Towards Standardization of Haptic Handshake for Tactile Internet: A WebRTC-Based Implementation

Ken Iiyoshi*, Mahrukh Tauseef*, Ruth Gebremedhin*, Vineet Gokhale, Mohamad Eid

***Abstract*—The rapidly rising interest in Tactile Internet (TI) has lead to the inception of IEEE 1918.1 working group (WG) with a primary objective of envisioning and standardizing various modules crucial for the realization of TI. One of the several core activities of the WG is to standardize haptic codecs for human-in-the-loop applications. This subsumes standardization of haptic handshake scheme for orchestration between heterogeneous nodes for seamless TI interaction. To this end, we present a novel haptic handshake protocol that facilitates exchange of haptic metadata between TI nodes through Tactile Internet Metadata (TIM) scheme. Through WebRTC-based implementation and real haptic devices, we provide a proof of concept of the proposed protocol. The mean and the standard deviation of the handshake latency is measured to be 47.25 ms and 23.38 ms, respectively, thereby making it a strong candidate for employment in TI applications. Finally, we shed light on future refinements to our implementation.**

***Index Terms*—Tactile Internet (TI), handshake, TI Metadata (TIM), WebRTC, request/response.**

I. Introduction

Tactile Internet (TI) [1] is undeniably one of the most promising technological innovations witnessed in the last few decades. TI greatly advances the state of the art in communication (involving audio-video-data transfers) by augmenting *haptic* modality. This enables humans to manipulate and interact with remote objects as if they were physically touching them. This paradigm shift in *perceived telepresence* has unlocked doors to a world of applications potentially impacting every aspect of human life [2]. A few interesting examples include telesurgery [3], remote disaster management, online shopping [4], and virtual reality gaming.

Owing to TI's *human-in-the-loop* characteristic, several crucial challenges need to be addressed before TI becomes completely realizable. The foremost ones include sub-10 ms end-to-end latency and a packet-level reliability of up to 99.9999%. Existing 4G communication standards fall way short of satisfying the above stringent demands of TI [1]. However, considerable progress has been achieved recently in the direction of Ultra-Reliable Low-Latency Communication (URLLC) – a salient feature of 5G communication [5]. TI is expected to significantly benefit from the network-level advancements achieved in the 5G domain.

- *Equal contribution.
- K. Iiyoshi, M. Tauseef, R. Gebremedhin, and M. Eid are with New York University Abu Dhabi, UAE.
  E-mail: {ki573, mt3312, rgg282,mohamad.eid}@nyu.edu.
- V. Gokhale is with University of South Bohemia, Czech Republic.
  E-mail: vgokhale@prf.jcu.cz.

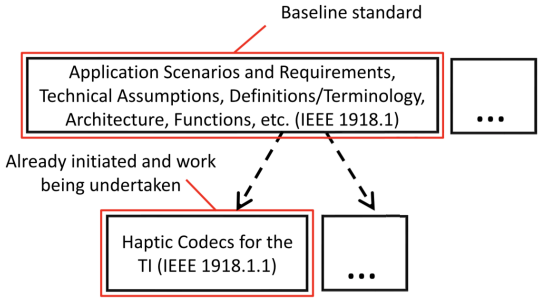


Fig. 1: TI standards WG and its baseline standard as a foundation for future TI standards [2]. IEEE 1918.1 and 1918.1.1 are already initiated.

The presence of haptic feedback introduces several challenges that are unique to TI, and hence need to be separately addressed. For example, codec designs for efficient encoding and decoding of haptic signals, artificial intelligence for TI, inter-media (haptic, audio, and video) as well as intra-media (haptic sensors and actuators) synchronization, and so on. In order to investigate the TI-specific issues, IEEE established P1918.1 TI standards Working Group (WG) [2]. The objective of this WG is to define a standard framework encompassing a generic TI reference model and architecture, in addition to standardizing the interconnections between multitude of interfaces featuring in the framework, as depicted in Figure 1. Further, in order to identify and standardize the TI modules specific to haptic communication, a sub-WG – P1918.1.1 – has been created. This has spawned a string of standardization activities, primarily in two domains: (i) design of *haptic codecs*, and (ii) development of *haptic handshake* scheme.

Significant progress has been made in the domain of *haptic codecs* that aims to standardize data reduction schemes for kinesthetic and tactile feedback for achieving low bitrate haptic communication. In this direction, a hardware and software reference setup was recently proposed with the view of providing an open-source framework for validation of kinesthetic codec proposals [6]. Furthermore, the committee is moving rapidly towards establishing *deadband coding* [7] as one of the standard kinesthetic codecs [8]. Furthermore, several codec proposals for tactile feedback are being solicited and evaluated.

On the other hand, *haptic handshake* – orchestration between widely heterogeneous TI nodes – is still in its nascent stages. In a typical TI setup, the nodes are characterized by non-identical capabilities and requirements in terms of sensing and display, codec compatibility, application needs,

978-1-7281-2355-4/19/$31.00 ©2019 IEEE

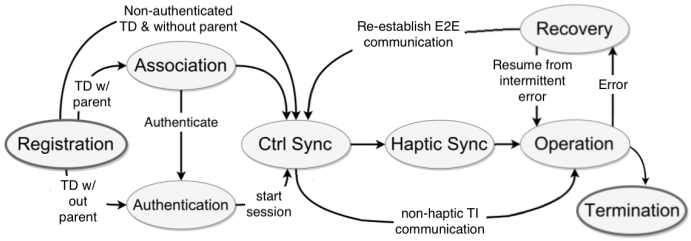


Fig. 2: FSM depicting the deterministic state transitions for a TD establishing, maintaining and terminating E2E communication with another TI component.

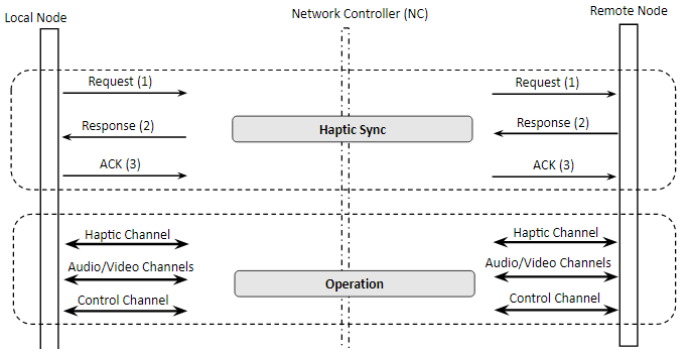


Fig. 3: Schematic of the proposed haptic handshake. Haptic synchronization involves a simple 3-way handshake consisting of request, response, and ack.

user experience, among many others. Before a TI connection is setup, it is imperative for every node to be aware of others' capabilities and requirements so that the adequate amount of information is communicated in an efficient manner. For example, a TI node with both tactile and kinesthetic sensing capabilities can refrain from transmitting tactile information if the receiving TI nodes have no tactile display capabilities. Therefore, it is crucial that the participating TI nodes advertise, through a comprehensive handshaking, their capabilities and requirements, and subsequently arrive at a consensus on related parameters for performing seamless TI interaction. However, standardization of the haptic handshake mechanism is yet to be achieved.

In this work, we take a first step in this direction by proposing a novel haptic handshake scheme. We also present the design of Tactile Internet Message (TIM) – a messaging format for exchanging metadata during haptic handshake. We implement our proposed handshake mechanism based on WebRTC API. Finally, we demonstrate its working through experiments involving real haptic devices.

The remainder of the paper is organized as follows. In Section II, we present the proposed haptic handshake protocol, and the corresponding TIM representation of the haptic metadata. In Section III, we describe the implementation details of the protocol, and in Section IV, we present experimental observation of our proposed protocol. Finally, in Section V we state our conclusions and discuss possible avenues for future work.

II. Proposed Haptic Handshake

In order to connect a Tactile Device (TD) into TI, the steps in Finite State Machine (FSM) presented in Figure 2 are followed [2]. A TD starts in the registration phase, establishing communication with the TI architecture. If the device is part of a larger device, it will associate with that "parent", authenticate, and then start the edge to edge (E2E) control synchronization ("ctrl sync"). If the device is independent, it can start the synchronization with or without authentication, depending on how critical it is. "Ctrl sync" manages connection set up, maintenance of parameters, and audio-video handshaking. If the communication does not involve haptic data, it will move directly to "ctrl sync". If the communication involves haptic data, it will move on to haptic synchronization ("haptic sync") state.

The "haptic sync" state is particularly relevant for this paper as it requires haptic handshake to exchange TIM and ensure that common parameters and codecs are used. Only after this, the actual haptic-audio-video (HAV) communication is managed in the "operation" state. If communication error occurs, the device switches to recovery state, and re-establishes E2E communication if necessary.

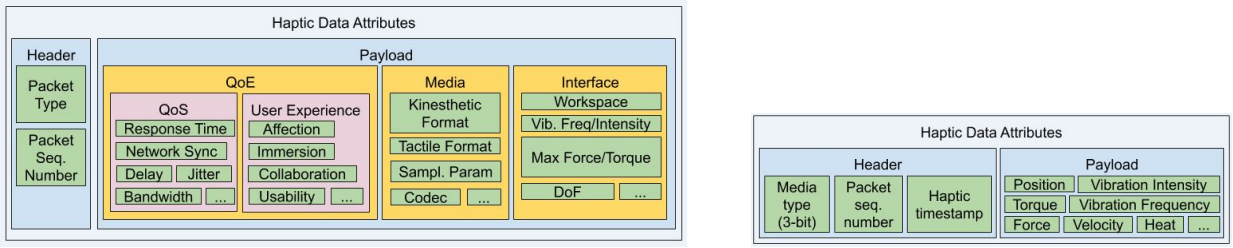
HAV communication often occurs between TDs of diverse capabilities. This necessitates a handshaking procedure for finding a common ground between them based on their TIM specifications (degrees of freedom, workspace, maximum torque/force, etc.). To avoid making application development device and Application Programming Interface (API) specific, a standard device-independent TIM should be exchanged over the handshake before the haptic data is exchanged.

A. Haptic Handshake

As a first step towards standardization of haptic handshake protocol, we created a scheme as shown in Figure 3. The three-way handshake consists of a request from the local node, then a response from the remote node, and finally an acknowledgement from the local node. *Request* serves to send local node's metadata with the specified media format. The remote node identifies any incompatible metadata or format and *response* with alternatives, as well as common metadata. *Acknowledgement (ACK)* confirms this so that communication can begin.

The operation phase consists of haptic, AV, and control channels. AV and haptic data are intentionally handled in separate channels so that third party developers can use different AV codecs based on their own application needs. The control channel is used to deal with dynamic control parameter adjustments during the data communication i.e. change in device or connection environment. This is different from the "ctrl phase" from Figure 2, which manages parameter settings before data communication starts.

For non-haptic TI session in Figure 2, the session invokes Session Description Protocol (SDP) for AV metadata in "ctrl



(a) (b)

Fig. 4: Schematic representation of TIM packet format for (a) request/response, (b) data.

sync" state and Real-time Transport Protocol (RTP) for AV communication in "operation" state. For haptic TI session, as seen in Figure 3, AV metadata exchange occurs in "ctrl sync" state and haptic metadata exchange in "haptic sync" state [8]. The haptic metadata exchange happens separately from AV metadata (SDP). However, haptic media communication may or may not happen with RTP.

B. Tactile Internet Metadata (TIM)

TIM is a message format used in haptic handshake to exchange metadata. WG created a scheme as shown in Figure 4, which displays a *request/response* packet containing metadata and a *data* packet containing haptic media payload [8], both of which consist of header and payload. Each request/response packet contains three payload groups: *Quality of Experience (QoE)*, *media format/parameters*, and *interface options*. The *QoE* group is further subdivided into *Quality of Service (QoS)* and *user experience*. A data packet contains data used in operation stage communication, as well as a timestamp in the header to keep track of accurate real-time data exchange.

Both packets have room for additional data type entries for future applications. As seen in Figure 4, the attributes in the interface options of the request/response packet cater to the limitations of the tactile device by keeping the attributes in the data packet within the lower/upper bounds of the specifications of the device. For instance, the TIM communicates the maximum force attribute during the handshake (via request/response packets) and a value for maximum force is set. If the payload of the data packet is force, it will stay within the bound of maximum force set during the handshake throughout the data exchange. Similarly, the attribute "immersion" could be set to "true" or "false" depending on the application.

III. Implementation of Haptic Handshake

Skype and Google Hangouts have established AV communication protocols. They also support text, which can be modified to support TIM exchange. However, these services are proprietary. Hence, we resort to open source protocols for haptic handshake due to their properties like accessibility, flexibility, and maintenance. Several open source options are available, such as easyRTC [9], WebRTC [10], Jitsi [11], Linphone [12], Jami [13], Riot [14], and Retroshare [15], among many others.

Web Real Time Communication (WebRTC), standardized through World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF) [16], is used in this project. WebRTC is an open-source Web-based Real-Time Communication API designed to enable cross-platform, cross-browser real-time multimedia communication between nodes. WebRTC is designed for peer to peer (P2P) communication as opposed to the conventional server-client architecture, which minimizes network congestion. Our design choice is fueled by the several factors. First of all, WebRTC enables real-time communication of audio, video and data in Web and native applications. This implies that the haptic media can easily be integrated into WebRTC by utilizing the data channel. Furthermore, WebRTC is based on UDP for data communication which fits well haptic data (in order to cope with real-time delivery) and TCP for reliable signalling. Additionally, WebRTC allows flexible control of *RTCDataChannel* by giving options to control maximum packet life and maximum number of retransmissions. Although Transmission Control Protocol (TCP) is more reliable, the tight latency constraints necessitate the employment of UDP for TI applications. WebRTC has an easy to use JavaScript API, which is what has been used in this paper.

The work flow of WebRTC AV communication is shown in Figure 5. In both local and remote nodes, WebRTC first uses the *GetUserMedia* JavaScript API to access any audio or visual devices in the local node. It then invokes *RTCPeerConnection* API to create a UDP peer connection between the local and remote nodes. In order to establish AV handshake, the local node creates an offer and sends it to the remote node. The remote node sets the received offer and creates an answer. Once the local node receives and sets the answer, AV control and communication can begin.

A. Haptic Handshake

Ideally, haptic handshake should proceed simultaneously with AV handshake. However, in order to realize this, the WebRTC architecture requires modifications at multiple layers, making it a tedious task. As an alternative, an application layer protocol is designed that is decoupled from AV handshake. As seen in Figure 5, WebRTC first completes an AV handshake and then spawns one AV session and two *RTCDataChannels*. One of these channels is used to carry out the haptic handshake and the other is used for haptic data communication.

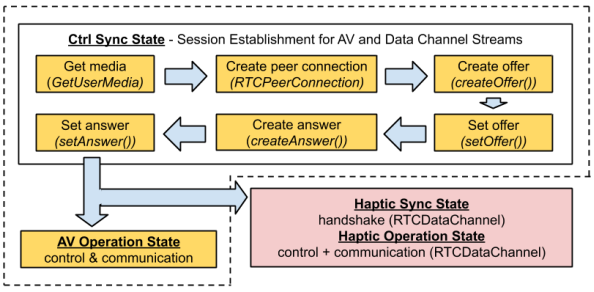


Fig. 5: Working of the WebRTC Program. Both local and remote nodes invoke GetUserMedia and RTCPeerConnection.

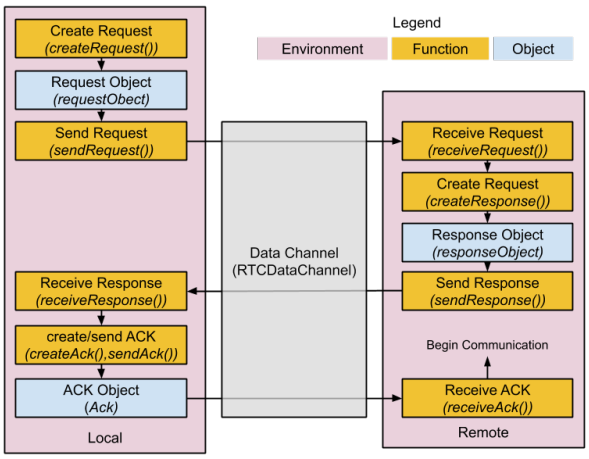


Fig. 6: Request/Response Model inspired by the offer/answer model of SDP [17]

These channels are indicated in Figure 5 as the block outside of the demarcated area which represents existing WebRTC architecture. The area within the demarcation corresponds to the "ctrl sync", while the additional block corresponds to the "haptic sync" (see Figure 2).

The request/response model of the haptic handshake is shown in Figure 6. A *createRequest* function is prompted to make a request object based on TIM. When the request is received at the remote node, the *createResponse* function is prompted. The TIM template of these objects is shown in Figure 7.

The haptic communication is implemented through two *RTCDataChannels*, which are WebRTC's built-in data channels for non-AV data. One channel establishes the haptic session through the haptic handshake. Once the handshake is complete and the session is established, this channel serves as a control channel for the rest of the session. The second channel is opened once the handshake is complete and is used to communicate the haptic data. Both of the data channels are automatically multiplexed by WebRTC.

The following constants were used to configure the two *RTCDataChannels*:

```
const dataChannelOptions = {[[Ordered]],
                            [[MaxPacketLifeTime]],
                            [[MaxRetransmits]],
                            [[DataChannelProtocol]],
                            [[Negotiated]],
                            [[DataChannelId]]};
```

When the parameter *Ordered* is set to true (default), it means choosing a reliable method of communication (leaning towards TCP). For UDP, it is set to false and *MaxRetransmits* is set to 0. The *RTCDataChannel* for haptic control is on TCP mode while the second for haptic data is on UDP mode. Additionally, *RTCDataChannel* can control *MaxRetransmits* or *MaxPacketLifeTime* attributes but not both.

RTCDataChannel is by default negotiated in-band between two nodes. This means that the local node calls *createDataChannel()*, and the remote node connects to the *ondatachannel EventHandler*. This enables a dynamic creation of *RTCDataChannel* where the number of channels is not predetermined. Alternatively, they can be negotiated out of-band, where both sides call *createDataChannel()* with an agreed-upon id to create data channels statically. This method opens the channels with lower latency and has higher stability as the creation of the channels is symmetric.

Based on the above descriptions, the *RTCDataChannel* used for haptic handshake and control was configured to TCP mode and was created statically to ensure stability. The *RTCDataChannel* used for haptic data communication was configured to UDP mode and created dynamically.

The following formats were used to create *RTCDataChannel* for haptic handshake and haptic data, respectively.

```
const dataChannelOptions = {Ordered: true,
                            MaxRetransmits:null,
                            negotiated: true,
                            id: 0};
```

```
const dataChannelOptions = {Ordered: false,
                            MaxRetransmits:0};
```

B. TIM

The TIM template shown in Figure 7 implements the proposal given in Figure 4. It comprises of an object with four different elements: type, session description, media description and codec parameters. The codec parameters are updated based on the codec being used to compress and decompress haptic data in the local and remote environments. The media description is set based on different haptic attributes - Quality of Service (QoS), User Experience (UE), Haptic Interface (*HI*), and other properties (i.e. deadband, sample rate, and tactile frequency). The media description also includes *UA0001* which allows users to add up to 9999 custom attributes (see Figure 7). Following this, the created request is sent to the remote node through *RTCDataChannel*. A time stamp is added and the session counter is incremented in the session description every time a request is sent. The media description

```
{
type: "request" or   response   or "ACK"

sessionDescription:
    "v=0
    o=- <timestamp><sessionVersionCounter> IN IP4 <IPAddress>
    s=Haptic SDP
    i=SDP for Haptic Handshake
    t= 0 0
    a=<add attribute at the session level>"

mediaDescription:
    "m=haptic: <DeviceName><portNumber> SCTP/DTLS HRTP 1
    i=Novint Falcon Haptic System
    a=QoS_hapLatency: <IntegerValue>
    a=QoS_hapJitter: <IntegerValue>
    a=QoS_hapReliability: <IntegerValue>
    a=UE_immersion: <Boolean 0 or1>
    a=UE_collabortation: <Boolean 0 or 1>
    a=UE_satisfaction: <Boolean 0 or 1>
    a=UE_presence:  <Boolean 0 or 1>
    a=Hap_Deadband: <Boolean 0 or 1>
    a=Hap_kinSampleRate: <IntegerValue>
    a=Hap_tacFequency: <IntegerValue>
    a=HapI_dof: <NaturalNumberValue>
    a=HapI_ws_x_y_z: <IntegerValueofx><IntegerValueofy><IntegerValueofz>
    a=HapI_fr_x_y_z: <IntegerValueofx><IntegerValueofy><IntegerValueofz>
    a=HapI_tr_x_y_z: <IntegerValueofx><IntegerValueofy><IntegerValueofz>
    a=UA_0001 (ad custom attributes here...)"

CodecParams:
    "RecordSignals=0;                // 0: Recording off, 1: Recording on
    ForceDeadbandParameter=0.0;      // for force data reduction
    VelocityDeadbandParameter=0.0;   // for velocity data reduction
    PositionDeadbandParameter=0.0;   // for position data reduction
    ForceDelay=0;                    // Constant force network delay
    CommandDelay=0;                  // Command channel constant delay
    ControlMode=1;                   // 0: position, 1: velocity
    FlagVelocityKalmanFilter=0;      // 0: disabled, 1: enabled
    LocalIP=127.0.0.1;              //local node
    RemoteIP=127.0.0.2; "             //remote node
}
```

Fig. 7: Template of Request/Response/ACK object inspired by the textual format of SDP [18]

parameters are configured to allow effective communication between two haptic devices. These parameters are updated to meet the specifications (i.e. device type, latency, degree of freedom, jitter, etc.) of both devices. The *CodecParams* of the response is set to the codec parameters received in the request. Ultimately, the response is sent back to the local node. The session counter in the session description of the response is incremented and a timestamp is added. Then, an acknowledgement is created and sent to the remote user. The acknowledgement *ACK* object comprises of type and session description only. Once the acknowledgment is sent by the local user, the haptic data communication is prompted.

IV. Haptic Handshake Experiments

In this section, we discuss a basic demonstration of our haptic handshake protocol, and present the handshake latency measurements.

A. Basic Setup

For easier realization of the proposed haptic handshake protocol, we emulated two nodes (one acting as local node and the other as remote node) by implementing WebRTC in two browser instances on the same desktop machine. Each browser, where AV data is displayed, has direct access to cameras and microphones. A demonstration of the handshake protocol where a local node and remote node were setup on local host with a P2P communication can be seen in Figure 8.

In addition to AV data, the browsers have to obtain haptic data from the C++ based devices. WG's reference C++

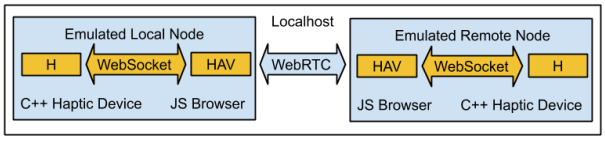


Fig. 8: Implementation of the proposed haptic handshake using WebRTC. JavaScript browsers require TCP for WebSocket.

software and hardware is used to set up and acquire haptic data [19]. Ideally, a web browser using WebRTC API would be able to access the haptic device directly in the same manner it accesses cameras and microphones. However, haptic devices are not yet registered by web browsers. Thus haptic communication between the haptic device and browser is done in WebSockets, which uses TCP. Adam Rehn's C++ WebSocket Server Demo is used for this while efforts are being made to integrate it to the handshake implementation [20].

Implementing the proposed haptic handshake protocol on two separate devices requires signaling, which is beyond the scope of the protocol. This is because different signaling protocols can be used depending on user needs. That being said, some of the standard options for signaling are WebSockets and Session Initiation Protocol (SIP).

B. Demonstration and Measurements

The web page demonstrating the haptic handshake is shown in Figure 9. The GUI allows the user to choose the audio, video, and haptic source devices before setting up the session. Once a WebRTC session is established after "set Answer", the haptic handshake can be started. The produced Request/Response TIM are displayed to the user. The user can also choose a different device (AV or haptic) during an HAV session which will trigger the haptic handshake again. This will address potential changes in codecs and formats. The structure of this web page was inspired by the WebRTC Sample Munge SDP [21].

The performance of the haptic handshake is measured through its mean latency. The difference between time stamps at the beginning and end of the haptic handshake is computed. The mean latency is calculated from 20 iterations of the handshake and is found to be 47.25 ms with 23.38 ms standard deviation. Lower latency is expected in future iterations as the native WebRTC API will be used.

V. Conclusions and Future Work

In this section, we will state our conclusions, and discuss avenues for future research.

A. Conclusions

In this paper, we presented the design of a haptic handshake protocol as a first step towards standardization of handshake scheme. We described in detail Tactile Internet Metadata (TIM) devised for conveying the haptic metadata of various TI nodes. Through implementation of the proposed handshake

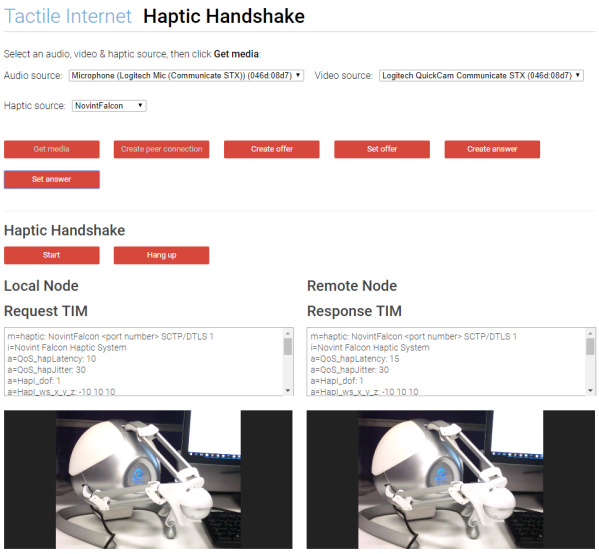


Fig. 9: GUI for allowing the user to choose among different configurations before initiating the haptic handshake.

protocol on a WebRTC-based platform and real haptic devices, we provide a proof of concept of its operation. Further, the mean and standard deviation of the haptic handshake latency is measured to be 47.25 ms and 23.38 ms, respectively, thereby substantiating its usefulness for TI applications.

B. Future Work

The current implementation is only for Novint Falcon haptic devices. This should be expanded to other haptic devices, and the implementation should be enhanced to support dynamic control i.e. switching haptic devices during communication. Congestion control should also be incorporated.

Switching from JavaScript Web API to native C++ implementation would address many issues as well. First, this would mean synchronization of AV data with haptic data, rather than using *RTCDataChannel* that prevents this. It will also allow the haptic device setup to be integrated with WebRTC. This will cut any unnecessary cross-language latency.

Since Websockets add an additional delay to the communication of Haptic data, it can be replaced with text files for reading and writing haptic data. It is yet to be checked if two programs can edit the file simultaneously, such as when two processes are sharing a file descriptor. Nonetheless, the implementation itself should be simple. Another alternative is to use netcode.io with browser extension, as they use secure UDP. This is yet to be tested [22].

The code interfacing haptic devices to browsers should be object oriented in order to increase portability for testing applications other than the kinesthetic codec. In conjunction with network simulators such as NS-3, the system can also be used for evaluating third party HAV communication applications under various networking environments.

Ultimately, the communication system, including WebRTC, should be implemented solely in C++. By doing so, the browser program and the haptic device-managing program can be merged together, minimizing latency, as shown in Figure 10. This is left for future work.

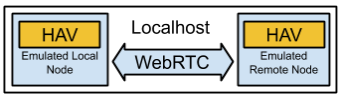


Fig. 10: Ideal Implementation of Tactile Codec.

REFERENCES

[1] G. P. Fettweis, "The tactile internet: Applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, March 2014.

[2] O. Holland, E. Steinbach, R. V. Prasad, Q. Liu, Z. Dawy, A. Aijaz, N. Pappas, K. Chandra, V. S. Rao, S. Oteafy, M. Eid, M. Luden, A. Bhardwaj, X. Liu, J. Sachs, and J. Arajo, "The ieee 1918.1 tactile internet standards working group and its standards," *Proceedings of the IEEE*, vol. 107, no. 2, pp. 256–279, Feb 2019.

[3] R. J. Anderson and M. W. Spong, "Bilateral control of teleoperators with time delay," *IEEE Transactions on Automatic control*, vol. 34, no. 5, pp. 494–501, 1989.

[4] R. de Vries, G. Jager, I. Tijssen, and E. H. Zandstra, "Shopping for products in a virtual world: Why haptics and visuals are equally important in shaping consumer perceptions and attitudes," *Food quality and preference*, vol. 66, pp. 64–75, 2018.

[5] M. Series, "Imt vision–framework and overall objectives of the future development of imt for 2020 and beyond," *Recommendation ITU*, pp. 2083–0, 2015.

[6] A. Bhardwaj, B. Cizmeci, E. Steinbach, Q. Liu, M. Eid, J. AraUjo, A. El Saddik, R. Kundu, X. Liu, O. Holland *et al.*, "A candidate hardware and software reference setup for kinesthetic codec standardization," in *2017 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE)*. IEEE, 2017, pp. 1–6.

[7] P. Hinterseer, S. Hirche, S. Chaudhuri, E. Steinbach, and M. Buss, "Perception-based data reduction and transmission of haptic data in telepresence and teleaction systems," *IEEE Transactions on Signal Processing*, vol. 56, no. 2, pp. 588–597, 2008.

[8] E. Steinbach, M. Strese, M. Eid, X. Liu, A. Bhardwaj, Q. Liu, M. Al-Jaafreh, T. Mahmoodi, R. Hassen, A. El Saddik, and O. Holland, "Haptic codecs for the tactile internet," *Proceedings of the IEEE*, vol. 107, no. 2, pp. 447–470, Feb 2019.

[9] D. Pelton, "Easyrtc framework tutorial," 2013.

[10] A. B. Johnston and D. C. Burnett, *WebRTC: APIs and RTCWEB protocols of the HTML5 real-time web*. Digital Codex LLC, 2012.

[11] E. Ivov, "Jitsi," *The architecture of open source applications*, pp. 121–132, 2011.

[12] Linphone. [Online]. Available: https://www.linphone.org/

[13] Jami. [Online]. Available: https://jami.net/

[14] Riot. [Online]. Available: https://about.riot.im/

[15] Retroshare. [Online]. Available: https://retroshare.readthedocs.io

[16] S. Loreto and S. P. Romano, "Real-time communications in the web: Issues, achievements, and ongoing standardization efforts," *IEEE Internet Computing*, vol. 16, no. 5, pp. 68–73, Sep. 2012.

[17] J. Rosenberg and H. Schulzrinne, "An offer/answer model with session description protocol (sdp)," 2002.

[18] M. Handley, C. Perkins, and V. Jacobson, "Sdp: session description protocol," 2006.

[19] IEEE P1918.1.1 Haptic Codecs for the Tactile Internet Task Group. (2018) Kinesthetic reference setup. [Online]. Available: https://cloud.lmt.ei.tum.de/s/8ol5mX6TCDBS8t4

[20] A. Rehn. (2019) Websocket server demo. [Online]. Available: https://github.com/adamrehn/websocket-server-demo

[21] Webrtc samples munge sdp. [Online]. Available: https://webrtc.github.io/samples/src/content/peerconnection/munge-sdp/

[22] The Network Protocol Company. (2019) netcode.io. [Online]. Available: https://github.com/networkprotocol/netcode.io