for VR-based surgery simulators. From our experiments, the WebRTC integrated simulation produced an average frame rate of 33 fps, and the hardware integration produced an average lag of 0.7 milliseconds in real-time.

**Keywords:** webRTC, cloud computing, surgical education, surgical simulation, remote collaboration.

# 1 Introduction

Web Real-Time Communication (WebRTC) is a web-based technology which provides audio/video calls, chats, peer-to-peer (P2P) file-sharing functionalities, and everything in between to web and mobile applications without additional third-party plugins. WebRTC can be used in many different domains. In this study, we investigate the usability of WebRTC on surgical simulations.

Surgery simulation is a specialty where students and professionals train and practice modern surgical procedures. Recent developments in virtual/augmented reality (VR/AR) introduce new possibilities and dimensions to surgery simulations. Due tof the high fidelity, real-time, 3D animations and the ability to manipulate hardware instruments attached or integrated into haptic devices, surgical communities have adopted VR/AR-based medical simulations [1]. Surgical simulations can range from simple suturing exercises for an individual student to advanced robotic surgery simulations for expert surgeons. Medical simulations have been shown to reduce costs, medical errors, and mortality rates while improving providers' performance [2][3].

Current surgical simulations require the physical presence of an experienced surgeon with a trainee, which can even be difficult due to the busy schedule of expert surgeons. When the COVID-19 pandemic hit the world in 2020, it forced people to work remotely, including medical professionals and trainees. This shift brought many changes to many industries across the world. Medical education was one of the most critical fields affected by this [4][5]. The disruption has curtailed an essential part of surgical education which is the acquisition of surgical skills through continuous practice [6][7]. With the current shift towards working and schooling from home, there is a need to explore new ways of using surgical simulations to foster remote collaborations and continuous practice to gain surgical skills.

Cloud computing, delivers computing services, including servers, storage, databases, networking, software, analytics, and intelligence over the internet to offer faster innovation, flexible resources, and economies of scale [8].

Recent advancements in cloud computing have the potential to open doors to exploit different ways of carrying out surgical simulations. This shift towards cloud computing can provide many benefits for running surgical simulations on the cloud compared to the traditional way of running surgical simulation applications on-site on bulky and costly equipment. This shift has several advantages, including but not limited to increased collaboration, cost-saving, being independent of platform dependency issues, and remote access [9]. One of the key components of surgical simulations is user interactivity and force feedback through surgical tool interactions. However, currently, cloud computing lacks support for dedicated or attachable specialized hardware support for haptic devices.

Therefore, this work aims to present our solutions for running surgical simulation applications in the cloud environment with integrated haptic devices. These solutions consist of three parts; I) integrating WebRTC [10] for surgical simulations, II) running WebRTC-based surgery simulation on Google Cloud, and III) integrating dedicated surgical hardware tools with haptic integration for high-fidelity interactions. All these components are accessible through web browsers from anywhere, anytime, and with

any device with an internet connection. To the best of our knowledge, no prior work runs surgical simulations remotely in the cloud with hardware integration.

Surgeons' schedules are already busy with cases. For every fellow/resident surgeon under training, the attending surgeon's schedule becomes even busier because the attending surgeon must supervise the critical portions of the surgeries. Attending surgeons usually work on multiple cases in parallel and ensure no critical parts overlap. Adding extra load on surgeons for being physically supervising a fellow/resident surgeon operating on a case in a VR-based surgery simulator is very challenging. This has adversely affected the acquisition of clinical and surgical skills, which is a critical component in training surgical residents. Current surgical simulations require a physical presence of an expert surgeon to supervise a fellow/resident. Using current surgical simulations is a challenge in situations like the COVID-19 pandemic, where physical gatherings were banned.

Most surgical simulations require high-performance computers, which are costly to run on due to the high computations and intensive realistic 3D rendering associated with realistic simulators [8][17]. Aside from that, most surgical simulation applications are platform-dependent. This means that they can only be executed on a specific platform. Cloud computing provides high-end computers with high specifications enabling surgical simulations to run compute-intensive realistic 3D rendering with high-fidelity interactions [5]. Using cloud environments with WebRTC for surgical simulations first eradicates the burden of acquiring high-end computers. Second, it helps solve platform dependency issues associated with the execution of surgical simulations. With the world shifting towards remote work and collaboration, cloud computing and WebRTC provide an avenue to enable remote collaboration from anywhere in the world without requiring the physical presence and the presence of special high-performance equipment to run a simulation. This would provide medical students and surgical fellows/residents an avenue to practice and gain more clinical and surgical skills remotely.

With the surgical simulation applications running on the cloud, all surgical residents and surgeons can use surgical simulations through a low-cost computer, tablet, or smartphone with the internet and a web browser. This is illustrated in Fig. 1 with an arthroscopic view scene from our virtual rotator cuff arthroscopic skill trainer (ViRCAST) [22]. We use ViRCAST as our testbed platform to investigate possibilities of developing surgical simulations over WebRTC and cloud technologies with haptic integration. Furthermore, these technologies may help researchers and developers of surgical simulations to focus on improving the realism of surgical simulations and developing new features without any hardware and resource limitations [9].

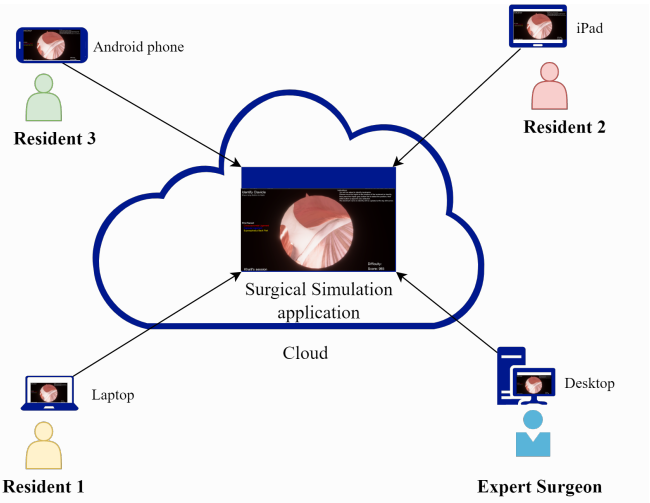



**Fig. 1.** Depicts cloud use for a surgery simulation using different devices at geographically different locations. In the same scene, multiple residents can collaborate with an expert surgeon's guidance and supervision.

There are several arthroscopic surgery-related simulators. A tabletop arthroscopic simulator, Sawbones "FAST" system [15], is similar to a basic arthroscopic surgery trainer, except that FAST only focuses on arthroscopic skills. FAST has a series of interchangeable boards to practice different scenarios such as navigating, triangulating, and merging. It is validated with an opaque dome and can reliably distinguish between experienced and novice surgeons [16].

One of them is the knee arthroscopy simulator developed by Tuijthof et al. [14] is one of them. It aims to provide a complete knee arthroscopy experience. The prototype design allows surgeons to practice meniscus examination, repair, irrigation, and limb extension. They evaluated their simulator to validate the face and content and found it to be an effective simulation of arthroscopic knee surgery, as well as the realism of the arthroscopic movement and mobility.

ViRCAST is our shoulder arthroscopy simulation platform to virtually simulate arthroscopic rotator cuff repair surgeries. ViRCAST platform's simulation components are illustrated in Fig. 2.

All these works provide a method of training surgical residents on performing various arthroscopic surgical procedures but face limitations such as remote collaboration and continuous practice in times of global pandemic, hardware limitations, and platform dependency issues when using these simulations. This study aims to investigate the possibilities of using cloud and WebRTC with haptic integration to alleviate the limitations associated with current surgical simulations.

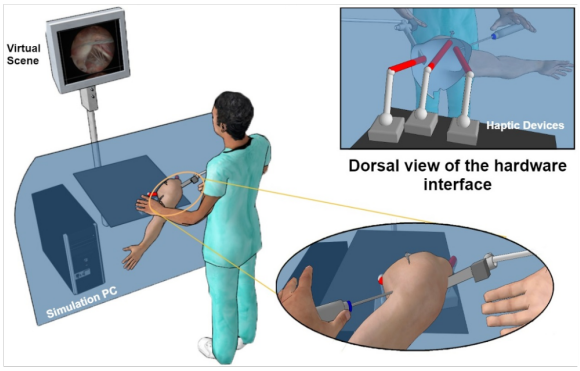



**Fig. 2.** Virtual Rotator Cuff Arthroscopic Skill Trainer (ViRCAST) simulation platform.

## 2 Method

We implemented a cloud-based solution with WebRTC running in the cloud by integrating the solution with our surgical simulation platform, ViRCAST. The architecture overview is illustrated in Fig. 3. The solution involves integrating the surgical simulation with WebRTC and deploying the WebRTC-integrated simulation on Google Cloud with hardware support for interaction and haptic feedback. This section details the design and development methodologies utilized for our cloud-based surgical simulation environment. Our implementation consisted of three core components: a) integration of ViRCAST with WebRTC, b) real-time interaction with ViRCAST through specialized hardware, and c) deploying WebRTC-integrated ViRCAST on the cloud. All these components are architecturally illustrated in Fig. 3.

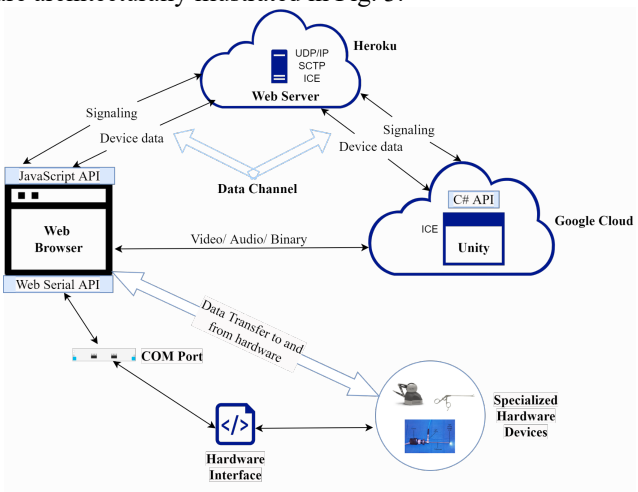



**Fig. 3.** Architectural overview of our implementation.

### 2.1 Integrating WebRTC With a Surgical Simulation

Our surgical simulation application was developed with the Unity Game Engine. In our ViRCAST simulation, we added two cameras to broadcast the scenes to the browser for multiple peers. This allowed each peer to have their own copy of the scene to interact with and a rendering stream to allow the surgical simulation to communicate with the signaling servers. With a C# render streaming script, we specified the signaling type, signaling URL, interactive connectivity establishment (ICE) servers, and an option to use hardware encoding or the default software encoding of the browser. As part of the simulator, we implemented a broadcast stream script in C# to stream media and data via multiple peer connections. This broadcast stream allowed us to attach components that needed to be streamed to multiple peers, such as audio in the browser, through the signaling server. This is illustrated in the Unity portion of Fig. 3.

Using the User Datagram Protocol (UDP/IP), we implemented a WebSocket signaling web server in Nodejs, which creates a peer-to-peer network between Unity and the web browser, which broadcasts the surgical simulation scenes from Unity to the peers connected by the web server through their web browsers (clients). We then implemented a data channel as part of the Nodejs signaling server built on top of stream control transmission protocol (SCTP) to transmit data between the peers and get the data from the specialized surgical tools for interactions with the simulation This is illustrated in the web server portion Fig. 3. Once we integrated WebRTC with our surgical simulation, the next step was solving the integration of the specialized surgical tools for interaction.

### 2.2 Getting Data from The Specialized Hardware to The Surgical Simulation

One of the most essential parts of surgical simulations is the surgical tools used to interact with the simulations, and most of these tools are specialized hardware (arthroscopes and other surgical instruments), as illustrated in Fig. 4. Haptic devices provide force feedback and ways to enable real-time interactions in the simulations [17][18][19][20]. ViRCAST connects real arthroscopic surgery instruments to haptic devices through 3D-printed custom connectors.

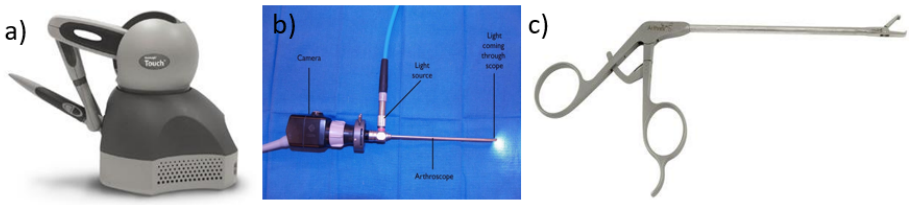



**Fig. 4.** Specialized hardware used in surgical simulations (a) a haptics device, (b) an arthroscope, and (c) a grasper.

In order to provide interactions with our surgical simulation, we had to tackle an important question: “How do you get the data from specialized surgical tools to the simulation in the cloud?” since cloud computing lacks support for specialized surgical hardware tools. Most of these surgical tools are USB enabled and the web browser cannot communicate with USB devices on computers. Due to security reasons, browser manufacturers do not allow browsers to access USB ports and the challenge was to find a way to enable the reading of data from surgical tools.

To solve this issue, we implemented a C++ interface as a middleware between the hardware and the web browser, as illustrated in Fig. 5. The interface uses dynamic link libraries to read float data from the hardware devices, converts it to bytes, and uses the Win32 API to send the data to the serial communication port (COM). The COM port is selected by the user at the first-time execution of the C++ application. Most of the data from these surgical tools are floats and the COM ports do not understand float types, so we converted the floats to bytes before being parsed by COM ports. Each set of data was terminated with a newline character to indicate the end of each set.

Web browsers cannot access COM ports on users' computers due to security reasons. In order to read the hardware data sent to the COM port by the C++ application, we implemented a script using the Web Serial API, which reads the bytes of data and parses it. In order to avoid buffer overflow in the data transfer between the C++ interface and the web browser, both applications use the same baud rate of 115 and 200.

The C++ interface sends data to the serial port through a stream. Streams are beneficial but challenging because they don’t necessarily get all of the data at once; the data may be arbitrarily chunked up. In most cases, each character is on its own line. Ideally, the stream should be parsed into individual lines, and each message shown as its own line. Therefore, we implemented a transform stream, which made it possible to parse the incoming stream of bytes and return the parsed data. A transform stream sits between the stream source (in this case, the hardware), and whatever is consuming the stream (in this case the browser), and transforms the data read from the COM port from bytes to strings before it's finally consumed. Similar to a vehicle assembly line methodology, as a vehicle comes down the line, each step in the line modifies the vehicle, so that by the time it gets to its final destination, it's a fully functioning vehicle. The web serial APIreads each character on its own line, which is not very helpful because the stream should be parsed into individual lines with each data shown as its own line. To do that, multiple transforms can be implemented in the streams, taking a stream in and chunk it based on a user delimiter. In our case, we implemented a Line Break Transform which takes the stream in and chunks it based on line breaks that we inserted in the data in the C++ interface.

Using the data channel in WebRTC, we sent the device data as a JSON file to the surgical simulation application. In the surgical simulation application, we implemented a C# script to read the data from the data channel and process the JSON data into floats. The transformed data was then applied to the surgical simulation enabling real-time interactions through the haptic device(s).

To ensure users are safe, before users can start using the simulation, the web application allows them to pick and connect to the surgical tools to be used in the simulation. This ensures that they give access to the hardware devices themselves instead of the

hardware devices automatically connecting themselves. The architectural overview is illustrated in Fig. 5.

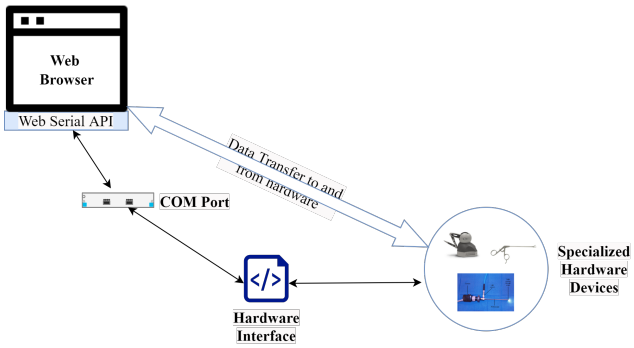



**Fig. 5.** Architectural overview for getting data from the specialized hardware to the surgical simulation.

### 2.3 Deploying WebRTC integrated Surgical Simulation in the Cloud

To test out our haptic integrated WebRTC-based ViRCAST simulation on the cloud, we chose the Google Cloud Platform (GCP) as a choice of cloud platform. With the main aim of running surgical simulations in the cloud fostering remote collaboration, using a Session Traversal Utilities (STUN) for Network Address Translation (NAT) server prevents some users from using the application because of the firewall issues associated with their networks. To solve this, we deployed a COTURN (an open-source implementation for Traversal Using Relays (TURN) around Network Address Translation (NAT) server) server on a Google Compute Engine instance. Traversal Using Relays (TURN) servers help bypass network firewalls. The architecture is illustrated in Figure 6.

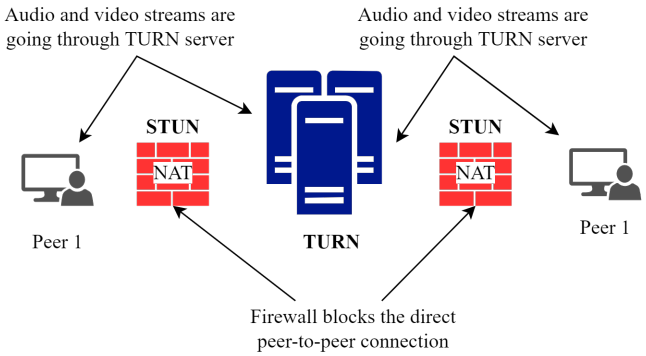



**Fig. 6.** Architectural of how the TURN server allows peers to connect over the firewall

The GCP Compute Engine instance was located at Council Bluffs, Iowa, North America (us-central1-f) with a machine type of e2-micro (2 vCPUs, 1 GB memory, Shared-core VM), 10GB SSD storage and a Bandwidth of 1Gbps running Ubuntu 20. The Unity application and the web server application communicate with the TURN server through Interactive Connectivity Establishment (ICE), so we configured the ICE servers of both the Unity application and web application to point to the external Internet Protocol (IP) address of the COTURN server.

After setting up the COTURN server, we deploy our WebSocket web server for signaling to Heroku's (a platform as a service (PaaS) that enables developers to build, run, and operate applications entirely in the cloud) free-tier dyno. The web server allowed the connection between multiple peers and the surgical simulation remotely. Once the signaling server was deployed, we updated the Unity application with the signaling server's URL from Heroku to enable communication between the Web server and Unity.

The next step of the deployment process is to deploy the WebRTC-integrated ViRCAST. We set up an NVIDIA RTX Virtual Workstation with Windows Server 2019 on Google Cloud located at Council Bluffs, Iowa, North America (us-central1-b). The NVIDIA Workstation Compute instance is configured with the NVIDIA Tesla T4 GPU with 1 GPU, 24 CPUs, 156 GB of memory, 32 Gbps bandwidth, and 50 GB SSD. We then set up firewall rules for the instance to allow Hypertext Transfer Protocol Secure (HTTPS) connections to the instance using Google Cloud's Virtual Private Cloud (VPC) network configuration.

After setting up the Google Cloud Platform (GCP) computer instance, we package the WebRTC-integrated ViRCAST in release mode as an executable and transfer it onto the NVIDIA workstation. We finally execute the surgical simulation application on the NVIDIA workstation running on the Google Cloud and access the surgical simulation from the web application deployed on Heroku.

## 3 Experiment & Results

In this section, we evaluate the performance of WebRTC over real networks. We specifically focus on studying the performance of our ViRCAST simulation with WebRTC and haptic integration over Google cloud instance. We consider two types of WebRTC nodes: (I) a remote wireless node and (II) a remote wired node. The experiment setup can be seen in Fig. 7.

Once the simulation is deployed on the cloud, the two nodes access the simulation through their browsers, and we gather the data for each node's framerate, packet loss, and jitter.

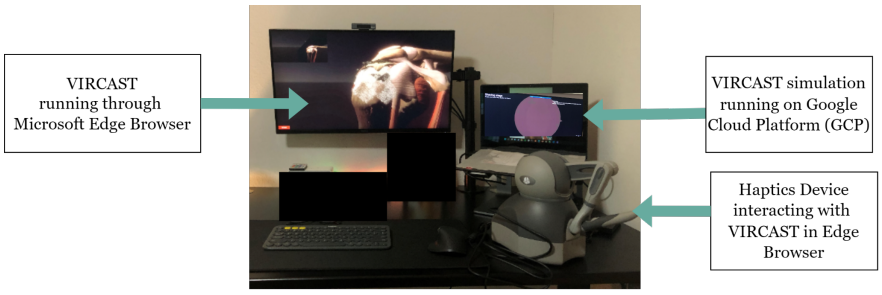



**Fig. 7.** Experiment Setup

### 3.1 Wireless Performance

The wireless node was at a residence in Conway, Arkansas (Location 1) and Lakeland, Florida (Location 2). The node is a Lenovo AMD Ryzen 5 laptop with 12GB RAM running Windows 11. The simulation was accessed through both Google Chrome and Microsoft Edge browsers.

In Tables 1 and 2, we present the number of bitrate, packets sent during each simulation for each browser, and frame rate for the wireless connections.

**Table 1.** Data and packets for wireless nodes gathered for running the simulation at location 1 with a Chrome and Edge browser.

| | **Total** | **Average** | **Vari-ance** | **Browser** | **Frame Rate (fps)** |
|---|---|---|---|---|---|
| **Data (Kbits)** | 533,167 | 1,618 | 0.284 | Chrome | 31 |
| **Packets** | 64,012 | 199 | 0.255 | Chrome | |
| **Data (Kbits)** | 437,494 | 1,298 | 0.36 | Edge | 30 |
| **Packets** | 55,368 | 168 | 0.295 | Edge | |

**Table 2.** Data and packets for wireless nodes gathered for running the simulation at Location 2 with a Chrome and Edge browser.

| | **Total** | **Average** | **Vari-ance** | **Browser** | **Frame Rate (fps)** |
|---|---|---|---|---|---|
| **Data (Kbits)** | 460,076 | 1,250 | 0.395 | Chrome | 33 |
| **Packets** | 58,149 | 162 | 0.319 | Chrome | |
| **Data (Kbits)** | 390,916 | 1,075 | 0.654 | Edge | 28 |
| **Packets** | 49,905 | 140 | 0.575 | Edge | |

For location 1 (chrome browser), there was a drop in packets at the initial start of the simulation and at minute 5, as shown in figure 8a. This was due to the network jitter, as depicted in Figure 8b. This produced drops in frame rates as shown in Figure 8c. Overall, the simulation produced a steady frame rate of 31 fps, as shown in Table 1.

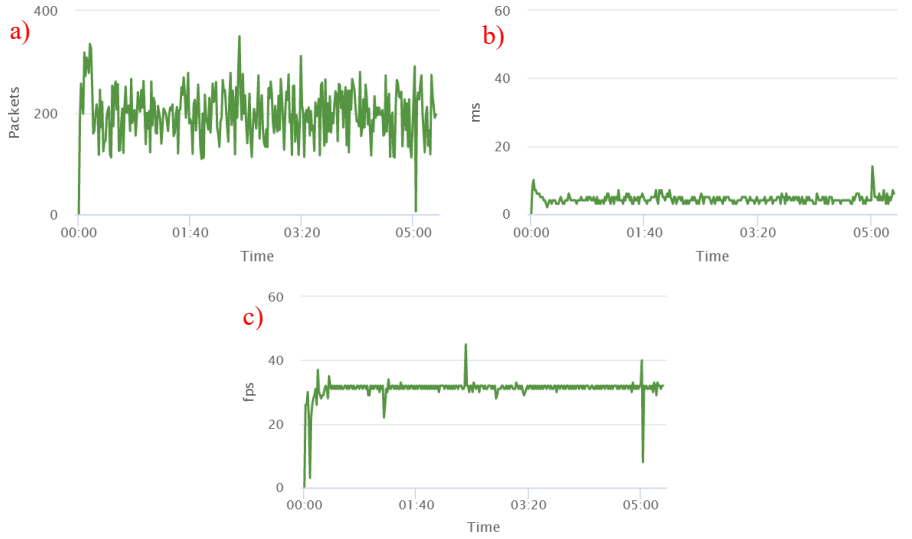
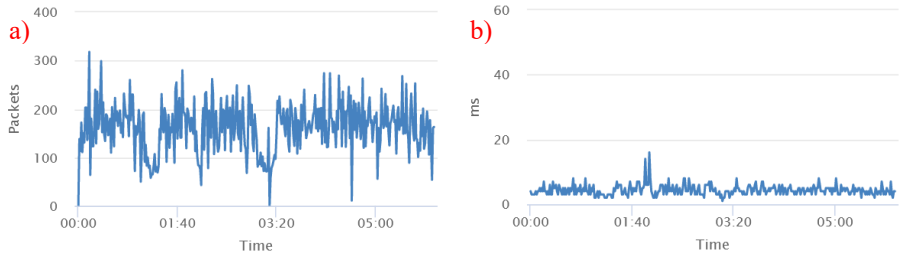



**Fig. 8.** Experimental results of (a) packets, (b) jitter, and (c) frame rate for wireless connection at Location 1 with the Chrome browser.

For location 2 (chrome browser), there were network spikes at minute 1, minute 2, minute 3, minute 4 and minute 5 as shown in figure 9a. This was due to the network jitter as shown in Figure 9b. Overall, the simulation produced a steady frame rate of 33 fps as shown in Table 2 and Figure 9c.



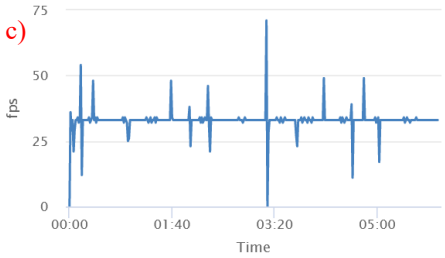



**Fig. 9.** Experimental results of (a) packets, (b) jitter, and (c) frame rate for wireless connection at Location 2 with the Chrome browser.

For location 1 (edge browser), there was a drop in packets at the initial start and at the end of the simulation from minute 4 to minute 5 as shown in figure 10a. This reduced the frame rates as shown in Figure 10c. This was due to the network spikes which produced high jitter as depicted in Figure 10b. Overall, the simulation produced a steady frame rate of 30 fps as depicted in Table 1.

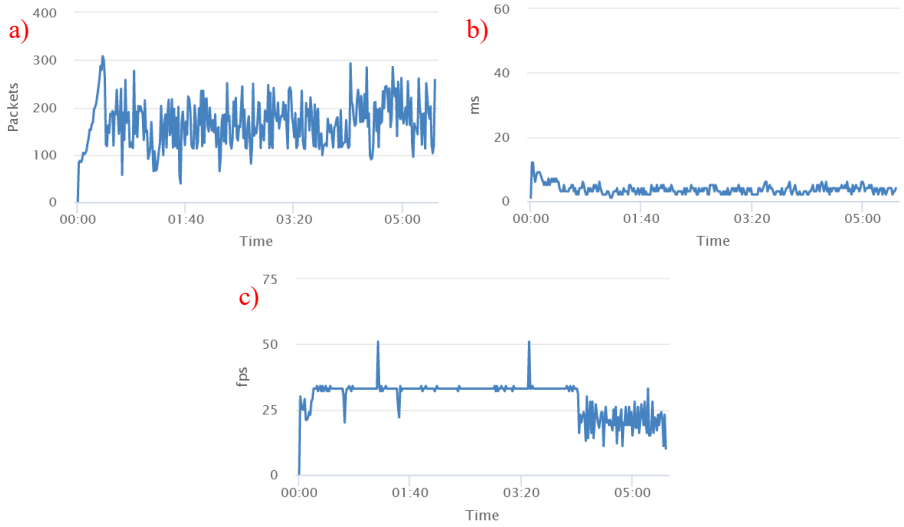



**Fig. 10.** Experimental results of (a) packets, (b) jitter, and (c) frame rate for wireless connection at Location 1 with the Edge browser

For location 2 (edge browser), there was a drop in packets at the initial start and at the end of the simulation from minute 4 to minute 4.60 as shown in figure 11a. This reduced the frame rates as shown in Figure 11c. This was due to the network spikes and traffic which produced high jitter as depicted in Figure 11b. Overall, the simulation produced a steady frame rate of 28 fps as depicted in Table 2. The network jitter at the end of simulation affected the framerate.

We realized the packet losses and jitter at the start of the simulations for all locations which led to drop-in frame rate at the start of the simulations as shown in Figures 8 to

11. We attribute this to the network bursts, and retransmission losses, and the long Round-Trip Time (RTT) for the wireless node.

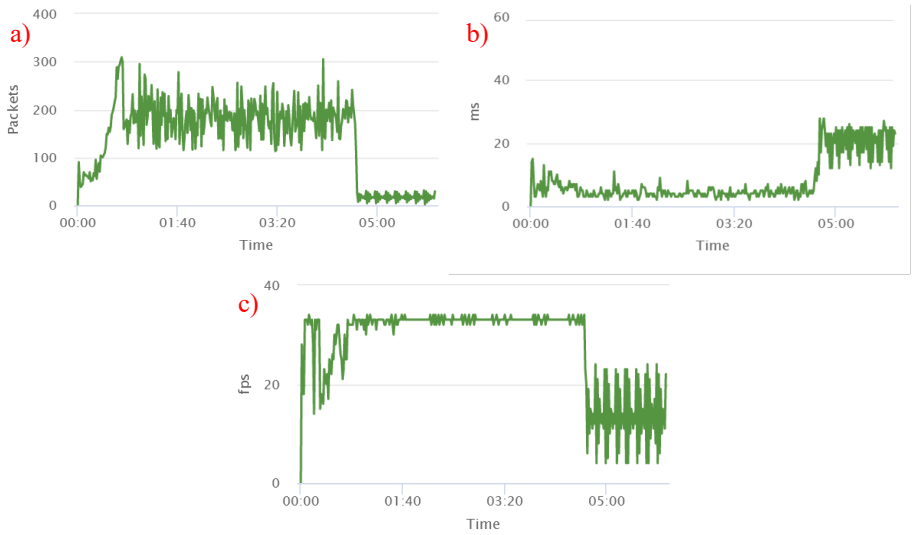



**Fig. 11.** Experimental results of (a) packets, (b) jitter, and (c) frame rate for wireless connection at Location 2 with the Edge browser.

We realized the packet losses and jitter at the start of the simulations for all locations which led to drop-in frame rate at the start of the simulations as shown in Figures 8 to 11. We attribute this to the network bursts, and retransmission losses, and the long Round-Trip Time (RTT) for the wireless node.

### 3.2 Wired Performance

The wired node was set up in our lab in Conway, Arkansas (Location 3). The node is a Supermicro desktop with 65GB RAM running Windows 10. The simulation was accessed through both Google Chrome and Microsoft Edge browsers.

In Table 3, we present the number of bitrate, packets sent, and frame rate recorded during each simulation for each browser for the wired connections.

**Table 3.** Data and packets for wired nodes gathered for running the simulation at location 3 with a Chrome and Edge browser.

| | Total | Average | Vari-ance | Browser | Frame Rate (fps) |
|---|---|---|---|---|---|
| **Data (Kbits)** | 564,140 | 1,592 | 0.313 | Chrome | 33 |
| **Packets** | 67,621 | 195 | 0.271 | Chrome | |
| **Data (Kbits)** | 612,315 | 2,007 | 0.186 | Edge | 33 |
| **Packets** | 70,766 | 237 | 0.175 | Edge | |

For location 3 (chrome browser), there was a drop in packets at the initial start and end of the simulation, as shown in figure 12a. This was due to the network jitter, as depicted in Figure 12b. Frame rates dropped due to the network jitter and drop in packets, as shown in Figure 12c at minute 00:00 and minute 5:30. Overall, the simulation produced a steady frame rate of 33 fps as depicted in Table 3.

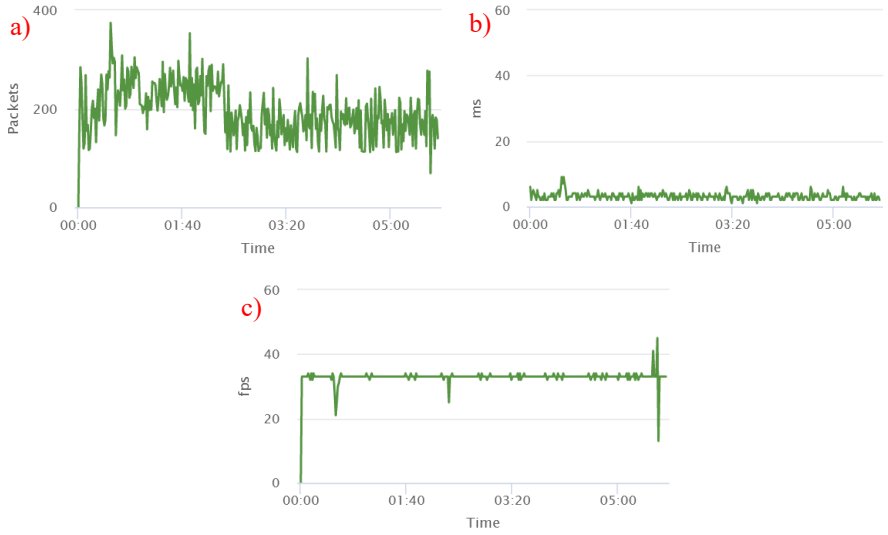



**Fig. 12.** Experimental results of (a) packets, (b) jitter, and (c) frame rate for wireless connection at Location 3 with the Chrome browser.

For location 3 (edge browser), there was a drop in packets at the initial start of the simulation and minute 2:40 as shown in figure 13a. This was due to the network high jitter as depicted in Figure 13b. Frame rates dropped due to the network jitter and drop in packets as shown in Figure 13c. Overall, the simulation produced a steady frame rate of 33 fps as depicted in Table 3.

The small packet loss and jitter led to a steady frame rate of 33 fps during the simulation as shown in Figure 13. We attribute the steady frame rate of 33 fps to the short Round-Trip Time (RTT) for the wired node.

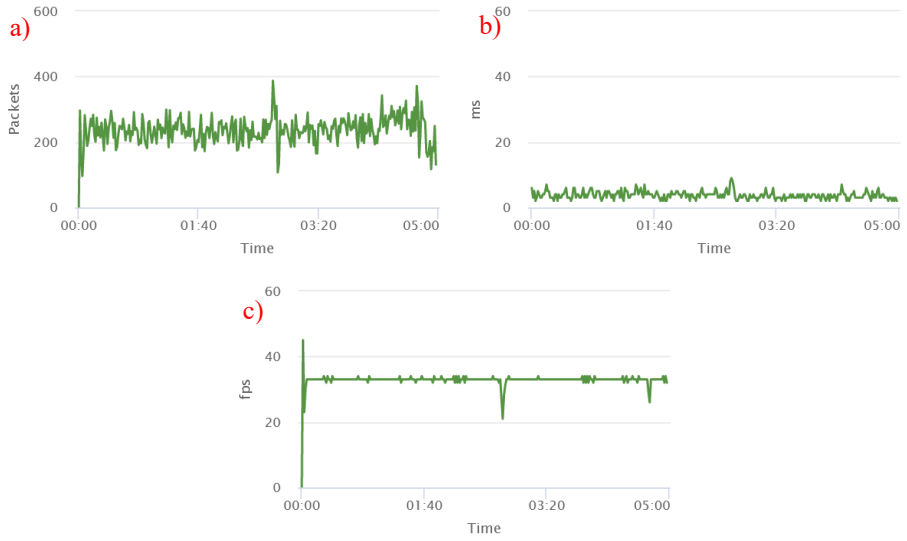



**Fig. 13.** Experimental results of (a) packets, (b) jitter, and (c) frame rate for wireless connection at Location 3 with the Edge browser.

### 3.3 Hardware Lag Measurement

One key component of the research is how the specialized hardware interacts with the surgical simulation in real-time efficiently with minimal lag. A situation where a user interacts with the hardware but response is noticeably delayed in visual feedback is detrimental to any real-time simulation. Therefore, lag has to be tracked and measured. In order to track the lag measurement, we added a timestamp for when the data is sent from the client to the server and timestamp to the server for when it receives the data from the client. During our testing, the lag was unnoticeably small from the beginning to the end of the simulation.

**Table 4.** Sample hardware lag measurement data. We tagged each data with a timestamp from the time the client sent it to when the server received it and applied it to the scene.

| **Time Sent (Client)**<br>**yyyy-mm-dd#hh:mm:ss.SSS** | **Time Received (Server)**<br>**yyyy-mm-dd#hh:mm:ss.SSS** | **Lag**<br>**(ms)** |
|---|---|---|
| 2022-02-28#16:02:03.7 | 2022-02-28#16:02:03.003 | 0.7 |
| 2022-02-28#16:02:03.7 | 2022-02-28#16:02:03.033 | 0.7 |
| 2022-02-28#16:02:03.7 | 2022-02-28#16:02:03.068 | 0.6 |
| 2022-02-28#16:02:03.7 | 2022-02-28#16:02:03.098 | 0.6 |
| 2022-02-28T16:02:03.7 | 2022-02-28T16:02:03.126 | 0.6 |

## 4 Discussion

One key component expected from simulations is the framerate. From our testing and data collection after running the simulation two times at each location for each browser, the wired node streamed at a steady framerate of 33 fps while the wireless connection gave a framerate of 30 fps.

We observed the drop in framerate for the wireless connection. It was due to the long Round Trip Time (RTT), which led to a loss of packets which increased the jitter in the streaming of the simulation. This led to poor video quality for the wireless node at some points. The wired node provided a steady frame rate of 33 fps producing a high video quality due to few packet losses producing less jitter.

In summary, our experiments produce results which proves that using WebRTC in the Cloud is feasible, fosters remote collaboration, and removes hardware limitations. From the research, we realized that the simulation on wired connection produced better video quality, high frame rate with few packet losses and less jitter as compared to the wireless connection. We observed the broadcast of a relatively high-bandwidth transmission over a short period and retransmission losses can degrade the frame rate of wireless networks, especially when the end-to-end RTT (Round-Trip Time) is long.

Moreover, one important aspect of the research was the specialized hardware integration. From our test, the hardware integration for the simulation produced an average initial lag of 0.7 milliseconds, that is the average time data moves from the hardware and interacts with objects in the simulation. This enabled interactions with simulation to be in synchronization with the hardware.

## 5 Conclusion

In this work, we presented a method of developing surgical simulations over WebRTC and cloud technologies with haptic integration. Our subject study was carried out with ViRCAST. Based on the data presented in the study, we determined that our proposed

method was found to be found effective for running surgical simulations. Our experiments revealed that the WebRTC-based simulation had an average frame rate of 33 fps, and the hardware integration resulted in a 0.7 millisecond real-time lag.

For future work, we plan to add live chat for messaging between participants, a recording feature for recording of sessions for later watch, polls for moderators to create polls for participants to vote on during sessions, and a Whiteboard for illustrations and discussions.

**Acknowledgement.** This project was made possible by NIH/NIAMS 5R44AR075481-03 and the Arkansas INBRE program, supported by the National Institute of General Medical Sciences (NIGMS), P20 GM103429 from the National Institutes of Health (NIH), and partially supported by NIH/NIBIB 1R01EB033674-01A1, 5R01EB025241-04, 3R01EB005807-09A1S1, and 5R01EB005807-10.

## References

1. Stone, Robert J. “The (human) science of medical virtual learning environments.” Philosophical transactions of the Royal Society of London. Series B, Biological sciences vol. 366,1562 (2011): 276-85. doi:10.1098/rstb.2010.0209
2. C. D. Combs, "Medical Simulators: Current Status and Future Needs," 2010 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, 2010, pp. 124-129, doi: 10.1109/WETICE.2010.26.
3. D. Morris, C. Sewell, F. Barbagli, K. Salisbury, N. H. Blevins and S. Girod, "Visuohaptic simulation of bone surgery for training and evaluation," in IEEE Computer Graphics and Applications, vol. 26, no. 6, pp. 48-57, Nov.-Dec. 2006, doi: 10.1109/MCG.2006.140.
4. Aikaterini Dedeilia, Marinos G. Sotiropoulos, John Gerrard Hanrahan, Deepa Janga, Panagiotis Dedeilias, Michail Sideris, Medical and Surgical Education Challenges and Innovations in the Covid-19 Era: a Systematic Review, in Vivo Jun 2020, 34 (3 Suppl) 1603-1611; Doi: 10.21873/invivo.11950
5. Chen, SY., Lo, HY. & Hung, SK. What is the impact of the COVID-19 pandemic on residency training: a systematic review and analysis. BMC Med Educ 21, 618 (2021). https://doi.org/10.1186/s12909-021-03041-8
6. Adesunkanmi, A.O., Ubom, A.E., Olasehinde, O. et al. Impact of the COVID-19 Pandemic on Surgical Residency Training: Perspective from a Low-Middle Income Country. World J Surg 45, 10–17 (2021). https://doi.org/10.1007/s00268-020-05826-2
7. Muhammad Osama, Farhan Zaheer, Hasham Saeed, Khadija Anees, Qirat Jawed, Sohaib Hasan Syed, Bashir A. Sheikh, Impact of COVID-19 on surgical residency programs in Pakistan; A residents' perspective. Do programs need formal restructuring to adjust with the “new normal”? A cross-sectional survey study, International Journal of Surgery, Volume 79, 2020, Pages 252-256, ISSN 1743-9191, https://doi.org/10.1016/j.ijsu.2020.06.004.
8. M. N. O. Sadiku, S. M. Musa and O. D. Momoh, "Cloud Computing: Opportunities and Challenges," in IEEE Potentials, vol. 33, no. 1, pp. 34-36, Jan.-Feb. 2014, doi: 10.1109/MPOT.2013.2279684.

9. José A. González-Martínez, Miguel L. Bote-Lorenzo, Eduardo Gómez-Sánchez, Rafael Cano-Parra, Cloud computing and education: A state-of-the-art survey, Computers & Education, Volume 80, 2015, Pages 132-151, ISSN 0360-1315, https://doi.org/10.1016/j.compedu.2014.08.017.
10. B. Sredojev, D. Samardzija and D. Posarac, "WebRTC technology overview and signaling solution design and implementation," 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015, pp. 1006-1009, doi: 10.1109/MIPRO.2015.7160422.
11. Andreatta, P. B., Woodrum, D. T., Birkmeyer, J. D., Yellamanchilli, R. K., Doherty, G. M., Gauger, P. G., & Minter, R. M. (2006). Laparoscopic Skills Are Improved With LapMentorTM Training: Results of a Randomized, Double-Blinded Study. Annals of Surgery, 243(6), 854–863. http://doi.org/10.1097/01.sla.0000219641.79092.e5
12. Laparoscopic warm-up exercises improve performance of senior-level trainees during laparoscopic renal surgery. Lee JY et al , J Endourol. 2012 May;26(5):545-50
13. Laparoscopic basic skills and cholecystectomy VR training curriculum was defined and validated using structured scientific methodology. Aggarwal et al, British Journal of Surgery 2009; 96: 1086–1093
14. Tuijthof, G.J.M., van Sterkenburg, M.N., Sierevelt, I.N. et al. Knee Surg Sports Traumatol Arthrosc (2010) 18: 218. https://doi.org/10.1007/s00167-009-0872-3
15. Goyal, S., Radi, M. A., Ramadan, I. K., & Said, H. G. (2016). Arthroscopic skills assessment and use of box model for training in arthroscopic surgery using Sawbones – "FAST" workstation. SICOT-J, 2, 37. http://doi.org/10.1051/sicotj/2016024
16. Sawbones. (2018). Sawbones FAST Arthroscopy Training System. From https://www.sawbones.com/sawbones-fast-arthroscopy-training-system
17. Z. A. Khan, S. B. Mansoor, M. A. Ahmad and M. M. Malik, "Input devices for virtual surgical simulations: A comparative study," INMIC, 2013, pp. 189-194, doi: 10.1109/INMIC.2013.6731348.
18. Sonny Chan, MS, François Conti, PhD, Kenneth Salisbury, PhD, Nikolas H. Blevins, MD, Virtual Reality Simulation in Neurosurgery: Technologies and Evolution, Neurosurgery, Volume 72, Issue suppl_1, January 2013, Pages A154–A164, https://doi.org/10.1227/NEU.0b013e3182750d26
19. E. Chen and B. Marcus, "Force feedback for surgical simulation," in Proceedings of the IEEE, vol. 86, no. 3, pp. 524-530, March 1998, doi: 10.1109/5.662877.
20. C. Basdogan, S. De, J. Kim, Manivannan Muniyandi, H. Kim and M. A. Srinivasan, "Haptics in minimally invasive surgical simulation and training," in IEEE Computer Graphics and Applications, vol. 24, no. 2, pp. 56-64, March-April 2004, doi: 10.1109/MCG.2004.1274062.
21. Web Serial API - Web APIs | MDN (mozilla.org)
22. Farmer, Jake & Tunc, Mustafa & Ahmadi, Daniel & Demirel, Doga & Halic, Tansel & Arikatla, Sreekanth & Kockara, Sinan & Ahmadi, Shahryar. (2020). Virtual Rotator Cuff Arthroscopic Skill Trainer: Results and Analysis of a Preliminary Subject Study. 139-143. 10.1145/3404663.3404673.