HAVE'2007 - IEEE International Workshop on
Haptic Audio Visual Environments and their Applications
Ottawa, Canada 12-14 October 2007

ALPHAN: Application Layer Protocol for HAptic Networking

Hussein Al Osman[1], Mohamad Eid[1], Rosa Iglesias[2], and Abdulmotaleb El Saddik[1]
[1]*Multimedia Communications Research Laboratory - MCRLab*
School of Information Technology and Engineering - University of Ottawa
Ottawa, Ontario, K1N 6N5, Canada
[2]Ikerlan Technological Research Centre
P°. J. Mª. Arizmendiarrieta, 2
20500 Arrasate-Mondragón, Gipuzkoa, Spain
[1]*{HALOsman, eid, abed} @ mcrlab.uottawa.ca,* [2] riglesias@ikerlan.es

Abstract – *The transmission of haptic information over the network has recently received significant attention. The wide spectrum of haptic applications makes it difficult to capture the widely varying requirements of these applications into one generic protocol. In this paper, we introduce ALPHAN (Application Layer Protocol for Haptic Networking), a novel application layer protocol for haptic communication. The distinguished features of this protocol are the followings: first, being an application layer protocol, it can easily be customized according to the application requirements and needs; second, the protocol utilizes an XML-based description to define its networking parameters; and third the protocol supports multi-buffering where priorities are attributed to different buffers in collaborative haptic environments. The implementation and preliminary evaluation of the protocol showed a satisfactory performance. We also present our immediate future development and possible research avenues.*

Keywords – *Haptics, Application layer protocol, HAML, collaboration, networking.*

1. INTRODUCTION

A. Haptics

Haptics is a multidisciplinary research that studies manual sensing (exploration for information extraction) and manipulation (for modifying the environment) through touch [1]. With the growing area of haptics many researchers have fostered their interest to integrate such a technology in networked virtual environments [2-4]. The community uses interchangeable terminology to describe the research of communicating haptic data over a network such as tele-haptics [2], shared haptic environments [3], or collaborative haptic audio visual environments (C-HAVE) [4]. Throughout this paper we will use the term C-HAVE to refer to haptic environments involving two or more users and distributed over a network.

The potential of C-HAVE is significant for wide spectrum of applications including distributing training or tele-mentoring and surgical simulations [5], entertainment and gaming [6], computer-mediated social interaction [7], visual art and museums [8] and assembly planning [9], among others.

On the other hand, a C-HAVE poses new challenges at both the application level and the communication (networking) level. The haptic interaction requires simultaneous and interactive input and output with the haptic device at extremely high update rate (normally 1 kHz). At the application level, consistency maintenance, access control, transparency, and stability are experiencing extensive research. At the networking level, Quality of Service (QoS) parameters such as the network latency, jitter, packet loss, scalability, and compression are key aspects that have been investigated and researched. This paper presents an application layer protocol that enables the application to define and pass the networking parameters and requirements and, adapt the communication accordingly. The application network requirements are passed using the Haptic Applications Meta-Language (HAML) [10]. HAML is an XML-based, technology-neutral description language for haptic models and interactions.

B. Related Work

Very few haptic communication protocols have been designed since it is difficult to capture the widely varying requirements of different haptic applications into one generic protocol. First, the generic transport layer protocols (namely TCP and UDP) have been shown not suitable to use for haptic data communication [11]. Nonetheless, UDP remains the favorable protocol for real time applications and thereby a foundation for haptic data communication.

The Synchronous Collaboration Transport Protocol (SCTP), proposed in [12], sends "normal messages" unreliably whereas key messages are sent reliably using sequence numbers. For each key message, a timer is set. If a timeout occurs before receiving the acknowledgement, it is resent; always as a key update (positive acknowledgment). The protocol was not designed specifically for haptic data communication. An extension for SCTP was proposed in [13] and named the Smoothed

SCTP. The extension involves adding a jitter smoothing mechanism to the SCTP protocol. On the sender side, the Smoothed SCTP and SCTP protocols behave exactly the same. On the receiver side, each received update is placed in a bucket according to its timestamp. The receiver constantly checks for updates that have been sent for, lets say T interval, and in case an update is found, it is retrieved from the bucket and forwarded to the application. This means that all updates, including the ones generated locally, are processed with a T delay.

The Light TCP [11], as the name suggests, is inspired from TCP yet avoids several TCP drawbacks. To support the concept of message obsolescence, the sender side queue accepts update messages from the application and processes them as follows: (1) A key update is placed at the end of the queue and marked as a key message to prevent it from being erased; (2) A normal update can replace older normal updates for the same shared object; (3) Unacknowledged update messages are placed again in the queue if no newer updates from the same object have been generated. A received update is forwarded to the application if its sequence number is bigger than the last received update's sequence number, otherwise it is dropped (no buffering).

Unlike the previous protocols, the Real-Time Protocol for Interactive applications (RTP/I) is an application layer protocol for network distributed interactive applications (not specifically for haptic applications). The messages exchanged between participants are categorized as: event, state and request-for-event packets [14]. In its quest to support the widest spectrum of applications, RTP/I adds unnecessary communication overhead. The extra overhead can not be afforded in haptic applications due to the very stringent real time requirements. For instance, this protocol performs application layer framing where the header alone is 28 bytes. At the typical haptic rate of 1000 kHz, this means that 28 kbytes/second of header information that are unnecessary for haptic applications.

C. Project Goals

The current paper presents the development of a haptic communications protocol. The protocol, unlike many of its predecessors, operates on the application layer. Different haptic applications, such as gaming or medical applications have dissimilar network requirements. A protocol at the application layer can be dynamically adapted to suit these application requirements. For instance, in the case where local lag is used, the lag value can only be efficiently determined if the maximum allowable response time is known, which in turn, widely varies from one application to another. A protocol on the application layer has better knowledge of the objects in the environment. Such knowledge is exploited in order to allow for a more efficient communication.

The remainder of the paper is organized as follows: in section 2, we present the description of ALPHAN and introduce the multi-buffering concept. Section 3 describes the implementation architectural details. In section 4 we discuss how the haptic data was simulated and present a preliminary performance evaluation. Finally, in section 5 we summarize our findings and provide potential improvements and future work.

II. ALPHAN COMMUNICATION PROTOCOL

A. Application Layer Protocol

The ALPHAN protocol operates on the application layer. To further improve the customizability of the protocol, the generic QoS parameters associated with a specific haptic application are listed and defined in HAML format (such as, the maximum allowable response time and jitter and packet loss, shared objects priorities, security, trustworthiness and availability). The protocol parses the HAML file and sets up the communication session accordingly. The HAML descriptions are assumed to be provided by the application itself that may include other aspects of the application such as the graphic and haptic models and rendering, haptic interfaces, among others.

B. Multiple Buffer Scheme

The ALPHAN protocol operates on top of UDP. UDP is chosen for its simplicity and speed. It does not impose any reliability schemes that are not usually needed in real time applications. Additionally, ALPHAN supports the notion of key updates by implementing an application layer reliability mechanism that is only applied to such updates while "normal updates" remain unaffected.

In this protocol, we propose a multiple buffering scheme where each object in the environment, whether it is a haptic or a graphic object, is attributed a sending buffer. Each sending buffer consists of two queue data structures, a sending queue and a retransmission queue. The sending queue holds all the updates to be sent over the network. The retransmission queue is somewhat similar to the TCP retransmission queue; it holds copies of the key updates from the time they are sent, until they are acknowledged. This enables re-sending the key update message in case a retransmission timeout occurs before the acknowledgement is received. Allocating a buffer for each object permits the decoupling of update transmission for different objects, especially if they are independent from each other.

Attributing a buffer for each object has two main advantages. Firstly, it is possible to attribute a priority for each buffer. The priority of the buffer corresponds to the importance of the object in the environment. Objects with higher priority will get precedence during the transmission of updates. This provides the protocol with a greater degree of customizability. Secondly, attributing a retransmission queue for each object makes the protocol more flexible. In the single buffer approach, copies of all the key updates that are transmitted are queued in the same retransmission queue. If the sending data rate is high enough, the queue will be full of obsolete key updates in the retransmission

0										1										2										3	
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1
V		H	C	E	Type			Payload Type								Sequence Number															
Participant ID																Object ID															
Fragment Count																Length															
Timestamp																															
Data																															

Figure 1: ALPHAN header format

queue since the application has probably produced newer ones. For graphics data, the newest key update is the most important one, since it holds the new state of the object. On the other hand, discarding an important haptic key update message may result in producing higher than intended forces and thus instabilities in the haptic display, on the receiver side. Therefore, for graphic objects the size of the retransmission queue can be set to one whereas for haptic objects (mostly objects that represent the haptic device proxy) the size can be varied according to the object's nature and application needs.

C. Application Layer Reliability

The application layer reliability is implemented only for key updates. When a key update is sent, a timer is started; if a timeout occurs before the receipt of the acknowledgement, it is resent again. Waiting for the correct amount of time before resending the packet can be crucial for efficient transmission. In order to adeptly estimate the retransmission timeout value, the round trip delay between the sender and the receiver is measured as in [15].

C.1 Packets I, P and B

The principle of I, P and B packets is partly borrowed from the MPEG standard. The I packets are sent reliably and are considered to be key packets. P packets are encoded differentially based on previous I or P packets while B packets are encoded based on previous and future I and/or P packets. P and B frames are considered to be "normal updates" and therefore are sent unreliably. The rate at which I and P packets are sent can be optimized to suit the network condition. Such form of encoding can definitely save significant network resources.

Furthermore, in order to reduce the amount of packets transmitted over the Internet, a signal prediction model can be used. This model is characterized by the Just Noticeable Difference (JND) linear threshold. Whenever the difference between the predicted signal and produced signal surpasses the JDN, an update is sent to the other participants. The authors in [16] proposed a linear predictive model defined simply by equation (1) where {vi, vi-1, vi-2 …} are the most current values produced by the model and {ti, ti-1, ti-2 …} are the corresponding timestamps. {vnew, vnew-1, vnew-2,…} are the last received updates and {tnew, tnew-1, tnew-2, …} are the corresponding timestamps.

$$v_i = \begin{cases} v_{new} & new - value - arrived \\ v_{i-1} + \dfrac{v_{new-1} - v_{new-2}}{t_{new-1} - t_{new-2}}(t_i - t_{i-1}) & else \end{cases} \qquad (1)$$

C.2 Local Lag

The local lag scheme is proposed in this protocol for consistency maintenance. In order to maintain the generality of the protocol, this feature is kept optional. Other consistency schemes, such as dead reckoning, could be easily implemented on top of ALPHAN.

Nonetheless the use of local lag not only facilitates consistency maintenance but also helps smoothing jitter. Choosing the most efficient local lag value is very important when using this scheme. As we have previously discussed, this choice is dependent on the delay in the network and on the maximum allowable response time. The latter can be read from the HAML description file of the haptic application and thus the protocol can be dynamically customized to fit the application needs. .

C.3 Application Layer Framing

As we have already mentioned, ALPHAN operates on the application layer and thus application layer framing is performed. Figure 1 presents the haptic frame header format as defined in the current version. A summary of the frame fields and their descriptions are listed in Table 1.

III. IMPLEMENTATION

A. Implementation Architecture

A.1 Overview

Based on ALPHAN's specifications presented in the previous section, we have developed a code library that implements the core functionality of the protocol. The library is developed in C++. The C++ programming is chosen, as opposed to other higher level programming languages like Java and C#, for its speed, especially when it comes to multithreaded programs. C++ also enables access to low level constructs and thus provides better control over the application's performance.

Figure 2 shows a high level view of the library architecture. The **Protocol Manager Module** thread is at the basis of the architecture and most other modules communicate with it. The **Protocol Manager Module** manages the library and interfaces directly to the application. The **Simulation Module** holds both the graphic and haptic threads. The **Sender Module** thread holds the sending buffers. Updates that are placed in buffers are sent as soon as possible according to their priority. The **Receiver Module** thread is responsible for receiving updates and decoding them. If a key update is received by this module, an acknowledgment is produced and buffered in the acknowledgment queue where it is retrieved by the **Sender Module**. All the updates received by the **Receiver Module** thread are buffered in the **Update Relayer** buffer. The **Update Relayer Module** thread is responsible for relaying packets to the **Protocol Manager** at the appropriate time. Note that the appropriate time is inscribed in the timestamp of the update. Such mechanism is necessary for the implementation of the local lag algorithm for consistency purposes or other jitter smoothing algorithms. The **Network Sensing Module** thread is responsible for calculating the Round-Trip Time (RTT) value and detecting any disconnections. The **NTP (Network Time Protocol) Module** is used to synchronize the simulation timer with the timers of the other participants' simulations.

Table 1: Frame fields description

Field	Bits	Description
V	2	The version of the protocol; set to 0 for the current version
H	1	This bit is set if the data is haptic, and reset if the data is graphics
C	1	Reporting collision state: the payload has position and force information if 1, position only if 0
E	1	Not relevant for haptics. Indicate the last fragment of a packet for graphics data
Type	3	The type of haptic data: I, P, B frame and state query types for late comers
Payload Type	8	Indicates single/multiple point of interaction for haptics and object type for graphics (rigid, deformable, etc.)
Sequence Number	16	Identify each packet and see if any packets were lost or delivered out of sequence
Participant ID	16	Identifies each participant in the C-HAVE environment.
Object ID	16	Identifies each object in the C-HAVE environment.
Fragment Count	16	Indicates the number of haptic updates carried in one packet. Identify the number of fragments for graphic data
Length	16	Indicates the length of the payload
Timestamp	32	Time at which the event in the payload should be executed. Important for the implementation of a local lag mechanism.

A.2 Sender Module

The ***Sender Module***'s behavior is represented in the state machine in Figure 3. The Sender stays in the idle state until one of the following events occur:

- **Receive a key update from buffer:** a timer is started and a copy of the update is placed in the retransmission queue before sending it. Placing the key update in the retransmission queue is essential in order to be able to resend it if not acknowledged.
- **Receive a normal update from buffer:** the update is sent.
- **Receive a timeout signal:** check to what update the timeout belongs, retransmit the corresponding update and re-place it in the retransmission queue.

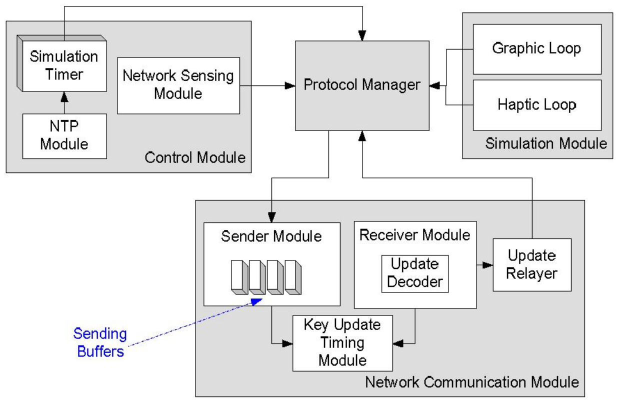


Figure 2: High level view of the library architecture

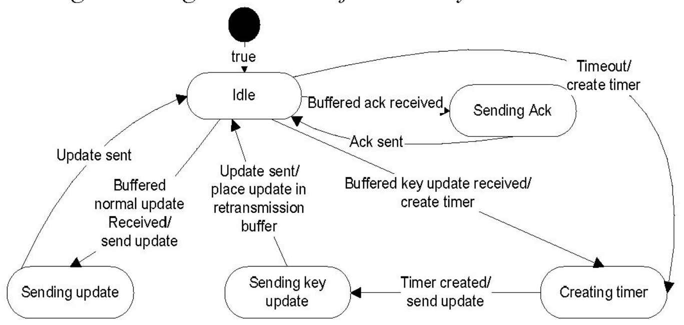

true
Idle
Timeout/ create timer
Buffered ack received
Sending Ack
Ack sent
Update sent
Buffered normal update Received/ send update
Update sent/ place update in retransmission buffer
Buffered key update received/ create timer
Sending update
Sending key update
Timer created/ send update
Creating timer

Figure 3: Sender module state machine diagram

A.3 Receiver Module

The **Receiver Module**'s behavior is represented in the state machine in Figure 4. The Receiver remains in the waiting mode until one of the following events occurs:

- **Receive a key update:** the sequence number of the received update is checked against the sequence number of the last executed key update for that specific object. If the received sequence number is larger, the packet is accepted, otherwise, it is rejected. An acknowledgement is generated and buffered in the acknowledgment buffer in order to be sent by the **Sender Module.** The update is buffered and only relayed to the application when it is time to be executed.
- **Receive a normal update:** the sequence number of the received update is checked against the greatest received sequence number for that specific object. If the received sequence number is larger, the packet is accepted, otherwise, it is rejected. The update is

buffered and only relayed to the application when it is time to be executed.

- **Receive an acknowledgment packet:** the timers of all the updates in the retransmission queue that have a sequence number smaller or equal to that of the acknowledgment are cancelled; these updates are also removed from the retransmission queue.

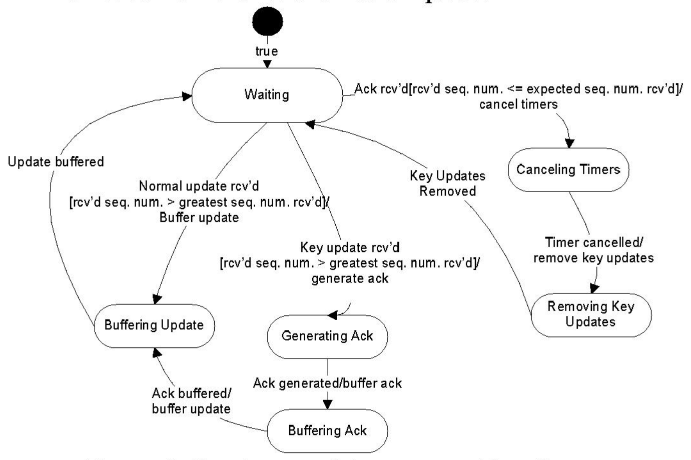

true
Waiting
Ack rcv'd[rcv'd seq. num. <= expected seq. num. rcv'd]/ cancel timers
Update buffered
Normal update rcv'd [rcv'd seq. num. > greatest seq. num. rcv'd]/ Buffer update
Key Updates Removed
Canceling Timers
Key update rcv'd [rcv'd seq. num. > greatest seq. num. rcv'd]/ generate ack
Timer cancelled/ remove key updates
Removing Key Updates
Buffering Update
Generating Ack
Ack buffered/ buffer update
Ack generated/buffer ack
Buffering Ack

Figure 4: Receiver module state machine diagram

IV. PERFORMANCE EVALUATION

A. Environment Setup

In order to analyze the performance of the protocol, we have developed an emulator program that generates pseudo-random haptic data at the haptic loop rate. We also generate graphic specific updates at the graphic loop rate to simulate the presence of other non-haptic objects in the environment. For the haptic and graphic data, the key updates are randomly chosen. On average, about 10% of the updates are tagged as key updates. The emulated environment contains three objects: A, B and C. A and B are graphic objects whereas C is a haptic object.

The simulation was performed on two computers with 100 Mbps Ethernet cards connected through a Local Area Network (LAN). The delay and jitter is simulated by buffering the received updates on the receiver side before being forwarded to the application, while packet loss was simulated by discarding updates according to a probability function.

B. Simulations and Results

The goal of the simulations is to prove the effectiveness of using the multiple buffering approach over the single buffering approach. In order to analyze our hypothesis, we ran the emulator program under different network conditions using both, the single buffering and multiple buffering schemes. We evaluated the results in terms of the throughput of key updates sent from a workstation and received at the other. At the receiver, duplicate key updates are counted only once.

Figure 6 shows the results of these simulations. Figure 6-a shows that in perfect network conditions (delay=0, jitter=1ms, and no packet loss), both approaches perform equally well. As soon as we start injecting delay, jitter and packet loss into the simulation, the resulting key updates throughput for both approaches starts to diverge especially for the haptic object C. Obsolete updates steals the available bandwidth from the fresher ones, especially if the updates are received but not acknowledged.

Another advantage of multiple buffering over single buffering is objects prioritization. We ran a simulation with only two graphic objects where object A is assumed more important than object B. We measure the total amount of updates received for each object with respect to time. Duplicate updates are not counted. Figure 5 shows the result under ideal network conditions. Object B's updates are delayed in favor of object A's updates. In the single buffering approach, the precedence in packet transmission can not be given to any of the objects as both object's updates are located in the same sending queue.

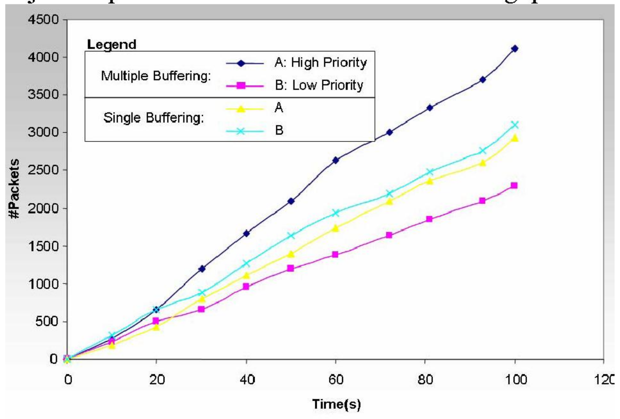
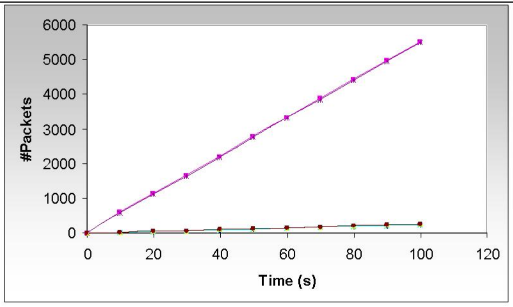


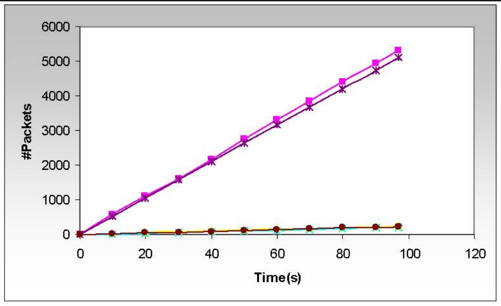
Figure 5: Sent packets versus time

V. CONCLUSION AND FUTURE WORK

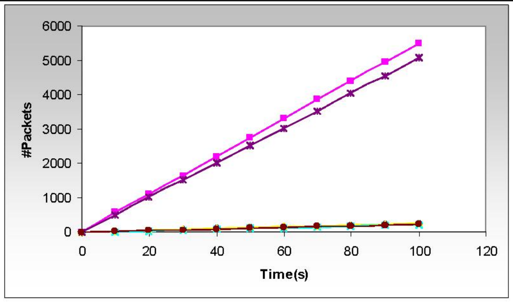
We proposed ALPHAN, an application layer protocol for haptic data communication over a non-dedicated network such as the Internet. The protocol is made to be highly customizable by passing application requirements to the protocol in HAML format. The protocol uses the concept of multiple buffering and packets prioritization to improve the C-HAVE application performance. Testing the ALPHAN protocol in a real haptic application scenario is our immediate future work.



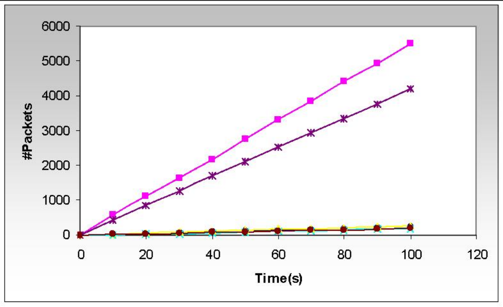
a) Delay 0, Jitter 1 and Loss 0%



b) Delay 80, Jitter 10 and Loss 2%



c Delay 80, Jitter 15 and Loss 2%



d) Delay 80, Jitter 20 and Loss 2%

Figure 6: Packet throughput for various delay and jitter configurations

As per future work, we plan to study the QoS parameters and their implications on the overall user performance (visually and haptically). This includes analyzing the effects of packet loss, latency, and jitter on the haptic interaction stability and transparency, as well as the overall user satisfaction. Additionally, the HAML part is not yet implemented in the current version, so we plan to build the HAML loader and test the performance of the protocol by varying QoS requirements applications from tele-surgery to entertainment and gaming applications.

REFERENCES

[1] M. Eid, M. Orozco and A. El Saddik, "A Guided Tour in Haptic Audio Visual Environment and Applications", J. of Advanced Media and Comm., v1 (3), pp: 265 – 297, 2007.

[2] X. Shen, J. Zhou, N.D. Georganas, "Evaluation Patterns of Tele-Haptics", Proc. CCECE2006, Ottawa, Canada, May 2006.

[3] J. P. Hespanha, M. McLaughlin, G. S. Sukhatme, M. Akbarian, R. Garg, and W. Zhu. Haptic collaboration over the internet. In Touch in Virtual Environments: Haptics and the Design of Interactive Systems. Prentice Hall, 2002.

[4] X. Shen, J. Zhou, A. El Saddik, N.D. Georganas, "Architecture and Evaluation of Tele-Haptic Environments", Proc. of DS-RT 2004, Budapest, Hungary, 2004.

[5] B. Chebbi, D. Lazaroff, F. Bogsany, P. X. Liu, N. Liya, M. Rossi, "Design and implementation of a collaborative virtual haptic surgical training system", IEEE Int. Conf. on Mechatronics and Automation. V. 1, Page(s): 315 – 320, 2005.

[6] C. Basdogan, C. Ho, M. A. Srinivasan, M. Slater, "An experimental study on the role of touch in shared virtual environments", ACM Trans. Comput.-Hum. Interact. 7, 4, 2000.

[7] S. Brave, A. Daley, "inTouch: A Medium for Haptic Interpersonal Communication", CHI '97, 363-364.

[8] C. Gunn, "Collaborative virtual sculpting with haptic feedback", Virtual Reality Journal, Vol 10, No 2, Oct 2006.

[9] R. Iglesias, A. Carrillo, S. Casado, T. Gutiérrez, J.I. Barbero, "Virtual Assembly Simulation in a Distributed Haptic Virtual Environment", in Int. con. on Adv. Design and Manufacture, 2006.

[10] F. R. El-Far, M. Eid, M. Orozco, A. El Saddik, "Haptic Application Meta-Language", DS-RT, Malaga, Spain, 2006.

[11] S. Dodeller, "Transport Layer Protocols for Haptic Virtual Environments", M.S. thesis, University of Ottawa, 2004.

[12] S. Shirmohammadi, N.D. Georganas, "An End-to-End Communication Architecture for Collaborative Virtual Environments", Computer Networks, Vol.35, No.2-3, 2001.

[13] S. Dodeller and N.D. Georganas, "Transport Layer Protocols for Telehaptics Update Messages", Proceedings of the 22nd Biennial Symposium on Communications, June 2004.

[14] M. Mauve, V. Hilt, C. Kuhmünch, W. Effelsberg, "RTP/I - Toward a Common Application-Level Protocol for Distributed Interactive Media", IEEE Transactions on Multimedia, 3(1):152–161, March 2001.

[15] P. Karn, C. Partridge, "Improving round-trip time estimates in reliable transport protocols", ACM Trans. Comput. Syst. 9, 4 Nov. 1991.

[16] P. Hinterseer, E. Steibach, S. Chaudhuri, "Model based data compression for 3D virtual haptic teleinteraction", ICCE '06, Jan. 2006.