uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Hussein Al Osman
AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.A.Sc. (Electrical Engineering)
GRADE / DEGREE

School of Information Technology and Engineering
FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Application Layer Protocol for Haptic Networking

TITRE DE LA THÈSE / TITLE OF THESIS

Prof. A. El Saddik
DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Prof. V. Groza

Prof. P. Liu

Gary W. Slater
Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Application Layer Protocol

For Haptic Networking

By

Hussein Al Osman

Ottawa-Carleton Institute for Electrical and Computer Engineering

School of Information Technology and Engineering

University of Ottawa

A thesis submitted to the
Faculty of Graduate and Postdoctoral Studies
in partial fulfillment of the requirements for the degree of
Master of Applied Science in
Electrical Engineering

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-48432-6
Our file Notre référence
ISBN: 978-0-494-48432-6

Contents

List of Figures

List of Tables

Acronyms

3D: Three Dimensional
ALPHAN: Application Layer Protocol for Haptic Networking
C-HAVE: Collaborative Haptic and Audio Visual Virtual Environment
CHVE: Collaborative Haptic Virtual Environment
CVE: Collaborative Virtual Environment
FIFO: First In First Out
GPP: Graphic Position Payload
GPS: Global Positioning System
GRP: Graphic Rotation Payload
GTMP: Graphic Transformation Matrix Payload
GTP: Graphic Translation Payload
HAML: Haptic Meta Language
HFP: Haptic Force Payload
HPP: Haptic Position Payload
HPVP: Haptic Force and Velocity Payload
HPVP: Haptic Position and Velocity Payload
HVP: Haptic Velocity Payload
ID: Identifier
IP: Internet Protocol
IPDV: Instantaneous Packet Delay Variation
LAN: Local Area Network
MRT: Maximum Allowable Response Time
MTU: Maximum Transmission Unit
NHE: Networked Haptic Environment
NTP: Network Time Protocol
OID: Object Identifier
PDV: Packet Delay Variation
PID: Participant Identifier
QoS: Quality of Service
RTO: Retransmission Timeout
RTP/I: Real-Time Protocol for Interactive applications
SCTP: Synchronous Collaboration Transport Protocol
SHE: Standalone Haptic Environment
SIRR: Synchronized Interaction Request Resolving
STL: Standard Template Library
TCP: Transport Control Protocol
UDP: User Datagram Protocol
VE: Virtual Environment

Acknowledgments

I would like to express my sincere appreciation to my thesis supervisor Dr. Abdulmotaleb El Saddik for his generous guidance, support and encouragement throughout my graduate studies. I would also like to convey my gratitude to Ph.D candidate Mohamad Eid for his continual support and friendly advices throughout my research.

I wish to thank my parents, my brothers Rida and Firas for their relentless moral support.

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 protocol is strategically placed at the application layer so that it can easily be customized according to the application requirements and needs. An extension to the HAptic Meta-Language (HAML) is proposed to define its networking parameters. The protocol introduces the concept of Multiple Buffering where priorities are attributed to different sending buffers in a collaborative haptic environment. The protocol also supports some concepts introduced in previous haptic communication protocols like local lag and key packets.

Chapter 1

1 Introduction

A multimedia system refers to the aggregation of multiple units of media in order to compose a harmonious system for presenting, processing and exchanging information. Such units of media can be categorized according to the human senses such as the sense of hearing (music, speech), sight (text, image, video, 3D graphics), smell (digital smell generator) and touch (haptics). Traditional multimedia systems concentrate mostly on the implication of the humans' sense of sight and hearing to convey information. Such multimedia systems saw users either witnessing the information display (television, cinema) or interacting and modifying it through predefined interfaces (computer games, software graphical user interfaces). Recently, computer scientists and engineers have been concentrating on immersing the users into the multimedia system by involving the other senses. Today, it is possible to modify information through touch and receive other forms of information through smell. Engaging the sense of touch has become recently a major field of research. This field is referred to as haptics.

It has been a constant interest for researchers to exchange multimedia information over long distances through communication networks. Perhaps the voice phone circuit switched networks are well known traditional examples. With the development of the packet switching technology, and evolvement of the Internet, new multimedia communication applications have surfaced. Examples of such applications include soft phones, video streaming, networked video games etc....

1.1 Collaborative Virtual Environments

The term Virtual Environments (VEs) refers to partially or totally computer generated simulations. Computer audio and video (graphics) simulations are traditionally used to create such environments. Video games are classic examples of Virtual Environments. On the other hand, Collaborative Virtual Environments (CVEs) applications are computer programs that allow multiple users to co-exist and

collaborate in a shared virtual environment to achieve a certain goal. In this case users can not only interact with the environment, but they can also interact with each other. Users are usually represented in the environment by their avatar. An avatar is a visualization of an entity (usually a user) that communicates its changing state with other avatars by continuously sending update messages.

A CVE usually requires a wide range of networking, data repository, and graphics capabilities. For instance, several data types should be supported such as real-time audio and video data, objects/scene description data, and control messaging data. Furthermore, CVEs require the consideration of the manner participants can interact with each other and the objects populating the environment and how a co-manipulated object should behave.

1.2 Haptics

The term haptics, originating from the Greek language, refers in general to the study of touch behavior. In recent years, the term has been associated mostly with technology that interfaces users to various computing devices through force feedback. Such haptic interfaces give users the ability to sense and manipulate virtual environments or remote real environments through touch [1].

1.2.1 Components of a Haptic System

A Haptic System is a multimedia system that incorporates the sense of touch as a possible media unit. Figure 1-1 represents a high level overview of a typical haptic system. The following describes the most important components of a haptic system:

- ***Haptic Device:*** an interface that allows a user to exchange force information with a virtual environment simulated by a computer. Haptic devices are responsible for transforming information received from a computer into forces through actuators. They also convert sensed forces applied by the users into digital information. Such information can be used for processing on a local computer or transferred over a network.

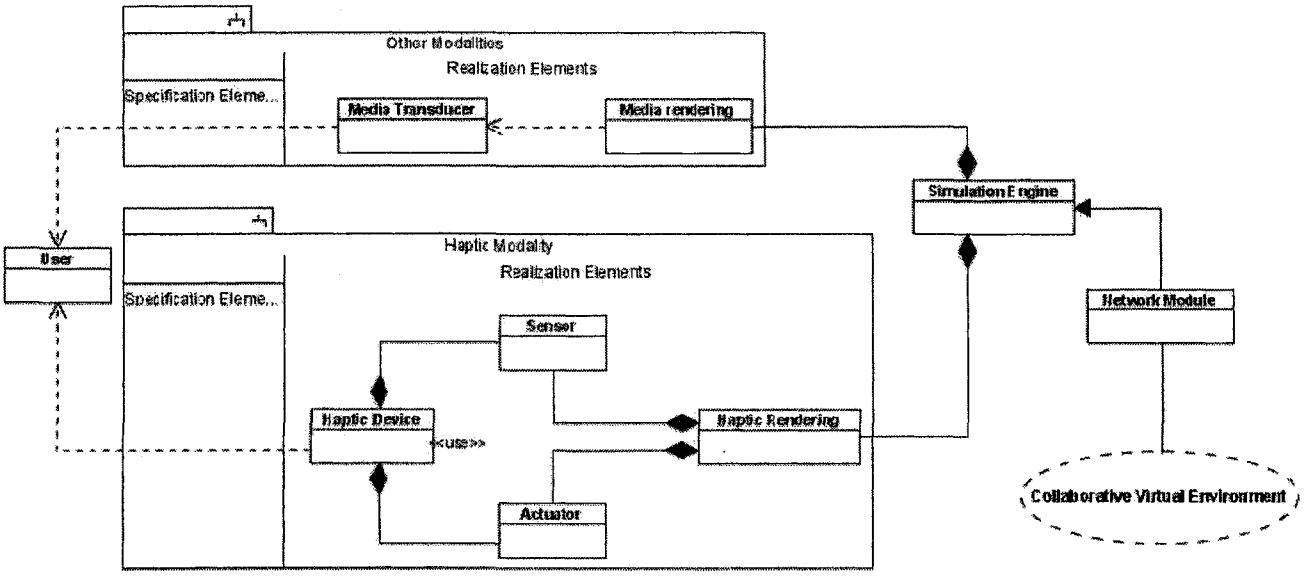


Figure 1-1 - Components of a haptic system [2]

- ***Haptic Rendering Module:*** composed of sensors and actuators responsible for constantly sensing and applying forces.
    - **Sensors:** responsible for sensing the forces exerted by the user typically on an object or another user in the environment.
    - **Actuators:** transform the digital representation of the feedback forces received into forces.
- ***Simulation Engine:*** simulates the virtual environment. Responsible for detecting collisions in the environment and computing the necessary force feedback.

1.2.2 Haptics and Collaborative Virtual Environments

The Sense of touch is an essential element of interpersonal communications, such as, but not limited to, a handshake, a hug or physical contact. It can be seen as a mean to express affection, intention or emotion [4].

Today, geographically dispersed users can talk, see, or interact with each other through text, voice and video. The Haptic technology has a potential to provide a new medium for user expression

through touch; consequently, this increases the perception of presence when it comes to distant communication.

Furthermore, haptic information can be used to promote learning over long distances. This is essential when it comes to learning a skill where the transmission of audio and visual information is not sufficient. For instance, transmitting haptic information through the network can be essential, for medical surgery applications where the force applied by the expert surgeon must be felt at the student's site [5]. This is called tele-mentoring where a trainer controls the actions of a trainee to carry out a task. Another form of distance learning or entertainment is the realm of remote manipulation where a user can haptically interact with a remote virtual scene. A typical application in such field is a chemistry teaching program. Students can haptically manipulate atoms to create molecules. This remote scene may be situated for example at a web-server.

A distributed Collaborative Haptic Virtual Environment (CHVE) or collaborative tele-haptic system is an example of a tele-haptic application, where multiple users, each with her/his own haptic device interact and manipulate virtual objects in a shared virtual environment. Generally, users share a common goal and collaborate towards its achievement. Usually two types of users are defined in such environment: passive and active users. Active users interact with the environment and with other users as opposed to passive users who simply witness the action. In some situations, some tasks may require closely-coupled collaboration between users, such as, lifting and moving an object together. These tasks are known as cooperative tasks [6][7], whereas other types of collaborative tasks involve either collaboration by taking turns or simultaneous collaboration.

The case of robotic tele-manipulation where a robot is controlled by a human operator through a haptic device may be seen as an example of a tele-haptic system. However, this research is out of the scope of this thesis.

1.2.3 Collaborative Virtual Environments Terminology

Many terms have been proposed to reflect the involvement of several users in an environment including collaborative haptics [8], shared haptic world [9], collaborative haptic virtual environments [10], cooperative haptics [7], distributed haptic virtual environments [12][13], and collaborative haptic audio visual environments (C-HAVE) [14]. Note that the aforementioned terms do not

stipulate that users participating in the collaboration are geographically dispersed and thus the application is distributed over a network.

Other terms have been proposed to specifically refer to networked haptic environments such as distributed haptics [12], tele-haptics [11][15], or networked haptics [16]. Here we describe the terms we are going to adopt to eliminate any ambiguities due to terminological conflicts.

First, we will be using the term *Standalone Haptic Environment (SHE)* to refer to a standalone environment that involves more than one participant interacting haptically with virtual/real objects that are located physically on the same machine. Second, the transmission of haptic information over a network – either dedicated or non-dedicated – will be referred to as *Networked Haptic Environment (NHE)*. The term will reflect the networking aspects of haptic data communication without considering collaborative scenarios. Finally, when both aspects are combined together, that is a shared environment that is distributed over a network, we will use the term *Collaborative Haptic-Audio Virtual Environment (C-HAVE)*.

1.2.4 Collaboration Paradigm

A virtual scene is composed of many objects which users can interact with. Objects can have different properties. They can be either static or dynamic. A static object remains stationary throughout the simulation. Usual examples of such objects are walls or immobile obstacle. The user can interact with static objects only by feeling their shapes and/or texture. On the other hand dynamic objects either move autonomously or are manipulated by the user. It is essential to associate a weight with dynamic objects in order to provide for a realistic manipulation. Objects in the environment can also be either solid or deformable. The shape of a solid object cannot change in response to force stimuli. Usually, we associate a high stiffness value with such objects. On the other hand, the shape of deformable objects is reworked according to the force subjected on the object and the simulated properties of the object. Deformable objects can also be elastic and inelastic. Elastic objects snap back to their initial shape when it is no longer subjected to forces. On the other hand, inelastic objects remain permanently deformed after the force stops.

In general, in C-HAVE applications, two types of interactions can be defined: dependent and independent interactions. In independent interactions, participating users exert forces on the

environment, but not on each other. An example of such application is a table tennis game, where players, using a virtual racket, exert forces on a virtual ball, but never directly on each other. In dependent interactions on the other hand, users produce forces on each other. Several types of dependant interactions exist: user-user interaction, user-object-user, user-object-object-user etc. Dependent interactions are more demanding since they require high stability in the haptic devices and the quality of the interaction can be easily degraded in the presence of delay and jitter.

1.3 C-HAVE Potential Applications

The potential of C-HAVE is significant for a wide spectrum of applications. We will mention only the most widely known:

- ***Distributed training:*** typical applications include those of assembly where several users collaborate to assemble the subcomponents of a system.
- ***Tele-mentoring:*** teaching students new skills can be facilitated through touch. For instance, for centuries, guiding students to write letters by moving their hand to produce the shape of a letter has been the natural way for teaching students how to write. Using the haptic technology, this can be done at long distances, where a teacher can guide students to write a letter through haptic devices.
- ***Surgical simulations:*** many collaborative surgical simulation applications have been developed. Through these applications, medical residents can collectively, using haptic collaboration technologies, rehearse critical surgeries.
- ***Entertainment and gaming:*** haptic enabled games have a huge potential in today's booming gaming and entertainment market. In the recent years, gaming companies have been tirelessly pursuing the objective of increasing the immersion of the player into the video games. Therefore, gamers will cease to be bystanders that control the game and witness the action from the outside; instead, gamers will become a part of the game. To achieve such immersion, an exchange of force between the player and the game environment is vital.
- ***Computer-mediated social interaction:*** haptic enabled social interaction is another promising C-HAVE field. Today, people can interact over distance through voice and video exchange.

But these modalities are somewhat limited in communicating the feelings of the communicating parties. Since human beings have learned to associate touch with different emotions, exchanging touch information can greatly improve the transparency of the interaction.

1.4 Thesis Contributions

The thesis introduces the Application Layer Protocol for Haptic Networking (ALPHAN). The main goal of this research is to develop a flexible haptic communication protocol that can support a wide range of distributed haptic applications. To achieve that goal, we have studied the existing haptic communication protocols. ALPHAN is developed to incorporate most of the features provided by these protocols in a customizable manner. To do so, we have:

- Studied C-HAVE applications and their varying requirements.
- Designed and implemented ALPHAN
- Designed and implemented a simulation application to perform a preliminary evaluation of ALPHAN
- Designed and implemented two C-HAVE applications and evaluated the performance of ALPHAN under different network conditions.

We have found that:

- Placing the haptic communication protocol on the application layer allows the protocol to support a wider range of C-HAVE applications since the latter can be customized according to the C-HAVE application needs.
- The use of multiple sending buffers at the sender side can improve the efficiency in which the haptic updates are transferred over the network.
- Under jittery network conditions, buffering the received updates at the receiver side improves the steadiness of the application.

1.5 *Resulting Publications*

1.5.1 Conference Publications

Al Osman, H.; Eid, M.; Iglesias, R.; El Saddik, A., "ALPHAN: Application Layer Protocol for HAptic Networking," Haptic, Audio and Visual Environments and Games, 2007. HAVE 2007. IEEE International Workshop on , vol., no., pp.96-101, 12-14 Oct. 2007

Al Osman, H.; Eid, M.; El Saddik, A., "Evaluating ALPHAN: A Communication Protocol for Haptic Interaction," Haptic interfaces for virtual environment and teleoperator systems, 2008. haptics 2008. symposium on , vol., no., pp.361-366, 13-14 March 2008

Al Osman, H.; Eid, M.; El Saddik, A., "Evaluating Evaluating ALPHAN with Multi-User Collaboration," DS-RT, 12-th IEEE International Symposium on Distributed Simulation and Real Time Applications, 2008

1.5.2 Journal Publication

Eid, M.; Al Osman, H; El Saddik, A., "Evaluating ALPHAN: An Object-based Communication Protocol for HAVE Applications," ACM Transactions on Multimedia Computing, Communications, and Applications (submitted)

1.6 *Thesis Outline*

The remainder of the thesis is organized as follows:

- Chapter 2 presents background knowledge on C-HAVE applications. It presents a generic C-HAVE architecture and explains the different components involved in such applications. It also discusses the different network architectures proposed for C-HAVE applications. Chapter 2 also presents the existing haptic communication protocols.
- Chapter 3 presents the Application Layer Protocol for Haptic Networking protocol. It also discusses the design and implementation of a code library that supports ALPHAN.
- In chapter 4, we evaluate ALPHAN using a specifically designed simulation application.

- Chapter 5 and 6 are devoted for the design implementation of C-HAVE applications. The performance of ALPHAN is evaluated for both applications.
- Finally, chapter 7 contains the concluding remarks.

Chapter 2

2 Background

2.1 *Generic Architecture of a C-HAVE System*

Figure 2-1 shows a generic architecture of the collaborative haptic world. The architecture presented here does not reflect the physical distribution of the nodes and storage components, but rather the logical distribution. For instance, the virtual objects database can be distributed over the participant's host machines (objects that are owned by a participant can have their information stored locally at that participant node).

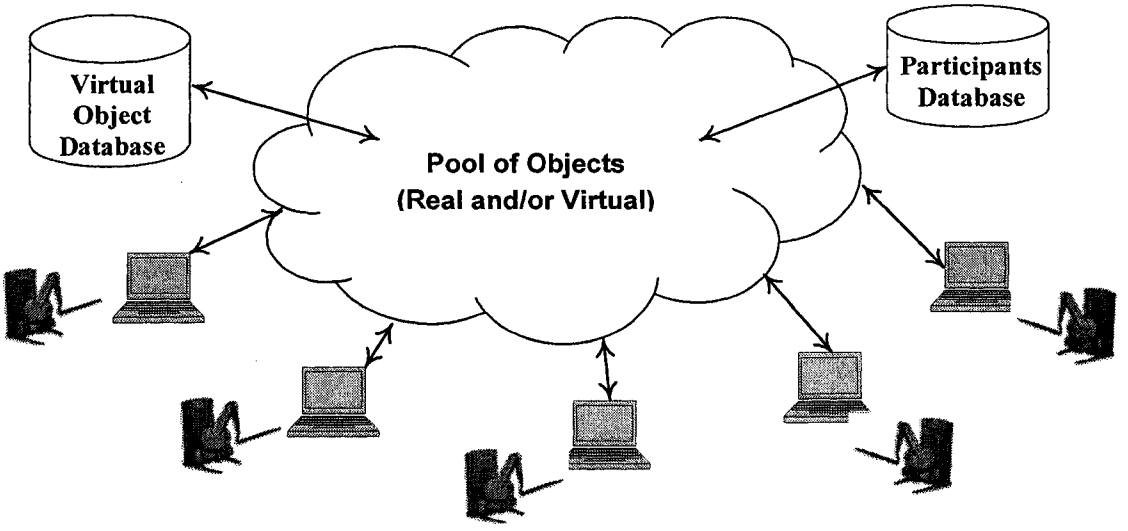


Figure 2-1 - Collaborative Haptic-Audio Visual Environment (C-HAVE) Architecture

The two databases are continuously updated throughout the simulation as new participants and/or objects join or leave the environment at run-time. The participants' database stores the identification information about the nodes (such as logical identifiers, IP addresses, and any other relevant information) that are currently interacting with the environment. Furthermore, the

participant's database can contain the quality of service (QoS) parameters associated with the links between all the connected nodes. This is useful for routing purposes when many users are participating in the simulation. The virtual objects database contains all the information about the objects that populate the environment. Such information include the object identifier, owner identifier, static properties of the objects (such as shape, color, size, etc.), and dynamic properties such as position, orientation, and velocity.

Collaborative haptic environments over a network pose new requirements and challenges at both the application level and the communication (networking) level. The haptic interaction requires simultaneous and interactive input and output by the haptic device with extremely high update rate (up to 1 kHz). At the application level, consistency assurance, access control, transparency, and stability are experiencing extensive research. At the networking level, quality of service parameters such as the network latency, jitter, packet loss, scalability, and compression are key aspects that have been investigated and researched. This chapter fills the gap of understanding the characteristics of haptic interaction over a network.

2.2 *Network Architectures for C-HAVE*

Remote haptic collaboration between people is achieved by distributing all the interactions with the shared environment to every participant. The most common network architectures in collaborative multimedia applications are the client-server and the peer-to-peer architectures, although many other hybrid architectures have been proposed [17]. The client-server architecture is more secure and thus common in the commercial world whereas the peer-to-peer architecture is preferable in the research context since performance is of greater importance. Within closely coupled collaboration, such as haptic interaction, there are two major concerns: consistency and responsiveness (delay and/or jitter).

Server-based systems provide high consistency, but they lack scalability and high responsiveness. Therefore, most collaborative haptic applications use the peer-to-peer distribution model. However, when the peer to peer approach is selected, additional consistency control measures are required to coordinate how the shared environment is seen and manipulated. Currently, the goal in the design of collaborative haptic applications is to find the balance between responsiveness and

consistency. Additional important factors that are considered in the design process are scalability, access control, and security, among others. Recently, many researchers started using hybrid architectures where both paradigms are combined in order to control the tradeoffs between consistency and responsiveness.

2.2.1 Client-Server Architecture

The client-server architecture, as shown in Figure 2-2, is a straightforward architecture that comprises a centralized server and one or more clients. The server runs the simulation of the shared haptic environment and communicates state changes to every participant. The participants have local representations of the simulation for rendering purposes; they do not perform any simulation activities locally. All participant interactions (graphic and/or haptic) are sent directly to the server. The server computes the new graphic and haptic state of the simulation and instructs the clients to update their local representations of the environment by sending the updated transformation matrices of the dynamic objects. As for the haptic device, each participant sends its device's position and velocity information and receives back the interaction forces that must be displayed by the haptic device. The collision detection and force computations algorithms run at the server side.

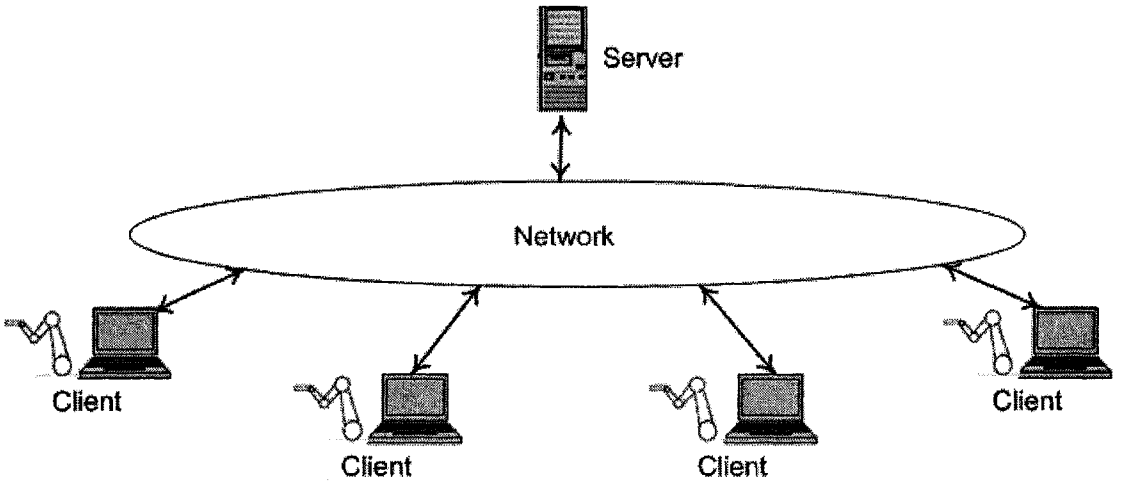


Figure 2-2 - The client-server architecture for tele-collaborative haptic applications

The client-server architecture's main advantage is the ease with which consistency among participants can be guaranteed since all the application data is stored and managed by the server [7][17]. Managing the simulation state across all participants is simple and thus synchronization

between what each client can see and feel is not an issue. Additionally, many researchers argue that this architecture is robust against the effects of latency and jitter [17]. In addition to the synchronization benefits, the single point of contact property makes the start-up of applications more simplified, particularly for late comers. The late comer does not need to contact every node in the network to inform them about its presence or to retrieve the state of the shared environment.

On the other hand, this approach has many disadvantages: First, the client-server architecture has poor responsiveness. That is all the data must go through the server, thus inflicting extra transmission delay. The added latency is sometimes intolerable for haptic applications. Second, the architecture is not scalable since adding too many clients makes the server a bottleneck. Third, the collaborative system has a single point of failure; if the server fails, the whole system is interrupted. The single point of failure problem can be solved with hardware redundancy although that solution can prove to be costly. Finally, there is a substantial waste of resources on the clients' side. These clients are performing very few processing since the state of the environment is always computed on the server side before being transmitted to the clients. On the other hand, this approach requires a high duty server, or a cluster of servers, able to swiftly perform complex computations in order to service all clients. For all the above mentioned limitations, most shared haptic applications use the peer-to-peer architecture. Several researchers have adopted the client-server architecture to build tele-collaborative haptic applications, [7][18][19][20][21] are good examples.

2.2.2 Peer-to-peer Architecture

A peer-to-peer architecture on the other hand has each participant (or peer) directly applying the user's inputs and computing the state of the shared environment locally. Each peer communicates its own updates with the environment to other peers through direct connections as shown in figure 2-3. The main benefit of this architecture is that it avoids any additional latency of interaction between the client and server and hence has proved to be a popular architecture for a number of collaborative haptic systems [10][22]. The following presents an example of a typical peer-to-peer interaction flow:

1. Each participant runs the simulation for the shared environment locally and interacts with the environment and other users.

2. Upon interaction, the client updates the virtual environment and calculates the reaction forces for itself and each client involved in the interaction. The client uses the position information received from the haptic device in the simulation.
3. The client sends to all the peer clients the updated state of the virtual environment and the reaction forces.
4. Other clients receive the interaction forces and the new state of the environment and update their local states accordingly. Notice that for realistic haptic interaction, the update rate should be as high as 1 kHz.

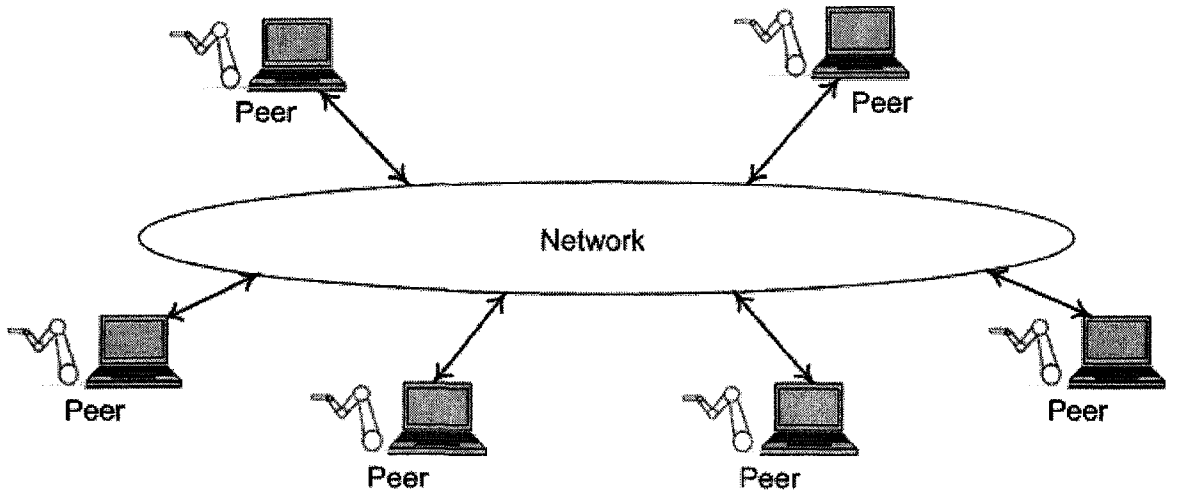


Figure 2-3 - The peer-to-peer architecture

The peer-to-peer approach is with no doubt the most used for haptic applications since it overcomes most of the client-server disadvantages mentioned above; mainly responsiveness and scalability. On the other hand, the consistency assurance with this scheme becomes a non-trivial task since the state of the environment is computed by each client individually and then delivered to peer clients. Some of the consistency assurance mechanisms are presented later in this chapter. Furthermore, the scheme is computationally expensive since each client computes the simulation and interaction forces for the entire environment locally. Finally, the procedure for late comers who join the application is not as intuitive as in the case of client-server architecture. In this case, the late comer needs to contact all other peers to establish the current state of the environment and make sure that

consistency is achieved at the time of joining the simulation and afterwards. The peer-to-peer architecture has been used by many researchers such as [7][10][23][24[25][26].

2.2.3 Hybrid Architecture

Hybrid approaches have been also proposed. For instance, one approach requires the use of a Lock Manger server (see figure 2-4). Each participant that wishes to interact with an object in the environment has to request the lock to the object from the Lock Manager [27]. If the object is not locked, the permission is granted and only then the interaction can take place. The operations that are performed on locked objects are transmitted to all peers. In this approach, the consistency is guaranteed since only one user is interacting with an object at a time. On the other hand, this approach is limited to applications where only one user has to interact with an object at a time.

The roaming-server is another hybrid architecture proposed by Marsh et al. In this architecture, a server is strategically placed on each local area network, where every server acts as a simulation engine (see figure 2-5). One of these servers is nominated to manage the simulation by ensuring that only one simulation engine is active at a time [27]. Clients connecting to the system will chose their preferred simulation engine, typically the one residing on their local network. Assume that client A expresses the need to collaborate with an object by sending a request to the environment in order to activate its preferred simulation engine. If all other simulation engines are dormant, the permission for manipulating the object is granted to the requesting client. Now assume client B wants to interact with an object in the environment, client B sends a request to the environment so that its preferred server be activated. But at this point, the request is rejected since another server is already active. Client B will have to then interact with the already active server even though it is not its preferred server. Given that both clients are interacting with the same server, consistency is thus automatically ensured. If a short period elapses with none of the users interacting with any object in the environment, all the servers are re-placed in the dormant state [27].

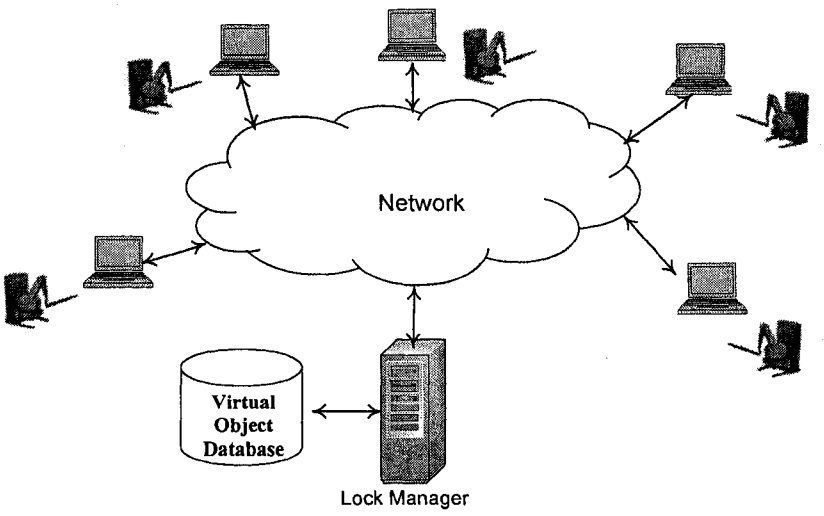


Figure 2-4 – Lock manager based hybrid architecture [27]

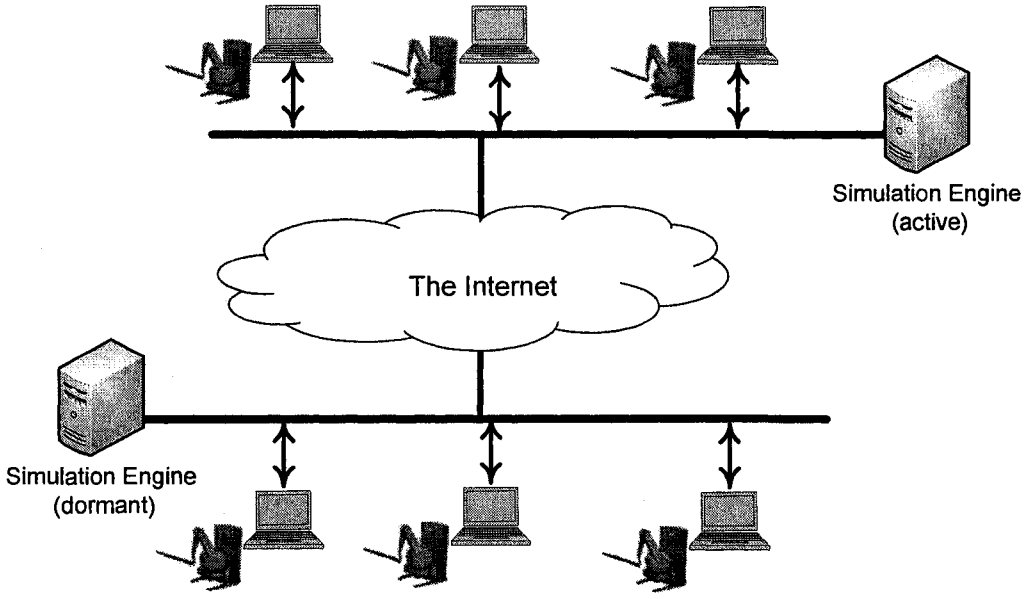


Figure 2-5 – Roaming server hybrid architecture [27]

[11] present the Synchronized Interaction Request Resolving (SIRR) architecture. In SIRR, when multiple users intend to interact with a common object in the environment, a central server (interaction manager) is temporarily used to organize such interaction. The central server orders the requests and stores them temporarily before applying them to the related object. The new state of the object is then reflected back to all participants (see figure 2-6).

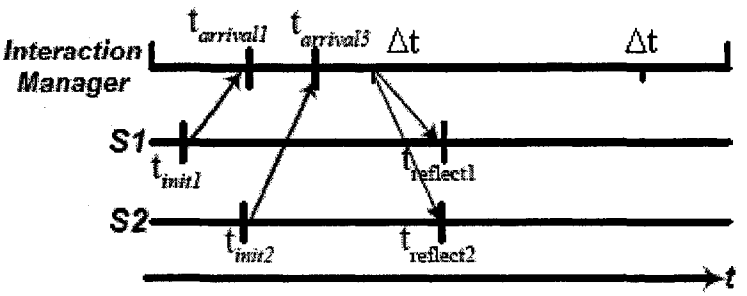


Figure 2-6 - Synchronized Interaction Request Resolving (SIRR) architecture [11]

Hespanha and his colleagues suggested a distribution architecture that partitions the haptic environment into nodes (a computer and an operator) [28]. Two databases are maintained: node database and objects database. The node database contains information about the nodes currently participating in the networking (logical ID, IP addresses of all connected hosts, and the latency and bandwidth available between them). The objects database contains information about all objects populating the haptic world. The states of the participants are synchronized periodically through broadcasts. Also, to solve the scalability issue, the authors proposed partitioning the nodes into local groups sharing a high bandwidth and low latency connection.

2.3 *Existing Communication Protocols*

In order to efficiently build a haptic communication system, issues such as network delay, jitter and packet loss must be taken into consideration. Such issues constitute a serious challenge when the communication of force information occurs over non-dedicated networks such as the internet. For instance, delays such as 100 ms, which are common over the internet, can cause the two users, on both ends of the communication, to undergo a different collaboration experience due to the inconsistencies caused by the network lag.

Mitigating the effects of delay and jitter can be performed on two levels: on the haptic communication level and on the human-computer interaction level. On the first level, techniques such as local lag, timewrap, dead reckoning and other jitter smoothing schemes can be used. On the second level, the use of visual cues to alert the user of the presence of network lag, and thus allowing him/her intuitively to adapt to the situation, have proven to be effective.

Early works on collaborative virtual environments have been largely concentrated on supporting large number of participants. But there has been a shift in this trend as it became apparent that the virtual universe could not successfully support more than ten to twenty users and that few applications require the participation of more than this number of collaborators. In haptic collaborative applications in particular, even fewer number of users are required for most applications. Nonetheless, it is always possible to have numerous passive participants. These participants can monitor the environment but are unable to actively modify its state. A multicasting scheme must be employed in order to support multiple active and passive participants during collaborations. With the scarcity of multicasting services on the network layer, haptic communication systems must support multicasting on the application layer.

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. Nonetheless, some attempts to produce such protocols were made at the DISCOVER lab at the University of Ottawa. Other protocols that tackle the issues of networked collaboration have been also developed. First, we will explore the generic transport layer protocols (namely TCP and UDP) and then we will investigate other haptic specific protocols.

2.3.1 TCP

The Transport Control Protocol (TCP) is a connection oriented, transport layer protocol. It is a reliable end to end delivery protocol and thus provides the following services: error control, sequence control, loss control and duplication control. This implies that packets sent over TCP are relayed to the receiver in the same order they were sent. While this property might be advantageous for many applications, it has a negative impact on haptic applications. Imagine the case where a haptic update is lost during transmission, all the updates that follow will be buffered on the receiver side and will only be relayed to the application when the lost update is successfully retransmitted. In this case, fresh updates are being needlessly buffered on the receiver side, while network resources are being exhausted in order to retransmit an obsolete one [29].

2.3.2 UDP

The User Datagram Protocol (UDP) is a connectionless, unreliable best effort transport layer protocol. It is a simple protocol that only provides a basic form of error detection using checksums whereas it makes no attempt to recover lost packets. While this protocol does not suffer from any of the drawbacks of TCP, it does not fully suit the reliability requirements of collaborative applications [29]. Nonetheless, UDP remains the most popular transport layer protocol for real time applications.

2.3.3 Light TCP

Light TCP is somewhat inspired from TCP, as the name suggests, while avoiding its drawbacks. To support the concept of message obsolescence, the sender side queue accepts update messages from the application and processes them as follows:

- A key update is placed at the end of the queue and marked as a key message in order to prevent it from being erased [29].
- A normal update can replace older normal updates for the same shared object. However, replacing an older update message with the new one is possible only if there are no other updates between the older message and the end of the queue [29].
- Unacknowledged update messages are placed again in the queue if no newer updates from the same object have been produced by the application.

Messages are taken from the queue and sent as one IP packet. The size of the packet is determined by a congestion control algorithm where the more congested the network, the smaller the number of updates transmitted as one packet [29].

At the receiver side, a received update is immediately forwarded to the application if its sequence number is bigger than the last received update's sequence number, otherwise it is dropped. There is no buffering.

The following is a list of the commonalities of Light TCP with TCP:

- Messages are queued on the sender side. However, unlike TCP, in Light TCP, the queue is not a simple FIFO queue;
- An acknowledgment mechanism is employed to ensure reliability;

- Sequence numbers are used for packet ordering;
- The sliding window and congestion control algorithm are both employed.

2.3.4 SCTP

The Synchronous Collaboration Transport Protocol (SCTP) is similar to the UDP protocol since most of the messages are sent unreliably. On the other hand, key messages are sent reliably. It also differs from UDP since the sequence numbers are used for packet ordering whereas in UDP, sequence numbers are not used at all. For each reliably sent message (key message), a timer is set. If a timeout occurs before the acknowledgement of the message in question, it is resent, always as a key update [30]. It has been proven that for collaborative applications, SCTP, which is an acknowledgment-based protocol, performs better than protocols based on negative acknowledgments, in which the receiver has to detect a loss and ask for a retransmission [30].

2.3.5 Smoothed SCTP

The smoothed SCTP protocol adds a jitter smoothing mechanism to the SCTP protocol. On the sender side, the Smoothed SCTP and SCTP protocols are completely similar. 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 δt milliseconds ago and in case an update is found, it is retrieved from the bucket and forwarded to the application [31]. This means that all updates, including the ones generated locally, are processed with a δt delay. This approach is very similar to the local lag algorithm that will be discussed later in this chapter.

2.3.6 RTP/I

Unlike the previous protocols presented in this section, the Real-Time Protocol for Interactive applications (RTP/I) is an application layer protocol. This is a general protocol designed for network distributed interactive applications (not specifically for haptic applications) Nonetheless, haptic applications fall under this category. In this protocol, the messages exchanged between participants are categorized as: event, state and request-for-event packets [32]. An event packet, as the name suggests, carries an event or a fraction of an event. This is the most common type of packets

interchanged between participants. A state packet carries the entire state of a subcomponent (object) in the environment. Such packets are crucial in order to be able to implement a late comer support mechanism where participants that are late in joining in the collaboration activities could be quickly informed of the state of the environment. The request-for-event packets are used by participants to indicate that the transmission of some subcomponent's state is required by the sender [32].

The problem with this protocol is its generality. In its quest to support the widest spectrum of interactive collaborative applications, the protocol adds a lot of communication overhead. The extra overhead cannot be afforded in haptic applications that have 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 kilo Bytes of header information are transmitted every second. This header contains many fields that are unnecessary for haptic applications.

2.4 *Performance Issues in C-HAVE*

In information technologies, there is an unavoidable time delay when it comes to data transmission. Moreover, the delay between two endpoints is not fixed, every time a packet is sent, it can be received earlier or later than the last received one. Most routers today do not guarantee reliable and in-order delivery at the network layer and thus packets can get lost or arrive out of order. Another factor to be taken into account is the available bandwidth and throughput to deliver information.

2.4.1 Network Latency

When information is transmitted over a non-dedicated network such as the Internet, packets experience delays caused by buffering, processing, transmission, and propagation. This delay varies depending on the physical distance between the communicating parties and in accordance to the load of the network.

2.4.2 Network Jitter

In any network environment, there is no constant delay for packet transmission. Every time a packet is sent, it can be received earlier or later than the last received packet. The variation of the network delay

is known as jitter. The increase in jitter has been shown to be one of the greatest factors in degrading the quality of the haptic experience [33]. Jitter can also cause the mass inconsistency illusion, where a user feels that the mass of a manipulated object is varying with time [34]. The Local lag algorithm can be used to smooth the effects of jitter [32].

2.4.3 Packet Loss

Packet loss occurs due to deficiencies in network resources caused by network congestion (an overwhelming increase in the amount of traffic circulating through the network). Additionally, the queuing strategies that are implemented in intermediate routing nodes determine the conditions and the rates at which packets are discarded. It has been demonstrated that the retransmission mechanisms that are implemented in TCP are not suitable for C-HAVE systems [35].

When it comes to C-HAVE applications, the loss of packets can cause small discontinuities in the simulation that are felt by the user. However, the loss of large bursts of position or force information packets can easily impede the collaboration and distract users. Therefore, most haptic transport protocols prioritize transmitted packets and treat them differently based on their corresponding importance. For instance, packets that need reliability (usually named key packets messages) such as a packet reporting a collision between two shared objects are usually delivered using an acknowledgment-based reliability mechanism. On the other hand, packets that do not require reliability (such as transient position information) are sent using the best-effort delivery approach.

2.4.4 Scalability

The presence of more than two users in a C-HAVE may be important for specific scenarios. For instance, in gaming environments the participation of multiple users is desirable. Conversely, applications aimed for training, learning or remote manipulation do not aim at supporting a large number of users [14]. Likewise, the idea of having C-HAVE applications supporting multiple users with only few users interacting at a time is feasible; however, the ability to support numerous simultaneous user interactions is still a farfetched goal. Moreover, cooperative tasks in most cases do not require more than five users interacting concurrently.

2.4.5 Consistency Assurance

Implementing a consistency assurance mechanism is an essential task when it comes to developing any peer to peer distributed virtual environment application. Such a mechanism is responsible for ensuring that every node participating in the simulation keeps a consistent view of the environment. When it comes to consistency, distributed applications can be organized in two categories: discrete and continuous applications. Discrete applications change their state only in response to user generated operations. Examples of such applications include distributed white boards and shared drawing tools [36]. Continuous applications, on the other hand, not only change their state with response to user generated operations, but also to the passage of time. Examples of such applications include multiplayer games and distributed virtual environments in general; C-HAVE applications fall under this category.

The most popular consistency assurance mechanism used for such applications is Dead Reckoning (DR) (especially in gaming). In Dead Reckoning, it is assumed that the behavior of each object over time is predefined. Thus, each object's state is updated locally. Operations are also performed locally on each object and after each local update, the resulting state is calculated with respect to the one that could have been obtained from the DR algorithm; if the difference between these two states is larger than a preset threshold, an update is broadcasted to all participants. The main difficulty with this algorithm is its potential to produce short-term inconsistencies during the transmission of an update destined to correct the error in the predicted state. Mauve has proposed an alternative approach to solve the problem of inconsistency using the local lag and the timewrap algorithms [37].

2.4.5.1 Local Lag

To make use of the local lag algorithm, it is assumed that each operation is qualified with two timestamps: the time when the operation is issued and the time when the operation should be executed at all nodes concurrently. The difference between these two timestamps is called the lag value. Figure 2-7 depicts this concept. As it can be clearly deduced, the longer the local lag value, the less responsive the system will seem yet the more consistent the local copy of the simulation on each node will be.

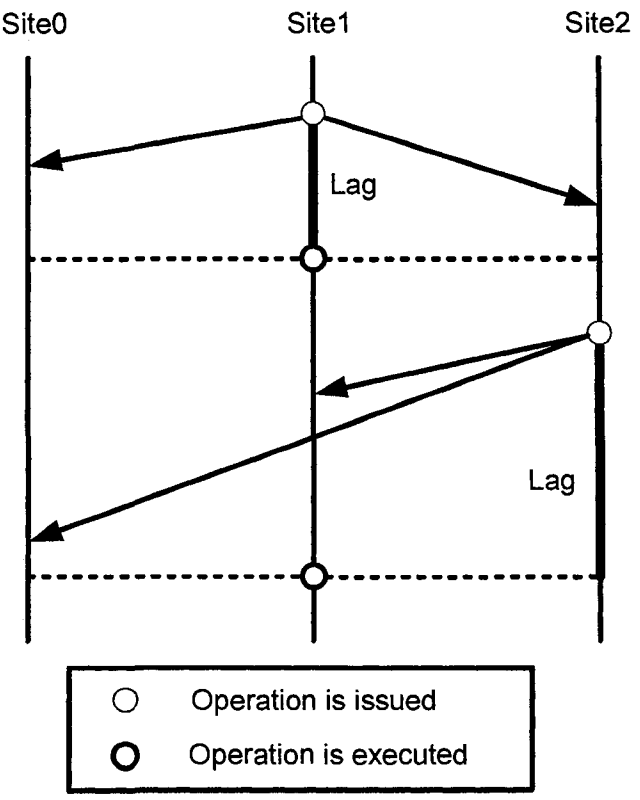


Figure 2-7 - Local lag concept [37]

Choosing an appropriate lag value is important in order to balance the responsiveness of the system and its consistency. Mauve proposed the use of a constant local lag throughout the execution of the application [37]. The local value is extracted from two values: the maximum of the average network delays among all participants and the maximum allowable response time of the application. The latter value is obtained from psychological tests and is more or less subjective [37]. Chen proposed an adaptive approach for choosing the local lag value [38]. That is, using statistical sampling, the network latency of the next short duration is estimated and the local lag value is corrected accordingly.

2.4.5.2 Timewrap

Local lag by itself cannot solve the entire consistency problem. There will always be times where packet transmission takes longer than the local lag time due to jitter. In this case, these packets are processed locally before being processed by the other participants and thus resulting in inconsistencies. Mauve proposed the timewrap algorithm in order to deal with such short term inconsistencies and therefore ensure complete consistency for all the participants [37]. The timewrap

algorithm consists of periodically correcting any short-term inconsistencies by rolling back each loop to a base state, where all the operations are received or locally produced, since the base state occurrence is re-executed. This can be summarized in the following steps:

1. Receive an operation or produce an operation locally.
2. Save the received or produced operation in the operation list in ascending order according to its t*, where t* indicates the time the operation should be executed.
3. Save the state of the system, S_t, in the state list according to t, where t is the time when S_t was the state of the system
4. Every T milliseconds, where T is the preset period of time, perform the following:
   - Retrieve the base state S_i, where S_i was the state of the system T milliseconds ago
   - Perform all the operations stored in the operations list between the time of S_i and the current time by applying them to S_i according to their chronological order indicated by t*. Each time an operation is applied, the new state produced replaces the old one in the state list.

2.5 *Motivation*

Although many haptic communication protocols exist, none of them is customizable enough to capture the varying requirements of various distributed haptic applications. All of these protocols lie on the transport layer of what has come to be known as the Internet Protocol Stack. By definition, the transport layer is responsible for responding to service requests from the application layer and issuing service requests to the network layer. As it is the case with all layered architectures, the interfaces that separate these layers are well defined. This leaves little room for a haptic application to impose its requirements on the transport layer protocol through the predefined interface.

In order to realize a fully customizable haptic communication protocol, it is essential to revise the placement of such protocols. Placing a haptic communication protocol on the application layer can allow it to gain this much needed agility. It would position the protocol in direct contact with the application; this presents the opportunity to redefine the interface between the application and the communication protocol. The Application Layer Protocol for Haptic Networking (ALPHAN) is

designed to achieve this flexibility. ALPHAN benefits from the concepts introduced in previous protocols, while adding new concepts to offer the haptic application developers the opportunity to adapt their own instance of the protocol according to their particular requirements.

Chapter 3

3 ALPHAN

3.1 *Application Layer Protocol for Haptic Networking*

The Application Layer Protocol for Haptic Networking (ALPHAN), unlike many of its predecessors, operates on the application layer. This design decision is favored by the very nature of haptic data and applications. When it comes to haptic applications, only the application knows its requirements. Therefore, placing the protocol on the application layer allows it to be better adapted to the application needs. For instance, in the case where a local lag algorithm is employed, the lag value can only be efficiently determined if we measure the maximum allowable response time; the latter value widely varies from one application to another and can only be measured by performing subjective psychological tests on a set of potential users. A protocol on the application layer can also have a better knowledge of the objects in the environment. Such knowledge is utilized in ALPHAN in order to allow for a more efficient interaction.

To further improve the customizability ALPHAN, the parameters that are used to configure the protocol according to a specific haptic application requirements can be listed and defined in a Haptic Meta Language (HAML) [39] format (such as the maximum allowable response time, shared objects priorities, security, trustworthiness and availability etc.). 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 and can describe other aspects of the application such as the graphic and haptic models and rendering, haptic data, haptic interfaces, among others [39].

Some of the features supported by ALPHAN were introduced in previous protocols and have been proven to be essential for haptic communications. Such features include the reliability for key updates and jitter smoothing mechanisms. Other features introduced by ALPHAN include multiple

buffering and the ability to prioritize objects according to their importance in the environment. Perhaps the biggest advantage of ALPHAN is its ability to be customized to fit the application's needs.

A packet is a block of data formatted to be transmitted over a packet switched network. In packet switching networks, packets can be transmitted through different routes between source and destination and can be received out of order. Every unit of information transmitted over the network to update the state of an object in the environment is called an update. Note that a haptic device is also considered an environment object. For example the position of a haptic proxy at one point of time is considered an update.

3.2 *ALPHAN Packet Structure*

All data transferred over ALPHAN is divided into updates that are framed by a common header field. Commonly every packet holds one update. Figure 3-1 depicts the ALPAHN packet structure. Notice that the header is compact, composed of 12 bytes in order to avoid producing unnecessary overhead. Yet the generality of the header is not compromised by its small size, it contains all the common information required for haptic and graphic data transmission.

All the header fields are carried in the network byte order, that is, most significant byte first. This byte order is commonly known as big-endian. Padding bytes are filled with zero. The timestamps of ALPHAN are 32 bit fields. They represent the milliseconds passed since 0h UTC on 1 January 1900. Timestamp values wrap after 2^{32} milliseconds. Participants of an ALPHAN simulation should synchronize their clocks, so that they can correctly identify the current timestamp cycle in case the timestamp field is being used by the application.

3.2.1 Field Summary

3.2.1.1 Version

Two bits are reserved to indicate the version of the protocol. Currently the version field is set to zero. The receiver must check the version of the update received in order to deduce how extract the information embedded in the header. Different versions can have a different header structure.

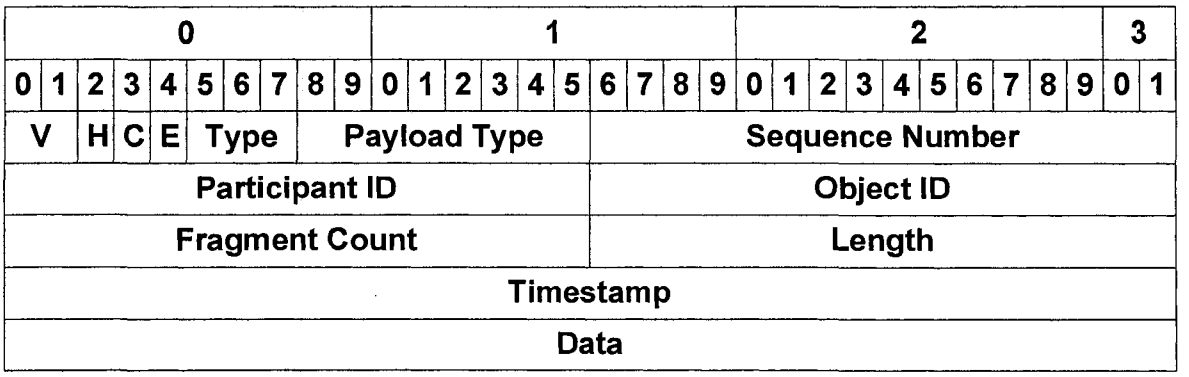


Figure 3-1 – ALPHAN packet structure

3.2.1.2 Haptic Flag

This on bit flag indicates whether the update contains haptic or graphic data. If the bit is set, the update contains haptic information. A haptic update contains information that either represents the forces being produced by a haptic device or any other types of data that contribute into calculating these forces on the receiver side; position and velocity information are commonly transmitted for that purpose. Graphic updates on the other hand contain any type of data that would be used to render objects graphically on the receiver side.

3.2.1.3 Collision Flag

This one bit flag indicates whether this update informs of a collision between two or more objects in the environment. If the bit is set, the update contains collision data. Collision events are very important for a realistic haptic interaction. To reflect their importance, collision updates can be set up to be treated with higher priority than other updates. On the sender side, such updates can be given precedence when it comes to network transmission. Collision updates are sent reliably.

3.2.1.4 End Flag

Graphic updates can be very large. If an update is larger than the Maximum Transmission Unit (MTU), it can be fragmented at the network layer by routers. This can be disadvantageous since one lost fragment can cause the whole packet to be discarded. ALPHAN can be customized to perform fragmentation on the application layer for large updates. If a portion of an update is lost, it can be retransmitted instead of retransmitting the whole update. This only applies when the update is being

sent reliably. An unreliably sent update in that case is never retransmitted but rather discarded. By default, ALPHAN assumes the MTU to be 1500 bytes. This value can be modified according to the network properties. Fragmentation would be rarely used for haptic updates since they are characterized by their small size and high frequency.

The E flag has another purpose. When several updates are sent in the payload of a packet, the E flag determines whether each update has its individual timestamp or not.

3.2.1.5 Type Field

Three bits are reserved to specify the type of the update. This means that up to eight update types can be defined. Only four types are predefined (see table 3-1).

Table 3-1 – Predefined update types

Update Type	Description
Key Update	Holds data representing key events that must be conveyed to the other side reliably. Such events are defined by the application that runs on top of ALPHAN. Note that collision updates are sent reliably regardless of the update type.
Normal Update	Makes up the majority of the updates that are sent over the network. They are sent unreliably over the network.
Incremental Update	Useful when bandwidth is limited, these updates hold incremental information with respect to the last received key update. These updates are especially useful when key updates are being often sent; this would produce small size incremental updates.
Control Update	Holds no useful information in its payload. The control update is used for the delay and jitter estimation between two hosts on the network.

Up to four other types of updates can be defined by application developers in the case they contrive other types that better suit their application requirements.

3.2.1.6 Payload Type

This field identifies the format of the ALPHAN payload and determines its interpretation by the application. Table 3-2 depicts the haptic payloads supported by default.

Table 3-2 – Predefined hatpic payload types

Payload Type	Description
Haptic Position Payload (HPP)	Contains a vector that represents the updated position of the haptic proxy in the environment.
Haptic Force Payload (HFP)	Contains a vector that represents the force information that should be applied on the receiving side haptic device
Haptic Velocity Payload (HVP)	Contains a vector that represents the velocity of the haptic proxy
Haptic Position and Velocity Payload (HPVP)	Contains a position vector followed by a velocity vector
Haptic Force and Velocity Payload (HPVP)	Contains a force vector followed by a velocity vector

Other types can be defined by the application to support various payload formats. These are highly dependent on the application's requirements. Moreover, the default payload types support haptic devices with a single point of interaction. Other payload types can be defined for haptic device with multiple points of interaction like the cyber glove.

Table 3-3 depicts the graphic payload types supported by default. Other types can be defined by the application to support various payload formats.

Table 3-3 - Predefined graphic payload types

Payload Type	Description
Graphic Position Payload (GPP)	Contains a vector that represents the updated position of an object in the environment
Graphic Rotation Payload (GRP)	Contains a vector that represents the rotation an object must undergo along all three axes. The angles are expressed in radiant.
Graphic Translation Payload (GTP)	Contains a vector that represents the translation an object must undergo along all three axes.
Graphic Transformation Matrix Payload (GTMP)	Contains the full transformation matrix that must be applied to the current transformation matrix of an object. The transformation matrix contains information about both the orientation and position of the object.

3.2.1.7 Sequence Number

There are independent sequence numbers for each object in the environment. For each transmitted full update, the appropriate sequence number is incremented by one. The acknowledgement of a key update carries the same sequence number of that update.

3.2.1.8 Participant Identifier

The Participant Identifier (PID) uniquely identifies a participant. This participant is the original sender of the packet. ALPHAN does not define a specific mechanism to choose the PIDs. A PID is persistent for the lifetime of the simulation. In particular, the restart of an application does not change the PID of the participant. If an application requires multiple ALPHAN sessions, the PID remains the same for the participant across all of these sessions.

3.2.1.9 Object Identifier

Every update is sent to uniquely modify the sate of one object in the environment. The Object Identifier (OID) uniquely identifies an object in the environment. Sixteen bits are reserved for this

field, which means that an environment can theoretically hold up to 65536 objects. ALPHAN does not define a specific mechanism to choose the OIDs. An OID is persistent for the lifetime of the simulation.

3.2.1.10 Fragment Count/ Update Count

This field has two purposes. When fragmentation is enabled, this field can be used to indicate the fragment count in case an update is fragmented. The fragment count for each update starts with zero and is incremented by one for each packet sent for this update by a participant. A receiver knows that it has received all parts of a fragmented update when it has received a packet with the fragment count zero, a packet with the E bit set and all fragments in between. Sixteen bits are reserved for this field which means that a single update cannot consist of more than $(2^{16}-1)$ fragments. In this case the update is said to be fragmented over multiple packets.

This field can also be used to indicate the update count. The update count represents the number of updates included in one packet. This is common in haptics where updates are fairly small. In this case, several updates can be fitted into one packet payload that can have a common header. Having a common header for several updates can drastically reduce overhead. This means that all updates fitted onto one packet must have the following properties:

- Have a common update type
- Have a common payload type
- Are Generated by the same participant
- Update the state of the same object
- Have the same length

If the E flag is set, all the updates will rely on a common timestamp in the header. This is useful in two cases:

- Timestamps are not being used since local lag is disabled and the application is not using any other schemes that require timing information.

- We assume that the time separating two updates is one millisecond since most haptic devices produce updates at that rate. In this case, the first update's timestamp is the timestamp indicated in the header; the timestamp of the second update is (timestamp +1) and so on...

If a timestamp is required for each update, than a timestamp field would precede each update in the payload of the packet. Figure 3-2 depicts this packet structure. The E flag is set to zero in this case.

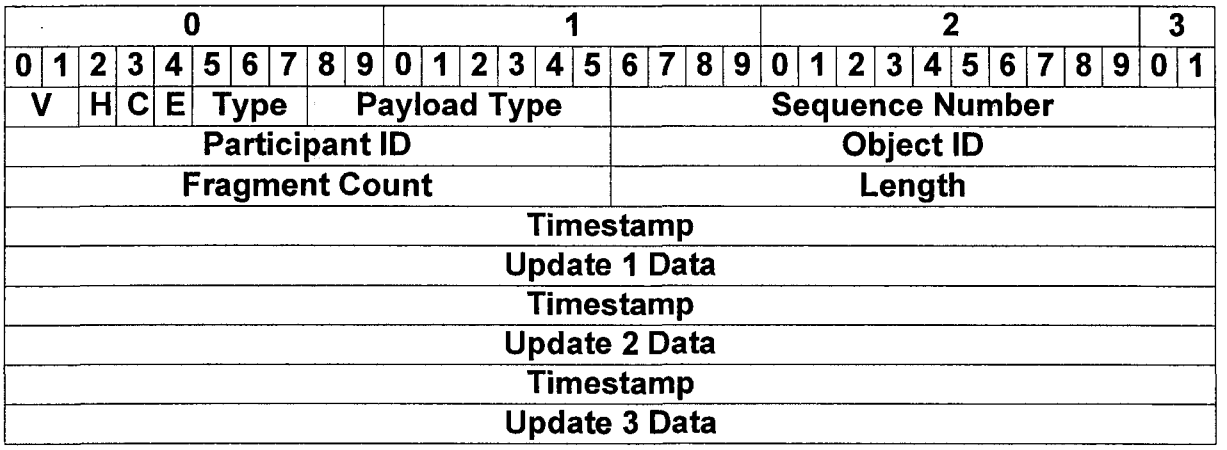


Figure 3-2 – ALPHAN packet containing multiple updates

3.2.1.11 Length Field

The length field specifies the length of the update in bytes, excluding the twelve bytes of ALPHAN header. If the length is not a multiple of four, then a minimal amount of padding is appended to the packet so that the total length of the packet including the padding octets is a multiple of four. In case multiple updates are transmitted within on packet, than the length field indicates the length of an individual update. As previously mentioned, the lengths of all the updates within one packet are assumed to be the same.

3.2.1.12 Timestamp Field

The timestamp value is expressed as milliseconds that have passed since 0h UTC on 1 January 1900 modulo 2^{32}. The timestamp indicates the instant of time where the update must be consumed at the receiver side. The timestamp is based on a physical clock. The physical clocks of all participants must be synchronized. The mechanism to synchronize the clocks is out of ALPHAN's scope. Typically clock synchronization is performed using the Network Time Protocol (NTP) [40] or a Global Positioning System (GPS). The field might also be used to establish global ordering among all

updates generated by all the participants. Such global ordering is very useful when it comes to implementing consistency assurance mechanisms like timewrap.

3.3 *Acknowledgment Policy*

Every key update must be acknowledged. The acknowledgement holds the same sequence number as the update it is acknowledging. An acknowledgment can potentially be used to acknowledge several key updates; its sequence number must be equal to biggest sequence number of the updates being acknowledged. These updates must belong to the same object. Figure 3-3 shows an acknowledgement packet structure.

3.4 *Multiple Buffering Scheme*

The proposed 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. Such reliability schemes cause undesirable overhead that cannot be afforded in real time communication. On the other hand, our proposed protocol supports the notion of key updates by implementing an application layer reliability mechanism that is only applied to such updates while other updates remain unaffected.

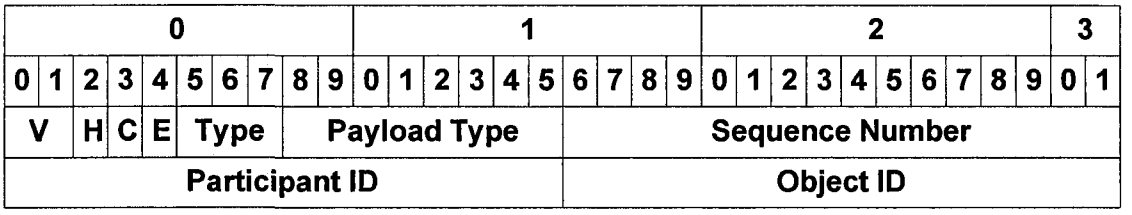


Figure 3-3 – ALPHAN acknowledgement packet

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 by default attributed a sending buffer. Each sending buffer consists of two queue data structures, a sending queue and a retransmission queue (see figure 3-5). The sending queue holds all the updates that are forwarded by the application to be sent. The retransmission queue is somewhat similar to the TCP retransmission queue; it holds the key updates from the time they are sent, until they are acknowledged. This allows us to re-send a key

update in case a retransmission timeout occurs before its acknowledgement is received. Allocating a buffer for each object permits the decoupling of update transmission for different objects in the environment, especially for objects that are independent of each other.

Attributing a buffer for each object has two advantages. Firstly, this scheme allows us to easily attribute a priority for each buffer. The priority of the buffer corresponds to the priority 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 will add flexibility to the protocol. In the single buffer approach, all the key updates that are transmitted are queued in the same retransmission queue, exactly like TCP. At some point, if the sending data rate is high enough, the queue will be full of obsolete key updates in the retransmission queue since the application has probably produced newer ones. For instance, for graphics data, if we are continuously sending the complete state of an object (transformation matrix), the newest key update is the most important one, since it holds the newest state of the object and the older key updates are useless. On the other hand, this does not apply for most haptic data since discarding an important key update can result in producing higher than intended forces on the receiver side. Figure 3-4 shows the path of a haptic object (haptic device proxy) in a two dimensional world. If key update 2 is lost (or discarded), than the receiver application will mistakenly produce unwanted forces that will take the haptic device from the state inscribed in key update 1 to the state inscribed in key update 3 directly. In the multi-buffering scheme, we can set the size of each retransmission queue of each object. For graphic objects, the size of the retransmission queue can be set to one, while for haptic objects, the size can be varied according to the application requirements. Whenever a new key update is to be added to the retransmission queue, the oldest key update is dropped in case the queue is full.

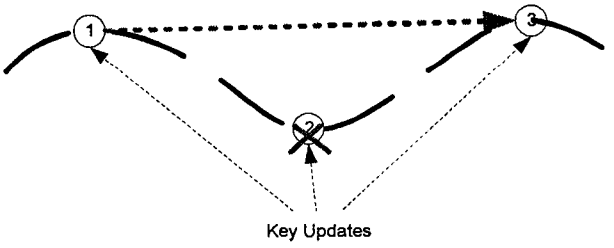


Figure 3-4 – Example of a haptic device path

3.5 *Network Conditions Monitoring*

3.5.1 End-to-End delay Estimation

Control data is used to measure the delay between two endpoints. This is done by transmitting a test packet from host A to host B on the network. Any "padding" appended to the header of the packet to make the test packet a given size should be filled with randomized bits. This is necessary to avoid a situation in which the measured delay is lower than it would otherwise be due to compression techniques along the path. Host B returns the packet to host A while preserving the sequence. Host A can thus measure the roundtrip time. The round trip time is divided by two to estimate the end-to-end delay between the two nodes. Host A should forward the received packet back to host B in order for the latter to calculate the end-to-end delay.

In the case where the clocks of both sender and receiver are synchronized, a different approach can be used to calculate the delay between two hosts. This approach can be summarized as follows:

a) Host A sends a test packet time stamped with the current time (T_1) to host B. T_1 is preserved for later use.

b) If the packet arrives within a reasonable period of time, host B records the current time (T_2), the time at which the packet was received.

c) Host B timestamps the test packet with the current time (T_2) and returns it to host A.

d) If the packet arrives within a reasonable period of time, host A records the current time (T_3), the time at which the packet was received.

e) The end-to-end delay at host A and B are calculated with using equations 3-1 and 3-2.

$$delayHostA = T_3 - T_2$$ **Equation 3-1**

$$delayHostB = T_2 - T_1$$ **Equation 3-2**

This approach can produce more accurate results if the clock synchronization is accurate enough. Typically for haptic applications, the clock synchronization accuracy has to be within 1 millisecond.

3.5.2 Packet Delay Variation Estimation

It is essential to estimate the Packet Delay Variation (PDV) or jitter in order to apply jitter smoothing algorithms like local lag. The jitter is estimated as follows:

1. Estimate the end-to-end delay (D_1) between host A and host B.
2. Perform another estimation between the same hosts (D_2)
3. The Instantaneous Packet Delay Variation (IPDV) is calculated using equation 3-3.
4. The PDV is recalculated every time a new sample of the IPVD is obtained using equation 3-4 where α is a heuristic value between 0 and 1. The bigger α, the slower the PDV value will change over time. $PDV_{(i+1)}$ is the new PDV value, where PDV_i is the old one.

$$IPDV = |D_1 - D_2|$$ **Equation 3-3 [41]**

$$PDV_{(i+1)} = PDV_i \times \alpha + (1 - \alpha) \times IPDV_{i+1}$$ **Equation 3-4**

3.6 *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 of the update, it is resent again. Waiting for the correct amount of time before resending the packet can be crucial for efficient transmission. If the sender waits too long before resending, time is wasted. Conversely, if the sender waits for a short time than it risks of unnecessarily resending packets that might eventually make it to the receiver and thus wasting valuable network resources. In order to adeptly estimate the retransmission timeout value, it is necessary to periodically, every T ms, measure the round trip delay between the sender and the receiver. Equations 3-5 and 3-6 are used to calculate the waiting time where S_i is the last measure of the end-to-end delay calculated through equations 3-1 or 3-3, $D_{(i+1)}$ is the estimate of the end-to-end delay time for the next T milliseconds and α is a decimal number between 0 and 1 that dictates how fast does D adapt to change [42]. RTO is the retransmission timeout value and β refers to the Key Update Multiplier. It is a value higher than 1 that can be specified by the application. This value is set to 2 by default. R is the number of retransmissions performed so far for this key update.

Equation 3-5[42]

$$RTO = [\beta \times (R + 1)] \times D$$ **Equation 3-6**

3.7 *Signal Prediction*

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 JND, an update is sent to the other participants. [43] proposed a linear predictive model defined simply by equation 3-7 where $\{v_i, v_{i-1}, v_{i-2}, \ldots\}$ are the most current values produced by the model and $\{t_i, t_{i-1}, t_{i-2}, \ldots\}$ are the corresponding timestamps. $\{v_{new}, v_{new-1}, v_{new-2}, \ldots\}$ are the last received updates and $\{t_{new}, t_{new-1}, t_{new-2}, \ldots\}$ are the corresponding timestamps.

$$v_i = \begin{cases} v_{new} & new-value-arrived \\ v_{i-1} + \dfrac{v_{new-1} - v_{new-2}}{t_{new-1} - t_{new-2}}(t_i - t_{i-1}) & else \end{cases}$$ **Equation 3-7 [43]**

The reader might notice that this approach is analogous to the dead reckoning scheme described in chapter 2, only that this model is better adapted for haptics and it is not used for consistency purposes, but rather for compression.

3.8 *Local Lag*

The local lag scheme is the proposed approach for consistency maintenance in this protocol. On the other hand, in order to maintain the generality of the protocol, this feature is only optional. In case other consistency schemes are chosen, like dead reckoning for example, they could be easily implemented on top of the protocol. Nonetheless the inclusion of local lag can prove convenient since not only it facilitates consistency maintenance, but it also helps in smoothing out the jitter effects. 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 average delay in the network and on the maximum allowable response time. The latter value can be read from the Haptic Applications Meta

Language (HAML) description of the haptic application and thus the protocol can be dynamically customized to fit the need of the application.

If $[\mu \times (D + PDV)] < MRT$

$$timestamp = currentTime + \mu \times (D + PDV) \qquad \textbf{Equation 3-8}$$

Else

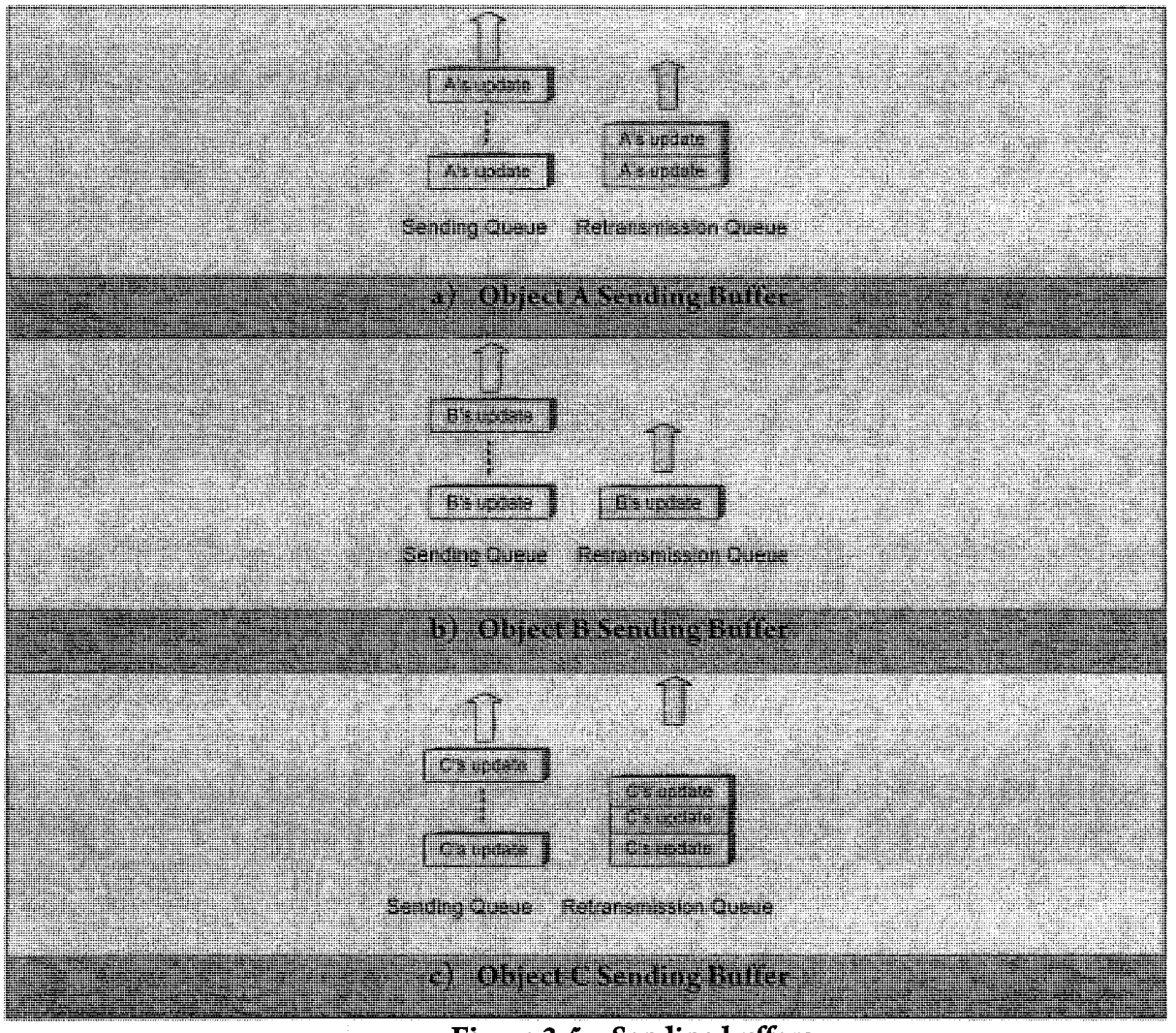


Figure 3-5 – Sending buffers

When local lag is enabled, an update's timestamp is calculated using equation 3-8 where:

- μ is the lag multiplier that controls the local lag algorithm. Choosing a large value for μ would ensure that an update is consumed almost every millisecond in a timely fashion on the receiver

side. But it also results in reducing the responsiveness of the application felt on the receiver side. A trade-off must be performed to ensure the best results for specific network conditions. By default, μ is set to 1.

- PDV is the Packet Delay Variation (jitter)
- D is the end-to-end delay between source and destination
- MRT is the Maximum Allowable Response Time

When local lag is disabled, the timestamp is simply equal to the current time.

Note that in order to use the local lag algorithm, the clocks of all the participating computers must be synchronized since time stamping must be used.

3.9 Objects Priorities

The priority given to an object is relative to its importance in the environment. For instance, in a surgical haptic application, the most important object is the organ being operated on and thus it has the highest priority. Other objects in the environment like subordinate organs or unused surgical tools are of secondary importance and thus should be given a lower priority. There are 5 different priorities defined in ALPHAN, with 5 being the highest and 1 the lowest priority. As we have previously mentioned, objects with higher priority will get precedence during the transmission of updates. An object with priority 5 is allocated five times more bandwidth by ALPHAN than an object with priority of 1. This means that objects with higher priority will get serviced more often and thus keep their sender queues relatively empty compared to other objects with a lower priority.

3.10 Customization through HAML

The Haptic Meta Language is an XML-based language designed specifically to provide a technology-agnostic description of haptic models and environments [39]. HAML expresses a haptic environment in terms of its ergonomic requirements and specifications for haptic hardware and software interactions [39]. We propose an extension to HAML where ALPHAN specific parameters are recorded. Figure 3-6 shows a sample of such file. Table 3-4 provides a description of some of the ALPHAN parameters declared in the sample HAML file. Note that although by default ALPHAN maps each object to a sending buffer, this property can be overridden. In the example presented in

table 3-4, ALPHAN is customized to have one buffer holding the updates of two objects. If no buffer settings are specified in the HAML file, the default setting will be automatically chosen. Also, by default, the retransmission queue size of each object can hold an unlimited number of key updates.

```
<?xml version="1.0"?>
<ApplicationDS
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com applicationds.xsd">
        <ApplicationName>Cataract Surgery</ApplicationName>
        <FieldOfUsage> Medicine </FieldOfUsage>
        <ApplicationType>
                <HostBased>
                <StandAlone> no </StandAlone>
                </HostBased>
        </ApplicationType>
        <ALPHAN>
                <network dataPort="8000" controlPort="80001">
                        <keyUpdate>
                                <timerMultiplier>1.5</timerMultiplier>
                                <maxRetransmission>4</maxRetransmission>
                        </keyUpdate>
                        <localLag enabled="true">
                                <maxLag>60</maxLag>
                                <lagMultiplier>0.6</lagMultiplier>
                        </localLag>
                        <fragmentation enabled="true">
                                <MTU>2134< /MTU >
                        </fragmentation>
                        <maxUpdateCount>5</maxUpdateCount >
                </network>

                <participants>
                        <participant uid="0" ip="137.156.212.64">
                        <participant uid="1" ip="137.134.112.112">
                        <participant uid="2" ip="137.153.124.21">
```

```
            </participants>

            <sharedObjects>
                <object uid="342" name="Cube1" priority="1">
                <object uid="321" name="Cube2" priority="4">
                <object uid="212" name="sphere" priority="2">
            </sharedObjects>

            <buffers>
                <buffer retransmissionQueueSize="5">
                    <mappedObjectUid>342</mappedObjectUid>
                </buffer>
                <buffer retransmissionQueueSize="12">
                <!--This buffer is customized to hold the updates of two objects-->
                    <mappedObjectUid>321</mappedObjectUid>
                    <mappedObjectUid>212</mappedObjectUid>
                </buffer>
            </buffers>
        </ALPHAN>
<!--Other HAML parameters-->
...
...
</ApplicationDS>
```

Figure 3-6 – HAML example

Table 3-4 – ALPHAN parameters description

Parameter Name	Description
Data Port	The network port used to send haptic and graphic data
Control Port	The network port used to send control data. Control data is used to measure the delay between two endpoints.

Key Update Timer Multiplier	Key updates are sent reliably. A timer is started for every sent updates. If a timeout occurs, the update is retransmitted. The Retransmission Timeout (RTO) is calculated using equation 3-6 where the key update multiplier is mapped to β.
Maximum Retransmissions	The maximum number of retransmissions before a key update is dropped.
Maximum Lag	The maximum time (in milliseconds) a packet can be buffered on the receiver side before being forwarded to the application when the local lag scheme is enabled. This Maximum Lag is equal to the maximum allowable response time of an application.
Lag Multiplier	This value is used to calculate an update's timestamp before delivery (see equation 3-8).
Maximum Transmission Unit (MTU)	The maximum size of an update. An update bigger than the MTU is fragmented.
Maximum Update Count	The maximum number of updates that ALPHAN can fit in a packet. The sum of the size of all the updates must be smaller than the MTU.
Retransmission Queue Size	The maximum number of updates that can be stored in the retransmission queue of the corresponding sending buffer. Whenever a new key update is to be added to the retransmission queue, the oldest key update is dropped in case the queue is full.

3.11 ALPHAN Library Implementation

3.11.1 Code Library Architecture

3.11.1.1 Overview

Based on the protocol's specifications presented earlier in this chapter, we have developed a code library that implements the core functionality of the protocol. The library is developed in C++. The C++ programming language 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 some low level constructs and thus allowing developers to gain better control over the application's performance. Note that haptic applications can be expected to send packets at a rate of 1 KHz of haptic data alone, which means that optimizing the code library for speed is of paramount importance. The library is developed for the Windows platform and makes use of the Win32 API. The Standard Template Library (STL) is also heavily used for its proficient implementation of data structures such as the queue, priority queue and map.

Figure 3-7 shows a high level view of the library architecture and table 3-5 gives a description of the library's modules.

Table 3-5 – ALPHAN library modules description

Module Name	**Description**
Application Module	The application module runs on top of ALPHAN. This module holds both the Graphic and Haptic loops. A thread is associated with each of these loops.
Protocol Manager Module	Runs as a separate thread. It 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 running on top of ALPHAN.
Sender Module	Runs as a separate thread. It holds the sending buffers. The updates that are placed in the buffers are sent as soon as possible according to their priority.
Receiver Module	Runs as a separate thread. It 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. It is them retrieved by the Sender Module in order to be sent to its appropriate destination. All the updates received by the Receiver Module thread are buffered in the Update Forwarder buffer.

Update Forwarder Module	Runs as a separate thread. This module is responsible for relaying packets to the Protocol Manager at the appropriate time. Note that the appropriate time is specified in the timestamp of the update. Such mechanism is necessary for the implementation of the local lag algorithm for consistency purposes or jitter smoothing. When an update is Forwarded to this module, it is stored in the Forwarder priority queue. The priority of each update is inversely proportional to the timestamp specified in the update. This results in ordering the updates in the queue according to their timestamps.
Network Sensing Module	Runs as a separate thread. This module is responsible for periodically examining the connection with the other participants in order to calculate the end-to-end delay and packet delay variation (jitter) between the local host and all other participants. It is also responsible for detecting any disconnections. The end-to-end delay value is necessary for both implementing an application layer reliability mechanism for key updates and implementing an application layer multicasting scheme to support multiple participants.
NTP Module	This module is responsible for synchronizing the local clock with the clocks of the other participants.

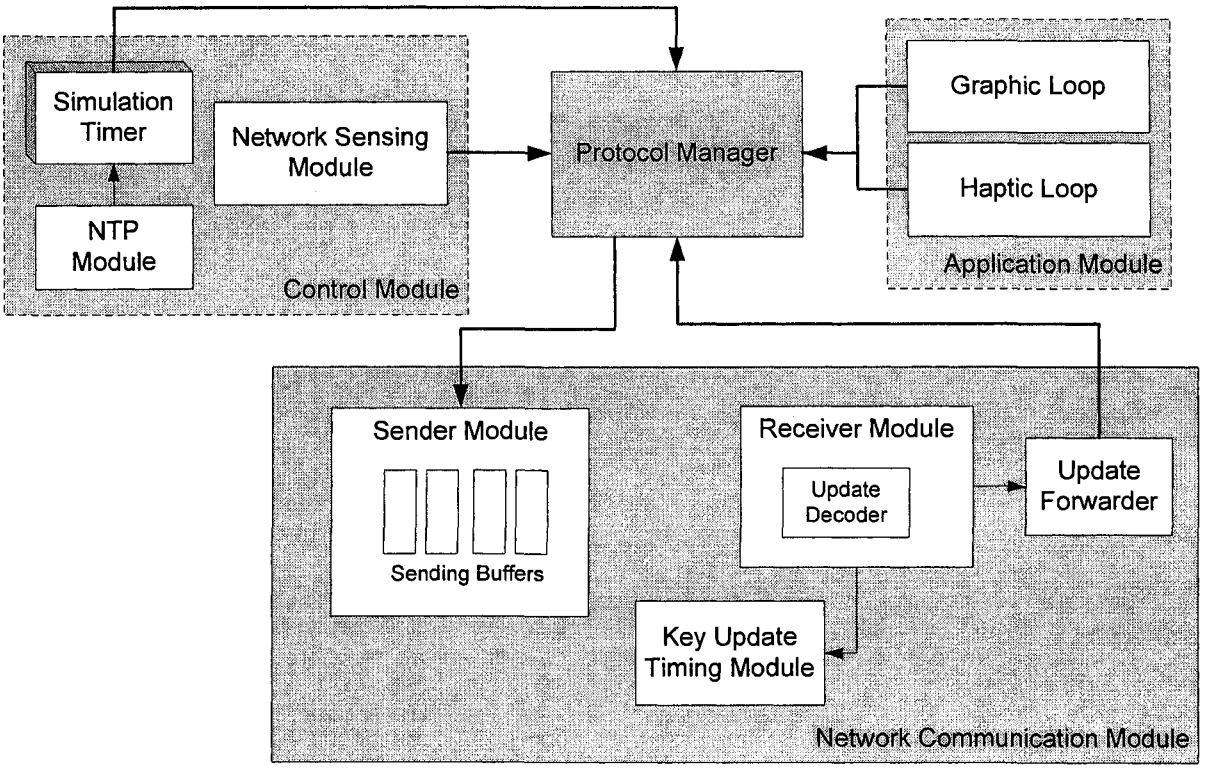


Figure 3-7 – ALPHAN library architecture

3.11.1.2 Sender

The ***Sender Module***'s behavior is represented in the state machine in figure 3-8. The Sender remains in the idle mode until one of the events described in table 3-6 occurs.

Table 3-6 – Sender state machine description

Event	**Response**
Receive a key update from buffer	A timer is started and the update is placed in the retransmission queue before being sent. Placing the key update in the retransmission queue is essential in order to be able to resend it in case it is not acknowledged. Note that at the beginning of the simulation, timers are created and placed in a timer pool. A timer is kept in a passive mode until it is invoked. It is essential to create timers

	ahead of time since each one of them runs in a separate thread. Creating and destroying many threads while the application is running can have an enormously negative impact on performance.
Receive a "non-key" update from buffer	The update is sent. The sequence number of that particular object is incremented.
Receive a timeout signal	Check to what update the timeout belongs; retransmit the corresponding update and place it back in the retransmission queue. Increment the retransmission counter associated with the update.

3.11.1.3 Receiver

The ***Receiver Module***'s behavior is represented in the state machine in figure 3-9. The Receiver remains in the waiting mode until one of the events described in table 3-7 occurs.

Table 3-7 - Receiver state machine description

Event	**Response**
Receive a key update	The sequence number of the received key update is checked against the greatest sequence number received for key updates 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 received update is buffered and only relayed to the application when it is time to be executed.
Receive a "non-key" 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 dropped. The received

	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 stopped; these updates are also removed from the retransmission queue.

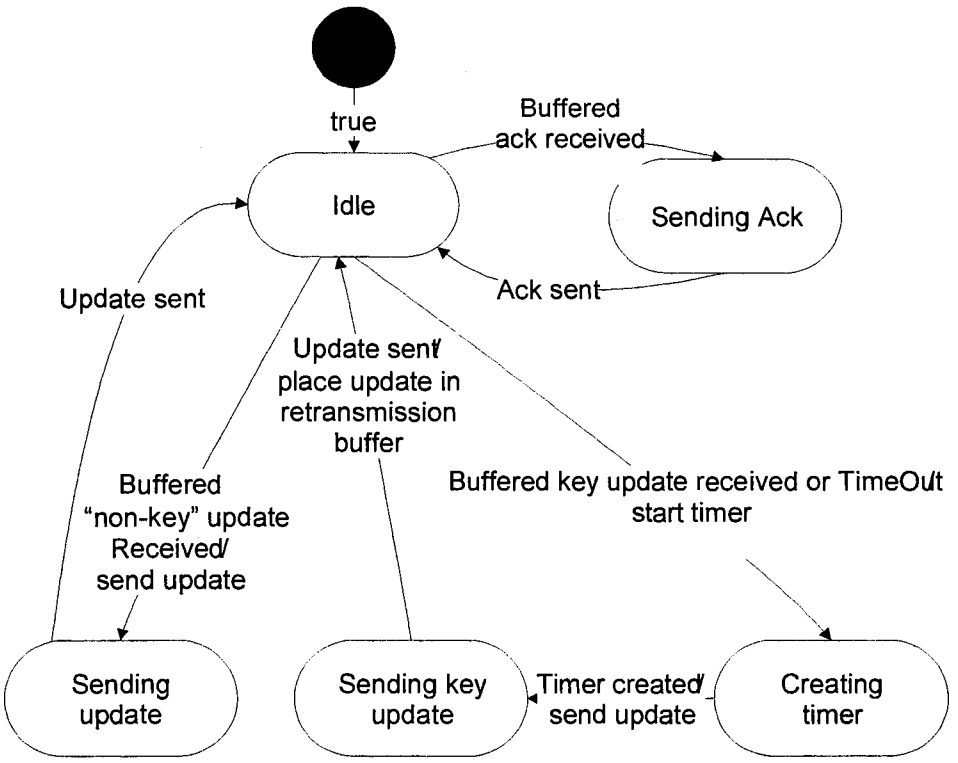


Figure 3-8 – Sender module state machine

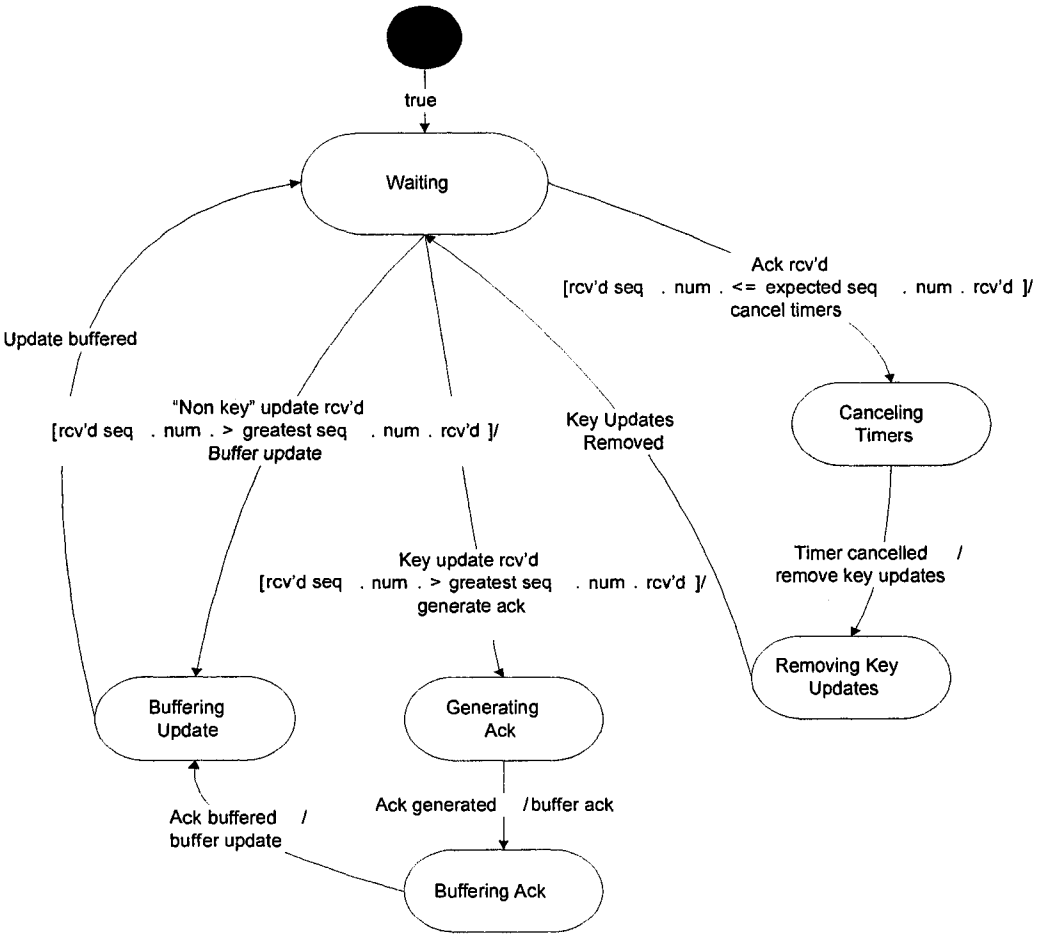


Figure 3-9 – Receiver module state machine

Chapter 4

4 Simulating ALPHAN

4.1 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 5% of the updates are tagged as key updates. The emulated environment contains four objects: A, B, C and D. A is a haptic object whereas B, C and D are graphic objects. This means that objects B, C and D generate 30 updates per second while object A generates 1000 updates per second.

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

4.2 Simulations and Results

The goal of the simulations is to prove the effectiveness of the Multiple Buffering scheme introduced in chapter 3 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. Every key update received is counted only once, which means that duplicate key updates received are not counted.

Figure 4-2 shows the results of the first simulation which evaluates the performance of the multiple buffering scheme with respect to delay and jitter. Note that although there are four objects in the environment, only the results of two objects (A and B) are shown; this allows us to avoid

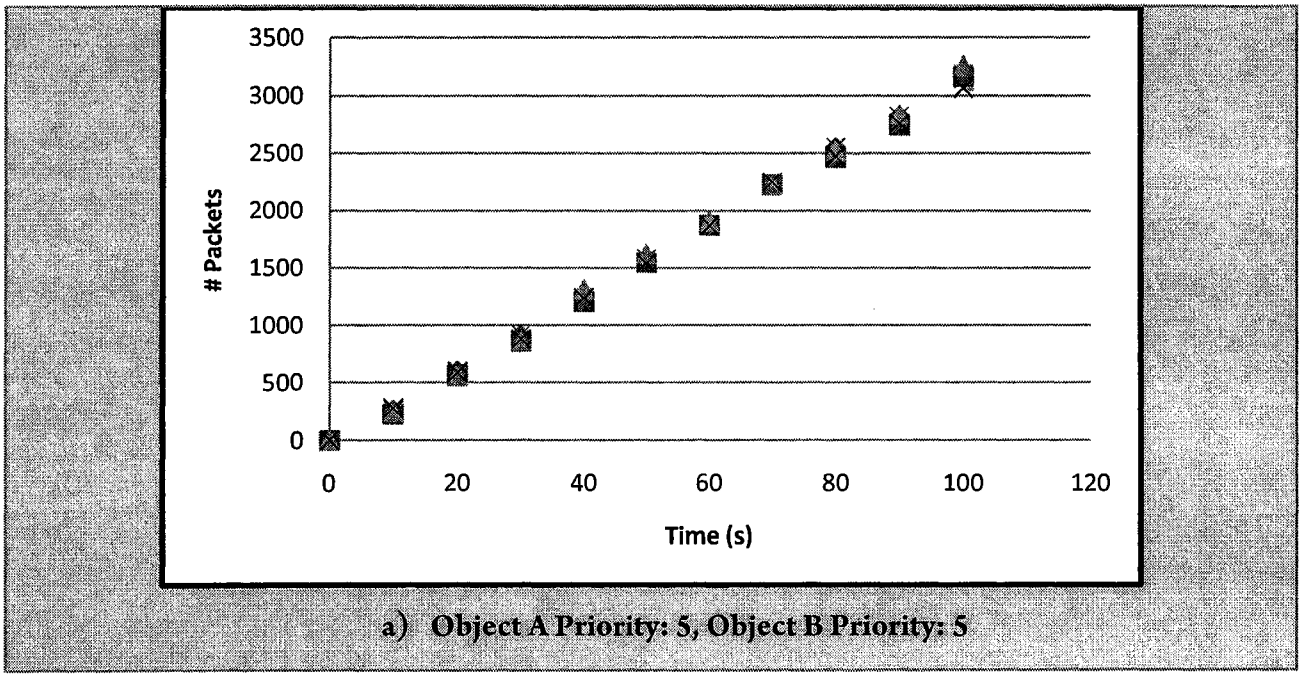
overcrowding the charts with information since the throughput of objects B, C and D are near identical; their updates are sent at the same rate. Figure 4-2(a) shows that in perfect network conditions (delay=0, jitter=1ms, and no packet loss), both approaches perform equally well. As soon as we start 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 A. This is especially clear in Figure 4-2 (f) where the delay is set to 100 ms and jitter to 15 ms with no packet loss.

The second simulation evaluates the performance of the multiple buffering scheme, under lossy network conditions, for haptic data transfer. Figure 4-3 compares the throughput of key haptic updates when both the multiple buffering and single buffering approaches are used. As it can be seen, the multiple buffering scheme can cope better with lossy network conditions; this is especially clear as the percentage of loss increases.

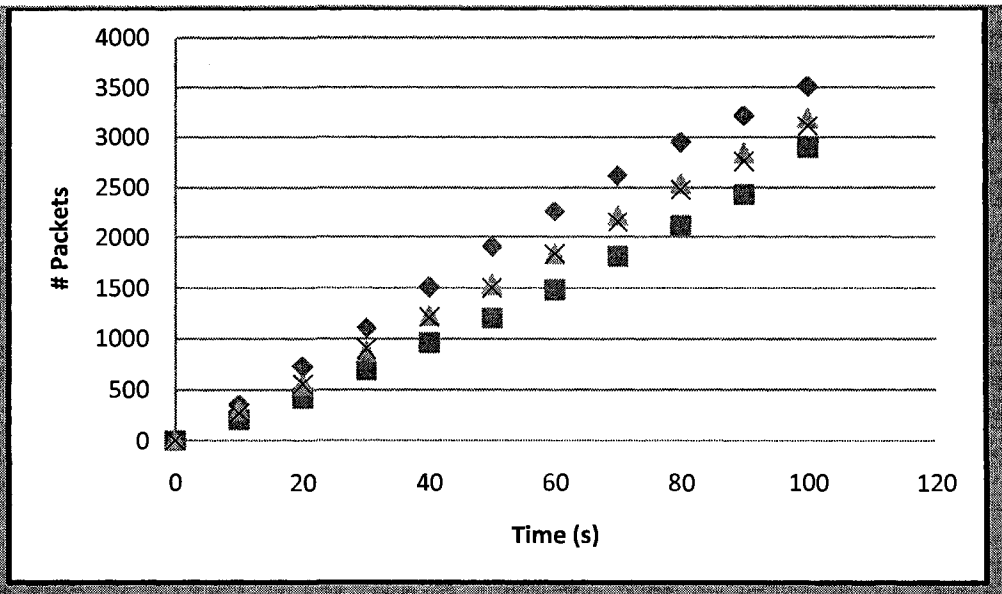
The difference in the performance can be explained as follows: in the single buffering approach, the retransmission buffer can easily fill up with obsolete updates under jittery and lossy conditions whereas in the Multiple Buffering approach, unwanted key updates are strategically discarded according to the properties of the object they belong to. Transferring obsolete updates steals the available bandwidth from the fresher ones, especially if these updates end up making it to the receiver side in the first transmission but are not acknowledged on time because of the jitter in the network. These obsolete key updates are sometimes useless even if they make it in the second retransmission and are often discarded by the receiver in favor of newer ones. Also, if we store numerous old updates while maintaining a timer for each, the performance of the application might suffer. The only way to limit the size of the retransmission queue is by stopping to produce newer key updates in order to service the old ones stored in the retransmission queue, this only can be done if we block the sending queue, which might further aggravate the performance problem. Moreover, as the retransmission queue grows bigger, it starts to compete for bandwidth with the sending queue in order to reduce its size.

Another advantage of multiple buffering over single buffering is the fact that we can attribute priorities to objects in order to give precedence during transmission for higher priority objects over lower priority ones. We emulate an environment containing two objects that transfer data at an approximate rate of 30 updates per second (graphic loop rate). It is assumed in this scenario that

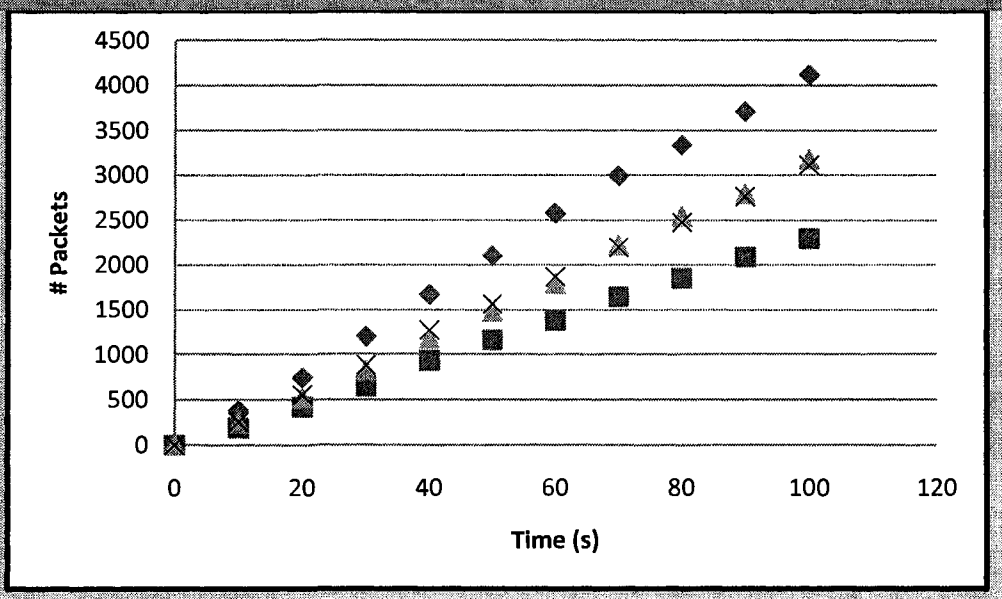
object A is more important than object B with respect to the application. In the first run, the priorities of Objects A and B were kept at 5 (maximum priority). In the subsequent runs, object B's priority is decreased while A's priority is kept at 5. We measure the total amount of updates received for each object with respect to time. Duplicate updates are not counted. Figure 4-1 shows the result of the simulation performed under ideal network conditions. As the priority of object B decreases, its updates are delayed in favor of A's updates. In the single buffering approach, the precedence in packet transmission cannot be given to any of the objects as both object's updates are located in the same sending queue.



a) Object A Priority: 5, Object B Priority: 5



b) Object A Priority: 5, Object B Priority: 4



c) Object A Priority: 5, Object B Priority: 3

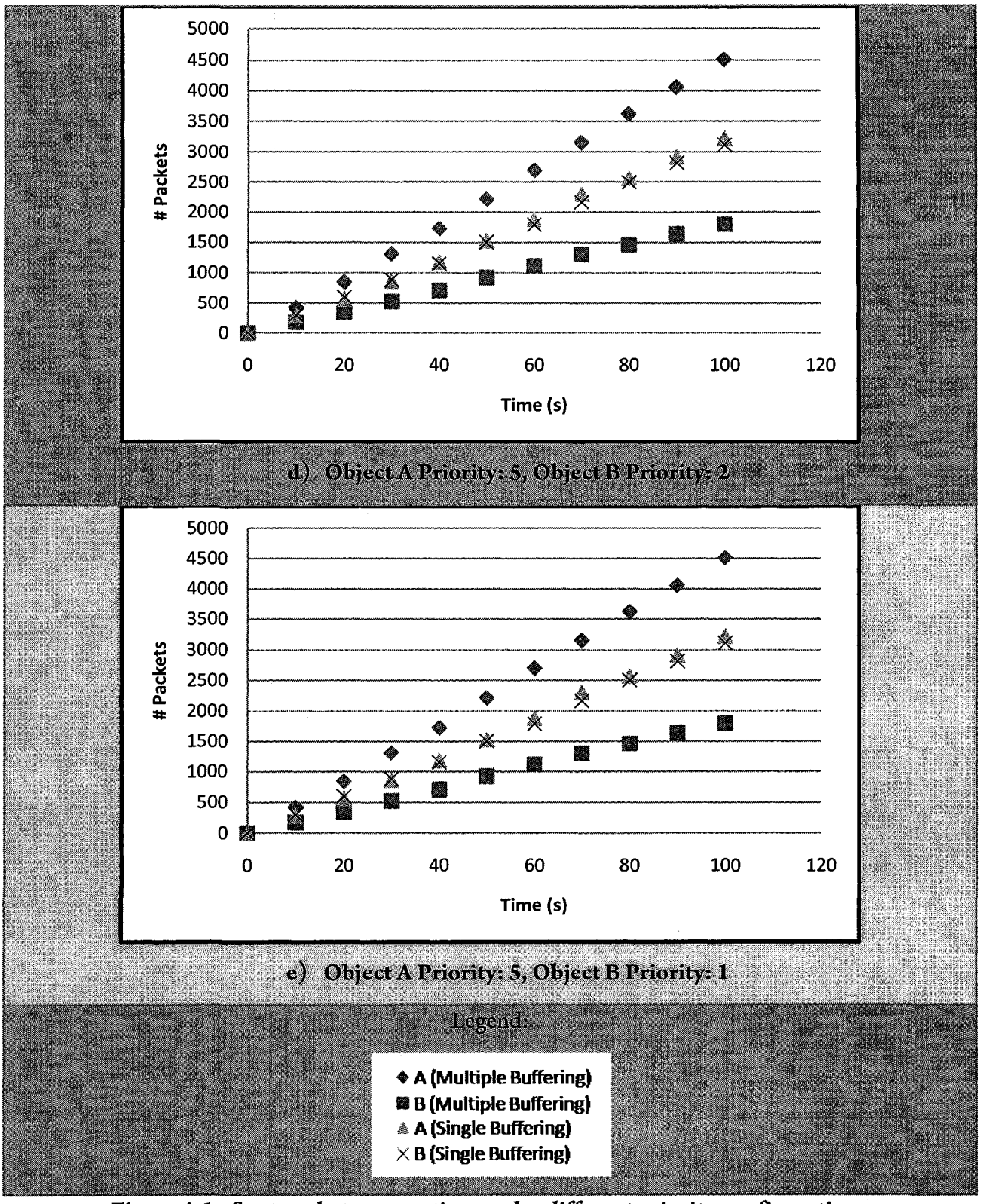
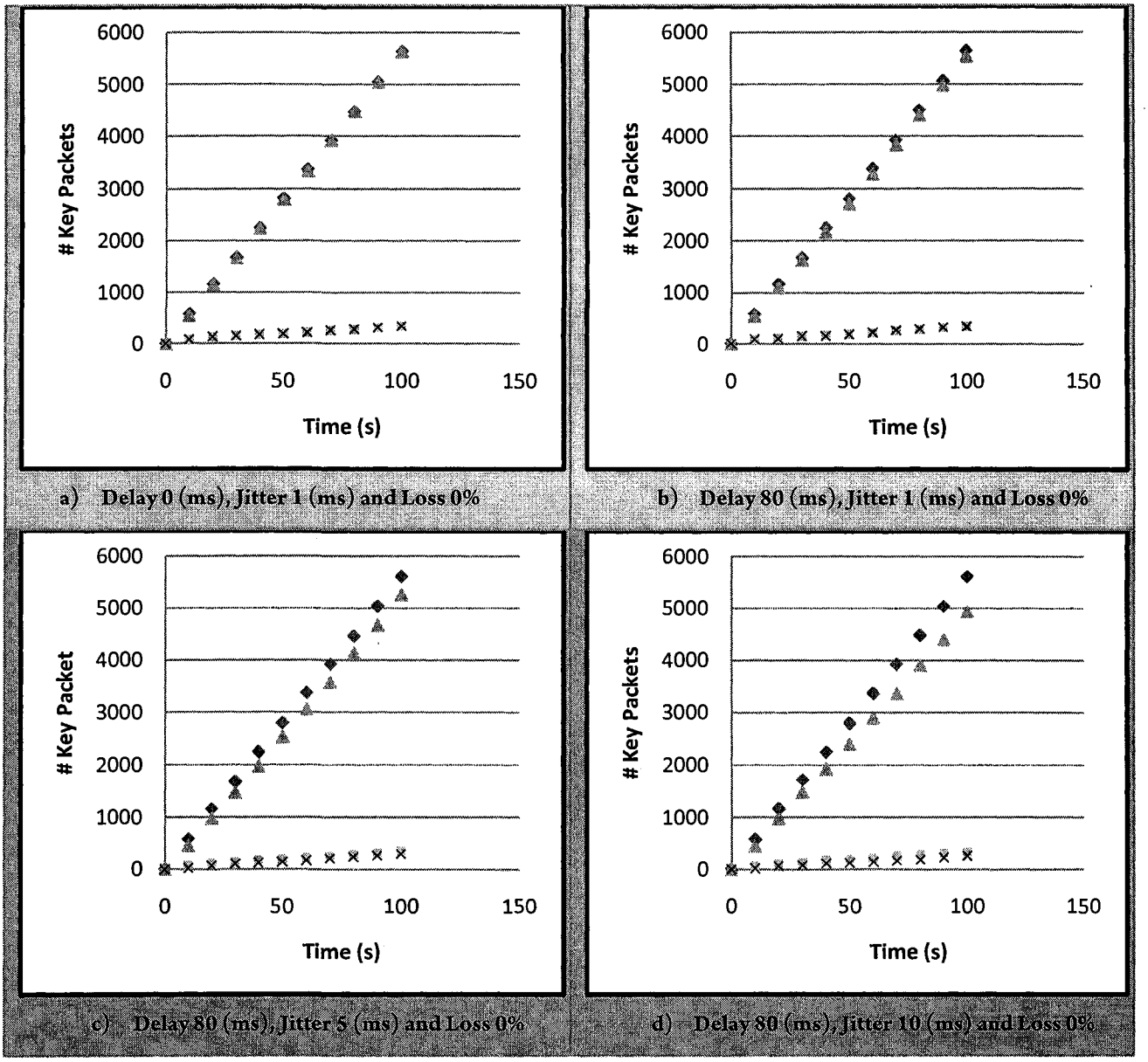


Figure 4-1 - Sent packets versus time under different priority configurations



a) Delay 0 (ms), Jitter 1 (ms) and Loss 0%

b) Delay 80 (ms), Jitter 1 (ms) and Loss 0%

c) Delay 80 (ms), Jitter 5 (ms) and Loss 0%

d) Delay 80 (ms), Jitter 10 (ms) and Loss 0%

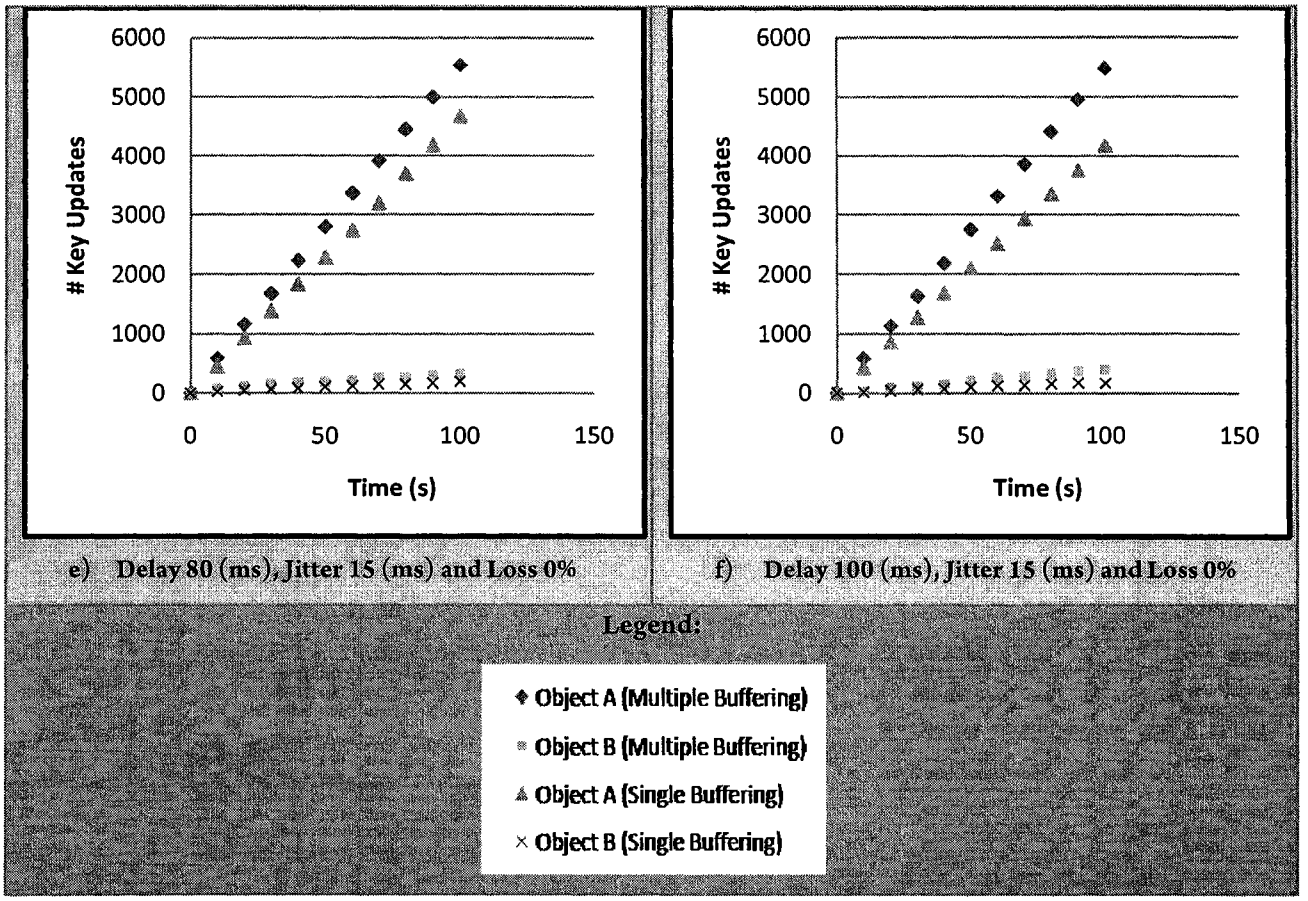
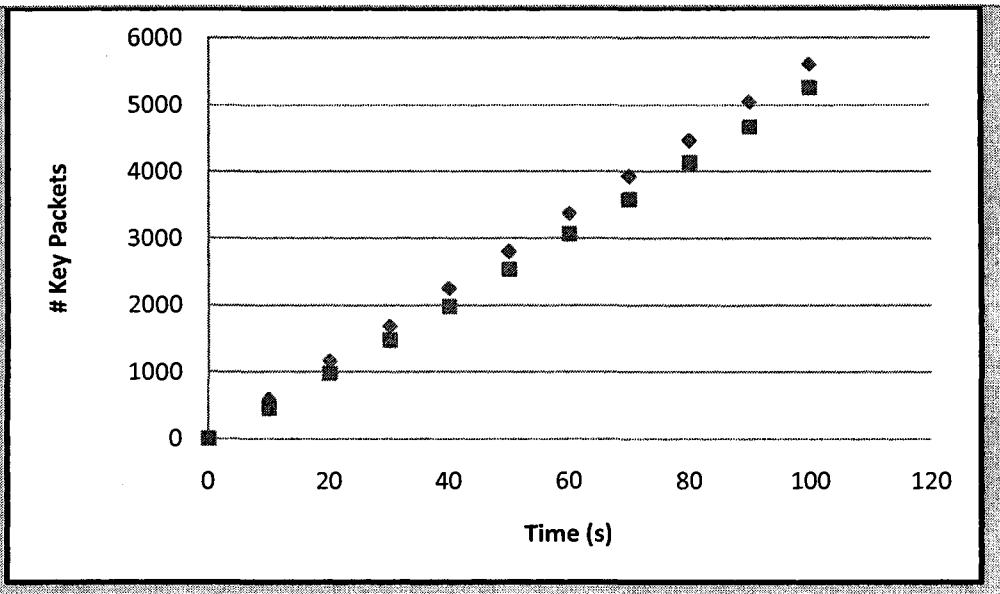
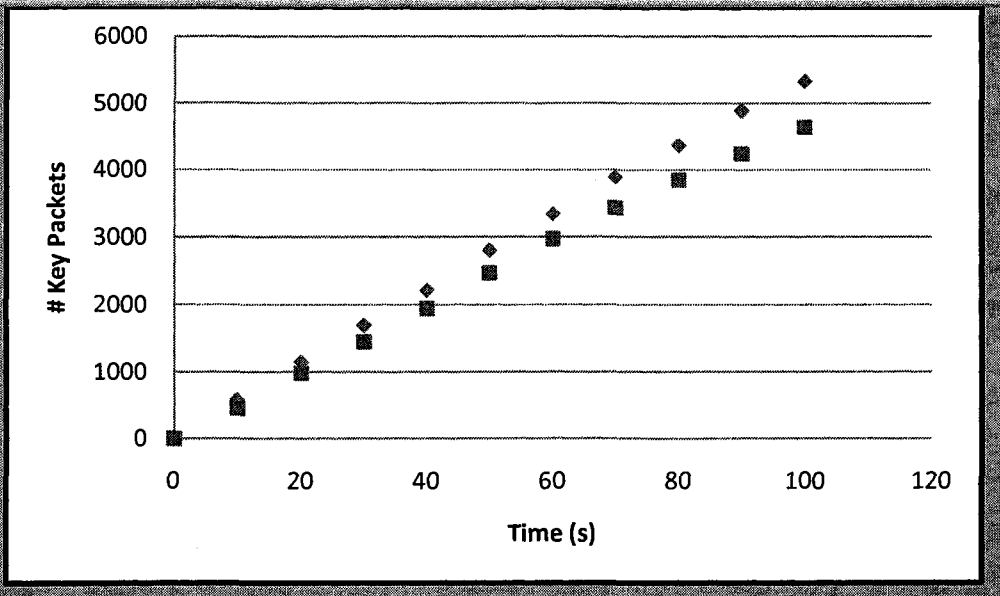


Figure 4-2 – Key Packets throughput for various delay and jitter configurations



a) Delay 80 (ms), Jitter 5 (ms) and Loss 0%



b) Delay 80 (ms), Jitter 5 (ms) and Loss 3%

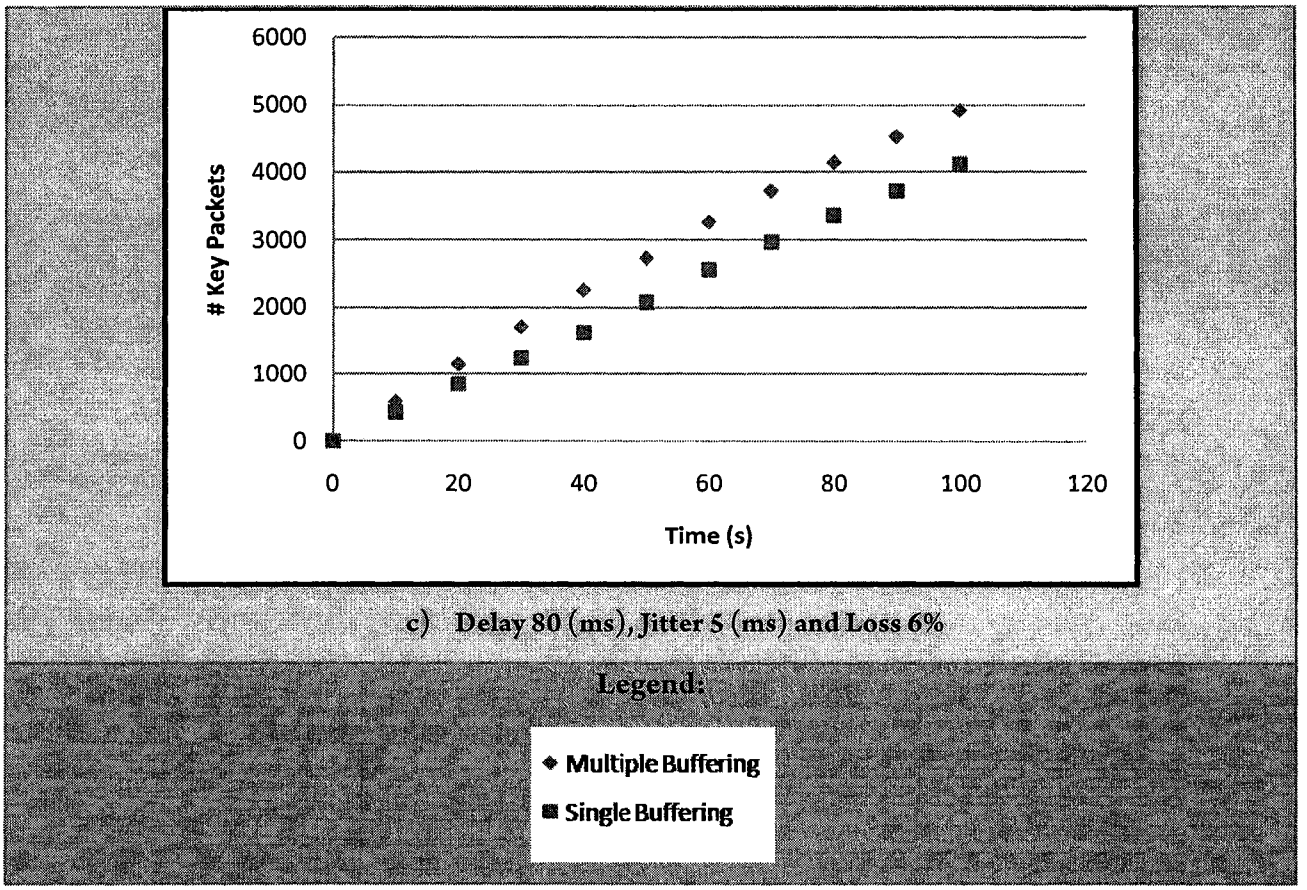


Figure 4-3 – Key Packets throughput for various network loss conditions

Chapter 5

5 "Balance Ball" Gaming Application

In order to further evaluate the performance of ALPHAN, we developed a cooperative haptic gaming application called "Balance Ball". The application is inspired by the children's game bucket on a board. A bucket is filled with water and placed on a long wooden board that is held by two players from each side. The two participating players must pass over obstacles while carefully balancing the board so that a minimal amount of water is dropped from the bucket. In our application, a virtual adaptation of the game is developed wherein the bucket of water is replaced with a ball. A snapshot of the game is shown in figure 5-1.

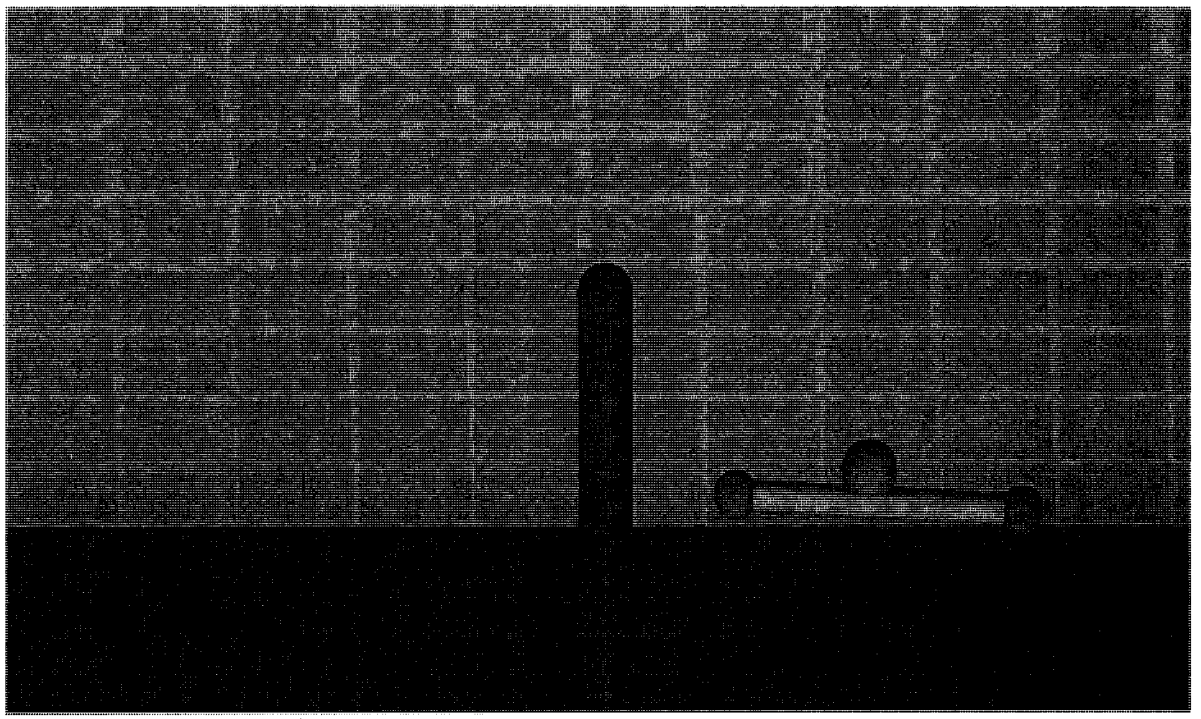


Figure 5-1 – Balance Ball game snapshot

5.1 The "Balance Ball" game snapshot

The virtual board is held from each side by a haptic device. The two haptic devices reside on geographically distant workstations. The performance of the players is graded according to two

metrics: the time it takes the players to complete a particular task and how well the ball is maintained in the center of the board during the execution of the task. In order to grab the board, the player touches an end of the board and presses a button on the haptic device in order to attach the haptic proxy to that end.

The simulation game sends the position of the haptic device to the other party and the haptic rendering is performed locally at each side of the network. To ensure consistency, the graphic data of both scenes is also exchanged yet at a lower rate. As per haptic rendering, the application is developed in C++ using the OpenHaptics API [44]. Figure 5-2 presents an overview of the forces at stake. F1 and F2 are the forces felt by player 1 and player 2 respectively. A player holding one end of the board would experience many forces that mimic the real world feeling. The gravity force generated by the weight of the board is constant during the simulation since at any point in time, the player is always carrying half of the board weight. The gravity force generated by the weight of the ball is calculated according to the position of the ball on the board. It is natural that the player closer to the ball would carry more of its weight. The mutual force applied by one player on the board is instantly felt by the other. A constraint force is also applied to restrict the movement of the player by adhering to the physical limitations of the board. Other forces are also felt by the user when a collision occurs. The friction coefficient that dictates the friction forces between the ball and the board is an important factor since it affects the velocity of the ball movement and thus the difficulty level of the game.

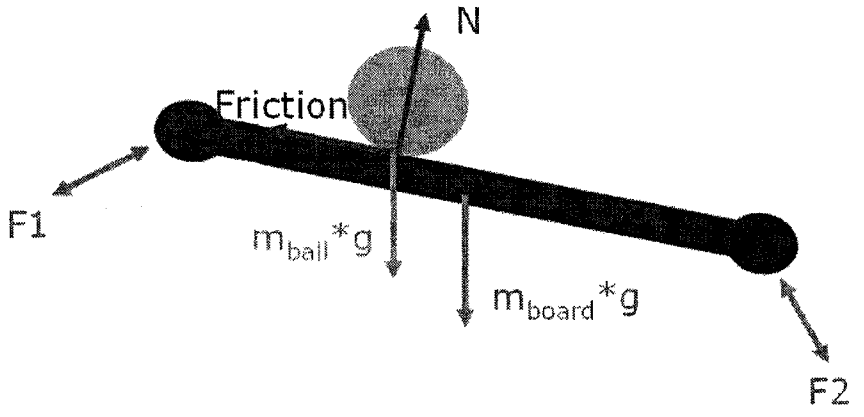


Figure 5-2 – Overview of the forces at stake

5.2 *Implementation Details*

As we have previously mentioned, only position information is exchanged between the two ends of the communication. It is assumed that the ball is owned by workstation A and thus it continuously sends the position of the ball to workstation B at the graphic loop rate. Both ends of the communication send the position of their proxies respectively to each other. Before attaching a haptic proxy to a board edge, its position information is sent at the graphic loop rate, after attaching it, it is sent at the haptic loop rate. These position updates will now be used to calculate forces on the other side and thus must be frequently sent. The incoming proxy position is used to calculate the forces the sender player has exerted on the receiver player by moving her/his proxy form point x to point y. These are called the mutual forces. Equation 6-1 is used to calculate the mutual force by relying on previously produced forces in order to avoid sudden jumps caused by either network loss or jitter. Fm represents the mutual force and λ is a heuristic value between 0 and 1.

$$FinalForce = \lambda \times Fm(t) + (1 - \lambda) \times Fm(t - 1) \qquad \textbf{Equation 5-1}$$

Figure 5-7 shows the ALPHAN settings used during the simulation. In order to conserve bandwidth, several "Normal Updates" are buffered and sent in one packet. This reduces the overhead created by the addition of a header for each update. On the other hand, buffering a large amount of updates before transmitting them can result in hindering the responsiveness of the application. Collaborative haptic applications in particular cannot afford major increases in response time since this would destabilize the application.

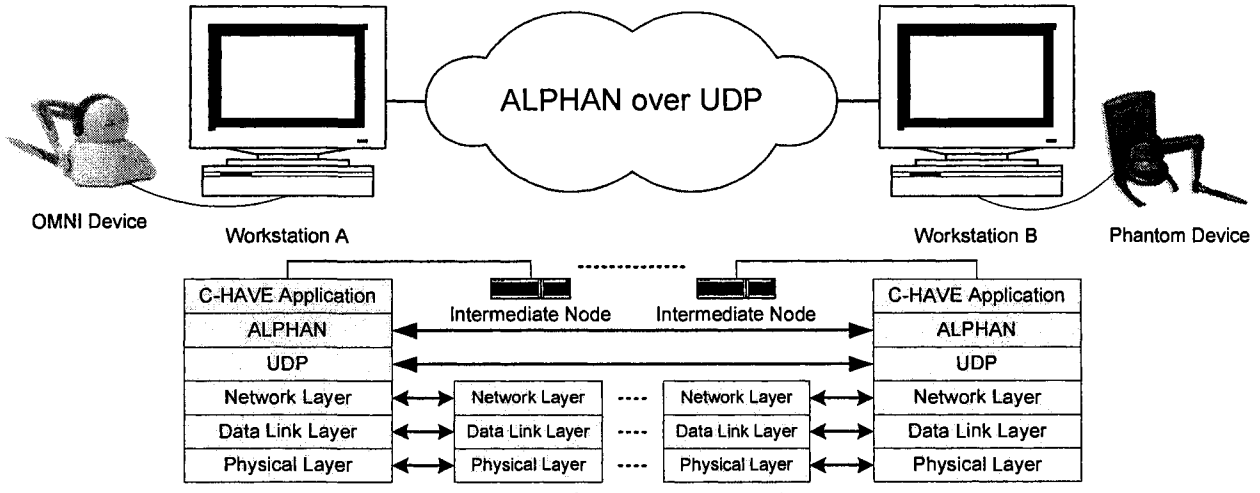


Figure 5-3 – The experimental setup

Key updates are often chosen using heuristic methods that depend on the nature of the application. In this application, the key updates are chosen at points, on the trajectory curve of the haptic proxy, where the slope evaluates to either zero or infinity, since this would indicate a major change in movement and thus must be carefully conveyed to the other side. Figure 5-4 shows an example of a possible trajectory for a haptic device attached to one end of the board.

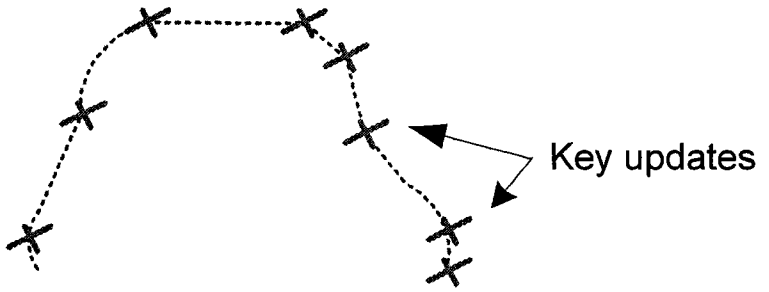


Figure 5-4 - Possible haptic device trajectory

5.3 *Performance Evaluation*

5.3.1 Task Objectives and Description

The objective of the designed task is to measure the level of collaboration and co-presence between the two players when the communication is handled by ALPHAN. Two users have to carry the board and move it across a barrier as shown in figure 5-1. The task is simulated in the presence of gravity and friction forces using two haptic devices. The collaboration is imposed by the gravity: if both users do not tightly synchronize their actions, the board deviates from the horizontal direction and eventually the ball moves away from the center of the board towards the lower side of the board. We perform the same task in two cases: first with perfect network conditions (no delay or jitter) secondly in the presence of constant end-to-end delay and jitter. Finally, we perform a comparison between the two cases and assess the performance of ALPHAN.

5.3.2 Experimental Setup

For the experiment we used two Pentium 4 PCs with 2 Gb RAM and 100Mbs Ethernet cards. The two haptic devices were the Phantom Omni and the Phantom Desktop, developed and marketed by

SensAble Devices, Inc. A snapshot of the experimental setup is shown in figure 5-3. Notice that ALPHAN is implemented on the application layer of the network protocol stack. The application resides on top of ALPHAN.

The experiment was conducted on an Ethernet Local Area Network. Network disturbances such as delay and jitter are simulated using a software tool we have developed for this experiment. In order to make use of the jitter smoothing algorithm, the clocks of both workstations must be synchronized. For this purpose, a Network Time Protocol (NTP) server is used. Both workstations maintain a connection with the NTP server in order to synchronize their clocks. The clock synchronization precision obtained for this experiment is within one millisecond.

Twelve subjects (forming six teams of two members each) took part in the experiment; all of them are graduate students from the engineering and computer science school. No particular reward was given to them for their collaboration. Prior to starting to record the experiment sessions, we asked the teams to perform "rehearsals" just to eliminate any inconsistencies in the performance due to the lack of experience with using haptic devices.

5.3.3 Performance Variables

In order to determine the effectiveness of the collaboration between the two users, two main parameters were measured: the task completion time and the variance of the ball position about the midpoint of the board. Based on the variance, we computed a score for the users out of 100. The score is computed based on the assumption that any variance larger than 1000 cm will be considered as zero score (the ball jumps off the board) and for smaller values, the score is calculated based on the relative difference between the actual variance and 1000.

5.4 Results

The two main quality parameters (the task completion time and the score) were collected during the experiment for two cases. Case 1 is by performing the task when no delay is imposed and case 2 where a delay of 50 ms and 10 ms jitter are injected. Table 5-1 shows the variance and corresponding scores for the six teams for both cases. It is clear from the table that the scores decrease when the delays are introduced. However this decrease is not significant in most of the cases. The performance of

ALPHAN has compensated for the introduced delay. In particular, the use of the local lag algorithm has decreased the effects of jitter on the simulation.

Figure 5-5 compares the task completion times obtained for each group under perfect network conditions and when a delay of 50 ms and jitter of 10 ms is introduced. The diagram shows that the task completion time has increased slightly which implies that the performance of the users has decreased. Figure 5-5 also shows the task completion time when the same simulation was performed with the local lag mechanism disabled. As it can be seen, smoothing the effect of the jitter using local lag has had a positive effect on the overall performance of the users.

Figure 5-6 shows the variations of the variance for Team 1 with and without the delay. The variations in the case where a delay was injected did not differ significantly from those with no delays.

Table 5-1 - Team scores with and without delays

Teams	**Without delay**		**With delay**	
	Variance (cm)	**Score (%)**	**Variance (cm)**	**Score (%)**
Team 1	198.04	80.19	252.16	74.78
Team 2	435.23	56.47	533.37	46.66
Team 3	414.22	58.57	540.47	45.95
Team 4	876.52	12.34	896.22	10.37
Team 5	673.03	32.69	698.50	30.14
Team 6	254.90	74.50	297.88	70.21

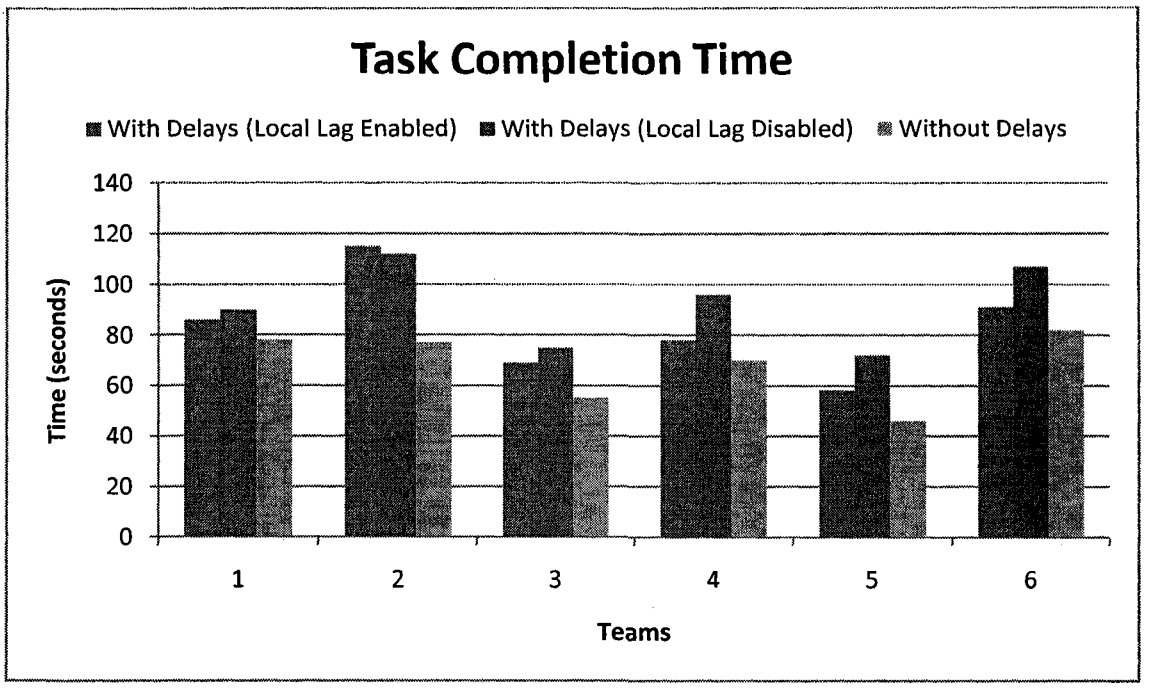


Figure 5-5 - The task completion time without and with 50 ms end-to-end delay and 10 ms jitter

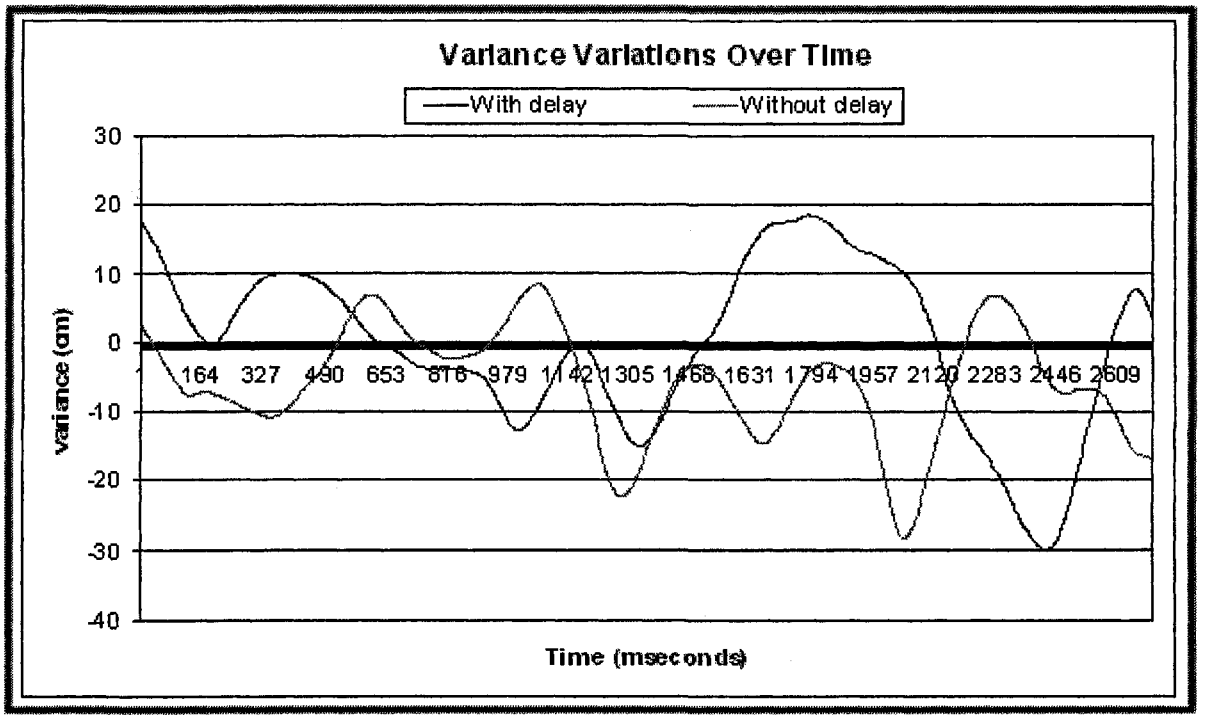


Figure 5-6 - The variations of the variance over time

```
<ALPHAN>
        <network dataPort="8000" controlPort="80001">
                <keyUpdate>
                        <timerMultiplier>1.5</timerMultiplier>
                        <maxRetransmission>3</maxRetransmission>
                </keyUpdate>
                <localLag enabled="true"/>
                <fragmentation enabled="true">
                        <MTU>1552< /MTU >
                </fragmentation>
                <maxUpdateCount>5</maxUpdateCount >
        </network>

        <participants>
                <participant uid="0" ip="137.156.212.64">
                <participant uid="1" ip="137.134.112.112">
        </participants>

        <sharedObjects>
                <object uid="0" name="Ball" priority="5">
                <object uid="1" name="Board" priority="3">
                <object uid="2" name="haptic device of first player" priority="5">
                <object uid="3" name=" haptic device of second player " priority="5">
        </sharedObjects>

        <buffers>
                <buffer retransmissionQueueSize="4">
                        <mappedObjectUid>0</mappedObjectUid>
                        <mappedObjectUid>1</mappedObjectUid>
                </buffer>
                <buffer retransmissionQueueSize="5">
                        <mappedObjectUid>2</mappedObjectUid>
                </buffer>
                <buffer retransmissionQueueSize="5">
                        <mappedObjectUid>3</mappedObjectUid>
                </buffer>
        </buffers>
</ALPHAN>
```

Figure 5-7 – ALPHAN settings for the "Balance Ball" application

Chapter 6

6 Multi User Experiment

6.1 Application

In chapter 5, we studied the behavior of ALPAHN for collaborative applications that involve two players. In this chapter, the performance of ALPHAN is examined for a collaborative application that involves three players through the implementation of a C-HAVE benchmark application.

The benchmark application consists of a simple game where three users attempt to lift a 3D triangular shape and place it in a triangular hole (Figure 6-1). The players' performance is evaluated according to the time it takes to perform the task. The application is especially useful for assessing the effect of network impairments on stability and perceived presence since the users will be continuously exerting forces on one another.

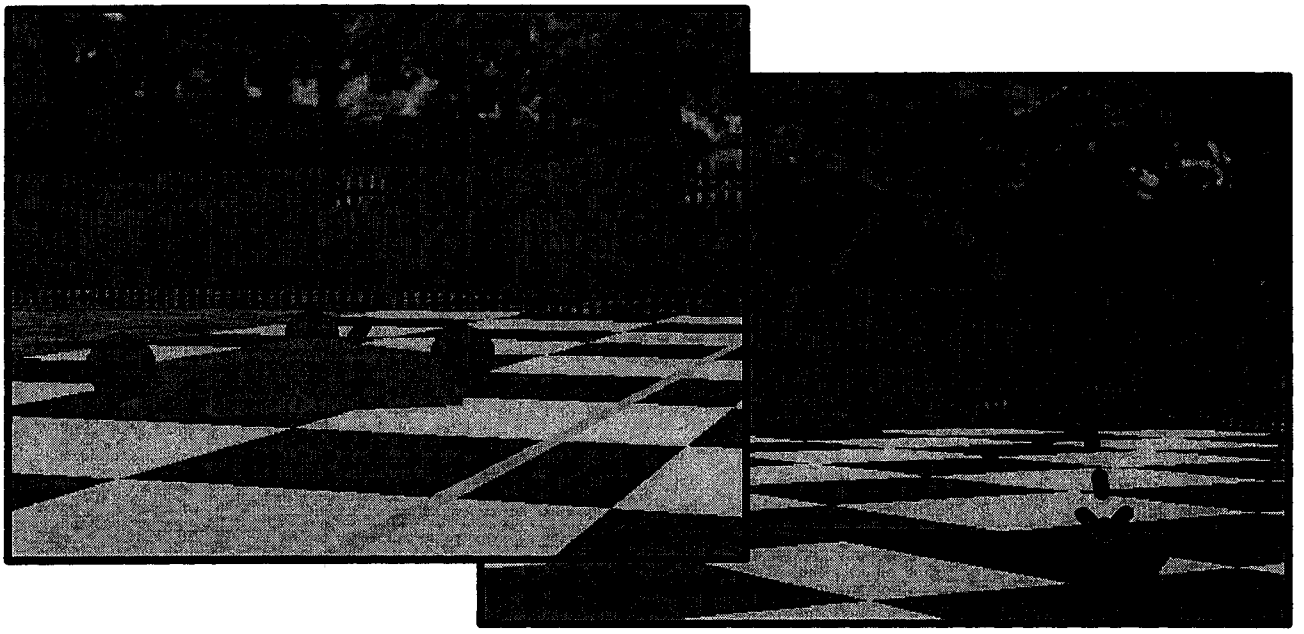


Figure 6-1 - Benchmark application snapshot

6.2 *Implementation*

This application is designed to examine the performance of ALPHAN for C-HAVE applications that require tight collaboration between multiple users. Figure 6-8 shows the initial ALPHAN configuration of the application.

The position data is used to calculate forces and therefore it should be sent at the haptic loop rate. Every haptic loop both the independent and dependent forces are calculated. The independent forces are responsible for simulating the interaction of the triangle with the environment (i.e. the triangle's weight, touching the ground, inserting the triangle into the hole). These forces are independently calculated at each workstation and do not require data transfer. The dependent forces simulate the forces of interaction exerted by the players on each other. They are calculated according to the positions of the participating players which are constantly exchanged at the haptic loop rate. Dependent forces are calculated as follows:

1. Calculate CT: the center of gravity of the triangle to be lifted
2. Calculate CP: the center of gravity of the triangle formed by the three proxies
3. Translate the triangle to be lifted so that CT=CP
4. On each participating workstation, calculate the distance vector Di between each proxy and its respective attachment point (see figure 6-2).
5. Calculate the dependent force using the following formula: Fi=Di*stiffness
6. Apply a low pass filter on the dependent forces being calculated in order to reduce instabilities.

6.3 *Framing*

For this application, two types of updates are used: normal update and key updates. Normal updates are sent unreliably while key updates are sent reliably. In order to conserve bandwidth, several normal updates are buffered and sent in one packet. The key updates are chosen at points on the trajectory curve of the haptic proxy, where the slope evaluates to either zero or infinity, since this would indicate a major change in movement and thus must be carefully conveyed to the other users. Key updates are

also used to communicate other important events when a user attaches to a triangle or when the target is reached.

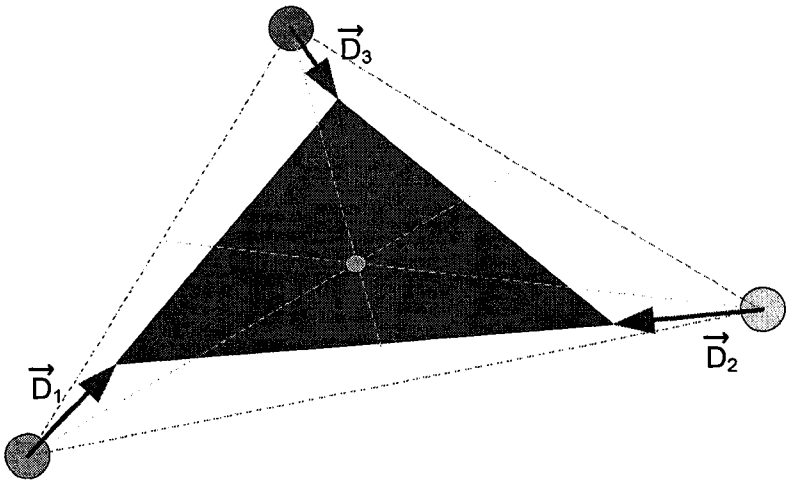


Figure 6-2 - Dependent Force Calculation

6.4 *Multiple Buffering Scheme*

The concept of multiple buffering introduced in chapter 3 is used in this simulation. For the purpose of this simulation, each user has two sending buffers (each sending buffer is composed of a sending queue and retransmission queue). In the case of the owner, the first sending buffer is used to hold the data that describes the state of the triangle. This buffer is given the highest priority. The second buffer holds general event data (attachment, target reached, collision with ground, etc) which are sent at a much lower rate and have a lower priority. The other two users have a buffer to hold data representing their haptic proxies' locations while the second buffer holds event data.

6.5 *Experiment*

6.5.1 Setup

Three Pentium 4 PCs with 2 GB of RAM and 100 Mbs Ethernet cards were used for the experiment. The haptic devices consisted of two Phantom Omnis and one Phantom Desktop, all developed and marketed by SensAble Devices, Inc. A snapshot of the experimental setup is shown in figure 6-3.

The experiment was conducted on an Ethernet Local Area Network. Network disturbances such as delay and jitter were simulated using a software tool we developed for this experiment. In

order to make use of the jitter smoothing algorithm (local lag), the clocks of both workstations were synchronized. For this purpose, a Network Time Protocol (NTP) server was used. Both workstations maintained a connection with the NTP server in order to synchronize their clocks. The clock synchronization precision obtained for this experiment was comfortably within one millisecond.

Fifteen subjects (forming five teams of three members each) took part in the experiment; all of them were undergraduate students from the School of Information Technology and Engineering, University of Ottawa. No particular reward was given to them for their collaboration. Prior to the start of recording the experiment sessions, we asked the teams to perform "rehearsal" sessions just to eliminate any inconsistencies in the performance due to a lack of experience in using haptic devices. During the simulation, the users were neither allowed to communicate verbally nor to see each other.

6.5.2 Evaluation Criteria

The users' performance is calculated according to time it takes them to complete the simulation. The following metrics were collected during and after the simulation:

- Time to complete: refers to the amount of time it takes to perform the task.
- Perceived presence: refers to how well is the user able to feel, through touch, the existence of other users.
- Stability: relates to the steadiness of the haptic devices throughout the simulation.

Both the perceived presence and stability metrics are subjective. They are collected with the help of a survey given to the users at the end of the simulation.

6.5.3 Results

The first experiment consisted of conducting the simulation under different delay conditions without injecting any jitter into the network. Each time the experiment was performed by a team, the time to complete metric was logged. The experiment was conducted twice by each team; first, with multiple buffering enabled and then with multiple buffering disabled. The results collected from all teams were averaged to produce a single time to complete value for each delay condition (see figure 6-4). Also, the users were handed a questionnaire to assess the stability and perceived presence of the simulation

on a scale from 0 to 10. Both parameters were well explained to the users. These were also averaged to produce one stability and perceived presence value for each delay condition (see figure 6-5).

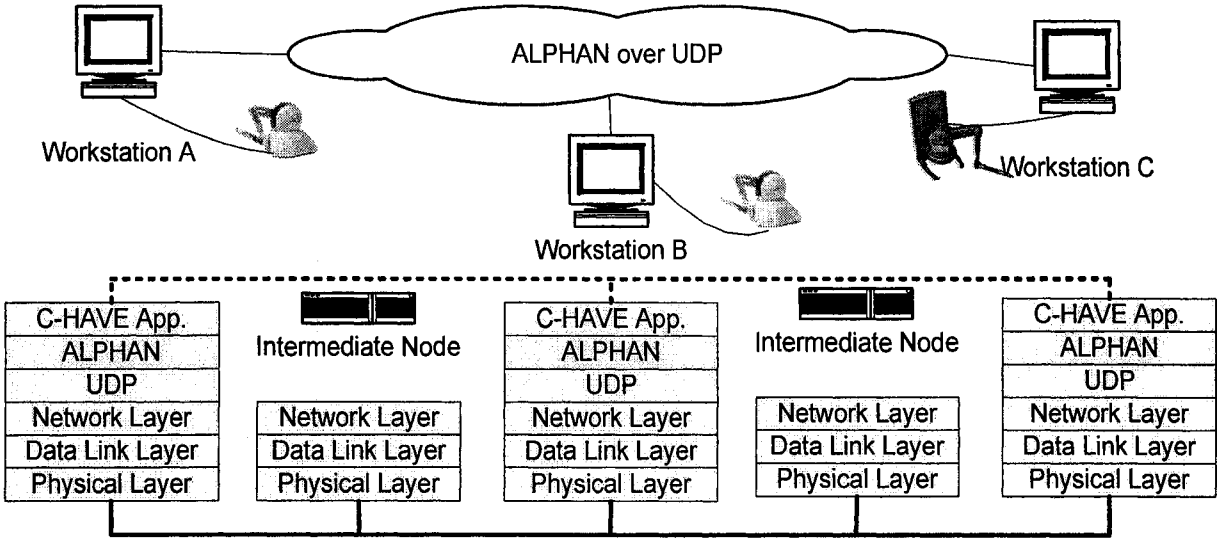


Figure 6-3 - Experimental setup

As expected, the performance deteriorates with the injection of more delay into the simulation. This is a direct result of the decrease in the stability and perceived presence of the simulation. As it can also be deduced from the results, enabling multiple buffering does not result in a dramatic increase in the users' performance. The multiple buffering scheme has been proven to increase performance especially in environments that include a multitude of objects with varying priorities; each object is usually allocated an independent sending buffer. Our environment only includes one object (although another buffer was allocated for important events). Figure 6-4 also shows that the stability of the simulation, as perceived by the users was well maintained as the delay increased, as opposed to the perceived presence that dropped significantly. Some users reported almost not feeling the presence of the other users as the delay increased to 150 ms.

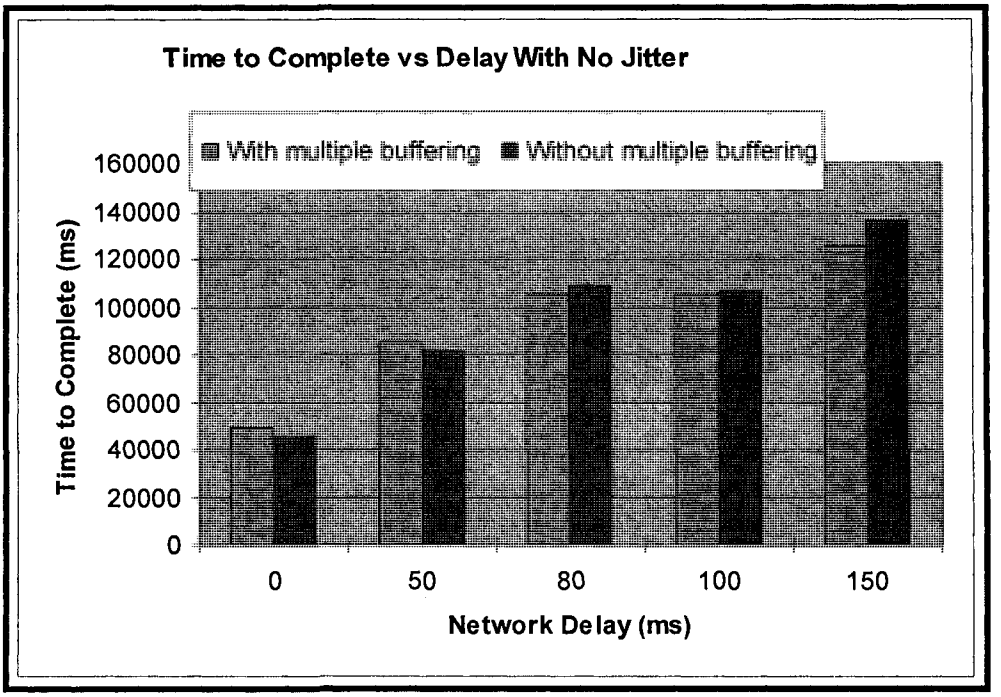


Figure 6-4 - Time to complete task vs. Network delay

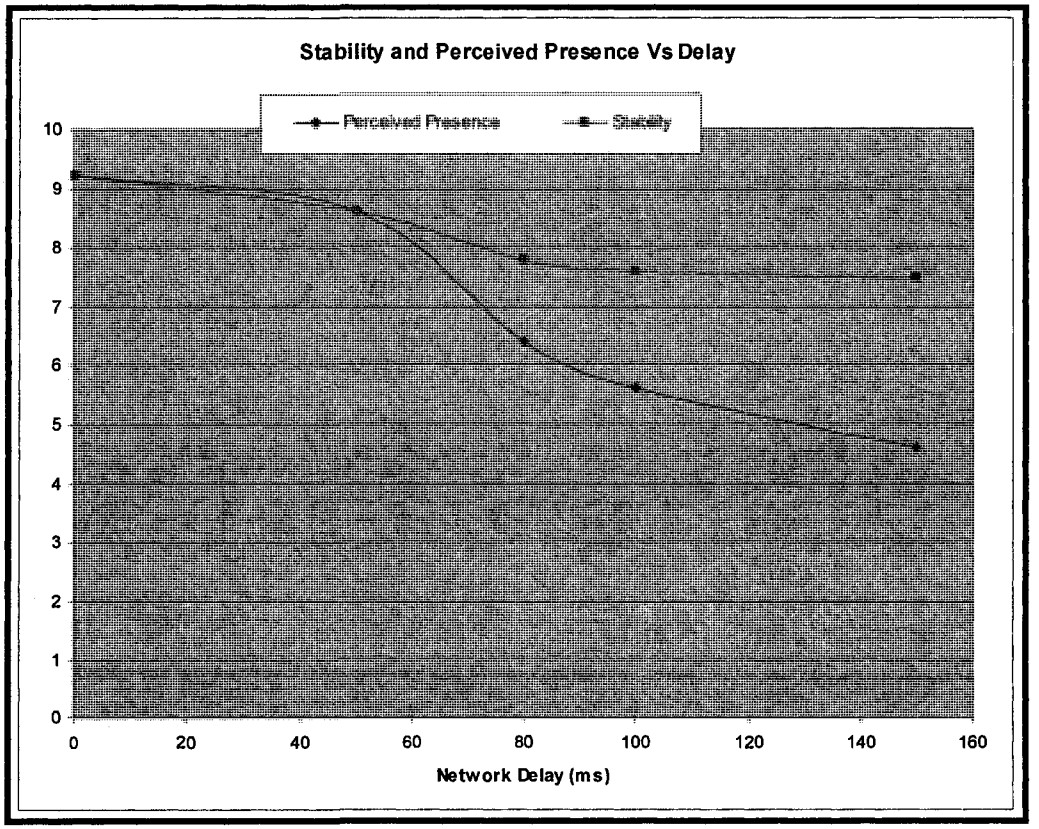


Figure 6-5 - Time to complete task vs. Perceived presence and Stability

The second experiment concentrated on the evaluation of the effect of jitter on multi-user collaborative haptic applications.

The delay was set to the constant value of 50 ms, while the jitter varied. Figure 6-6 shows these results. The enablement of the local lag algorithm has a dramatic effect on the performance of the users as the jitter increase. The local lag algorithm smoothes the jitter effect. This decreases any instability caused by jitter.

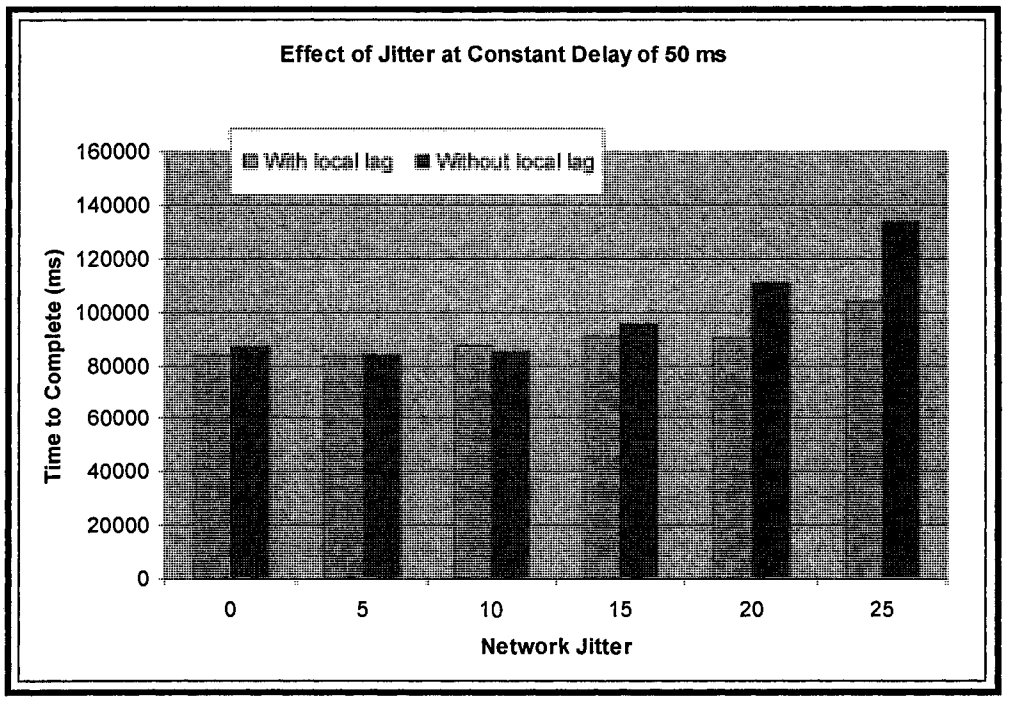


Figure 6-6 - Time to complete task vs. Network jitter

The third experiment consisted of performing the first experiment again, only this time, the application was modified to support only two users. Figure 6-7 shows a comparison with between the "time to complete" values obtained when the experiment was performed with the participation of two and three users respectively. The simulation with two users shows a significantly better performance. This shows that the effect of network impairments is amplified with the addition of more users. Nonetheless, the improvement in performance cannot only be attributed to network factor. In fact, it is much easier to conduct the simulation with two participants rather that three; the users have to collaborate with only another users, which result in less contradicting decision when it comes to choosing the path to be taken to reach the goal. This is obvious since the simulation with two

participants produced better results (smaller "time to complete") even when the network delay was set to 0 ms.

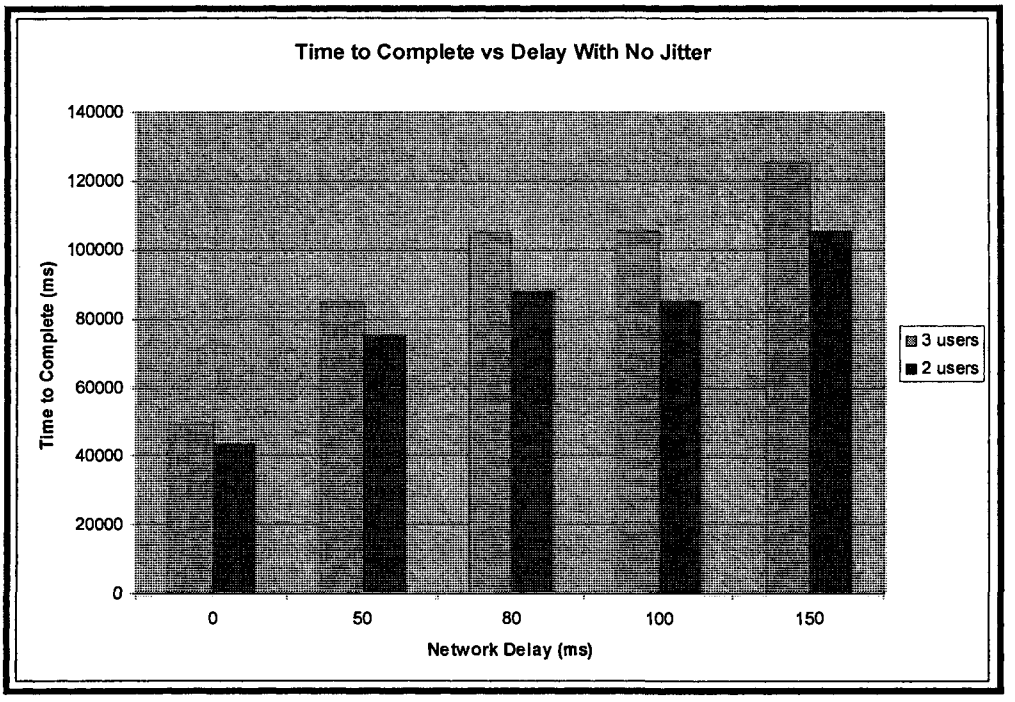


Figure 6-7 - Time to complete task vs. Network delay

```
<ALPHAN>
        <network dataPort="8000" controlPort="80001">
                <keyUpdate>
                        <timerMultiplier>1.5</timerMultiplier>
                        <maxRetransmission>3</maxRetransmission>
                </keyUpdate>
                <localLag enabled="true"/>
                <fragmentation enabled="true">
                        <MTU>1552< /MTU >
                </fragmentation>
                <maxUpdateCount>5</maxUpdateCount >
        </network>

        <participants>
                <participant uid="0" ip="137.156.212.64">
                <participant uid="1" ip="137.134.112.112">
                <participant uid="2" ip="137.153.124.21">
        </participants>
```

```
<sharedObjects>
        <object uid="0" name="Triangle" priority="5">
        <object uid="1" name="haptic device of first player" priority="5">
        <object uid="2" name=" haptic device of second player " priority="5">
        <object uid="3" name=" haptic device of third player " priority="5">
</sharedObjects>

<buffers>
        <buffer retransmissionQueueSize="1">
                <mappedObjectUid>0</mappedObjectUid>
        </buffer>
        <buffer retransmissionQueueSize="5">
                <mappedObjectUid>1</mappedObjectUid>
        </buffer>
        <buffer retransmissionQueueSize="5">
                <mappedObjectUid>2</mappedObjectUid>
        </buffer>
        <buffer retransmissionQueueSize="5">
                <mappedObjectUid>3</mappedObjectUid>
        </buffer>
</buffers>
</ALPHAN>
```

Figure 6-8 - ALPHAN settings for the benchmark application

Chapter 7

7 Conclusion and Future Works

7.1 Solved Issues

We proposed ALPHAN, an Application Layer Protocol for Haptic Networking over a non-dedicated network such as the Internet. The protocol is made highly customizable by passing application requirements to the protocol in HAML format.

7.1.1 Multiple Buffering

The results have shown (chapter 4) that the use of multiple buffering can result in a more efficient way to queue updates on the sender side. Firstly, it allows for the implementation of a prioritization system for different objects in the environments. Secondly, it permits the implementation of a more efficient algorithm for managing key updates residing in the queues.

7.1.2 Buffering on the Receiver Side

We have concluded that buffering on the receiver side can be essential to implement the local buffering algorithm over jittery networks. On the other hand, buffering should not b performed over non jittery networks and the updates should be forwarded to the application as soon as they are received.

7.1.3 Customizability

When it comes to distributing haptic applications over the network, every application has its own specific set of requirements. For instance, some applications require an especially stable interaction (collaborative applications) while responsiveness is the most crucial factor for others. ALPHAN approaches this issue by providing a large set of customizable parameters. The freedom to seamlessly tailor the communication protocol gives it the necessary strength to support the various, often conflicting requirements of complex haptic applications.

7.1.4 Application Design

When designing a C-HAVE application, the following issues have to be resolved during the specification phase:

- Chose whether to use a peer-to-peer architecture or client-server architecture. It is certainly useful to consider one of the many hybrid architectures, some of which are presented in chapter 2.
- Decide whether to use local buffering or not. In this case, not only the nature of the application has to be considered, but also the condition of the network supporting the C-HAVE application.
- Devise a strategy to choose key and non key updates. Key updates are sent reliably as opposed to non key packets. Sending too many unnecessary key updates can result in clogging up the network with acknowledgement updates and filling up the retransmission queues.
- Decide up to how many updates can be sent in one packet. Appending too many updates into one packet decreases the responsiveness of the applications since all these updates have to buffered and delayed on the sender side before being transferred. On the other hand, when bandwidth is limited, grouping many updates into one packet can be crucial to conserve bandwidth.
- Decide whether to synchronize the clocks of the participating workstations. Clock synchronization is essential for two purposes:
    - Implement jitter smoothing algorithms like local lag
    - Implement consistency algorithms like time wrapping
- Decide whether to make use of the multiple buffering algorithm. If the multiple buffering algorithm is to be used, the C-HAVE application designer has decide how many buffers to use, and how to map the objects to the buffers.

7.1.5 Preserving Bandwidth

Preserving bandwidth can be crucial when it comes to transferring haptic data over non-dedicated networks such as the internet. ALPHAN presents three mechanisms to preserve bandwidth:

- Update accumulation: buffering a number of updates on the sender side before transferring them in one packet to the receiver side. This will reduce the overhead created by the update's header.
- Incremental updates: these updates hold incremental information with respect to the last received key update. They are especially useful when key updates are being often sent; this would produce small size incremental updates.
- Lowering the rate of updates: the ideal rate for haptic interactions is generally assumed to be 1000 Hz. Nonetheless the rate can be reduced for several applications, especially the ones that do not require collaboration. Of course the rate reduction often comes at the cost of the stability of the haptic experience.

Other compression and signal prediction algorithms can be implemented on top of ALPHAN in order to reduce the amount of data being transferred.

7.2 Future Works

Our immediate future work will concentrate on implementing various C-HAVE applications on top of ALPHAN in different fields in order to identify possible improvements for the protocol. The next step is to concentrate on supporting the highest number of possible users in a C-HAVE environment. To achieve such goal, it is essential to investigate the different application layer multicasting schemes in order to develop the most efficient approach to multicast haptic packets over non-dedicated networks under different conditions.

We will also concentrate on extending ALPAHN to support other modalities, especially the transfer of video frames. Incorporating support for the Moving Picture Experts Group (MPEG) protocol can prove very useful for such purpose.

Any improvements to ALPHAN will always follow the philosophy of the protocol in terms of achieving the maximum customizability in order to attain the maximum support for the varying C-HAVE applications.

8 Bibliography

[1] B. Chebbi, D. Lazaroff, F. Bogsany, P. X. Liu, N. Liya, and 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.

[2] 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.

[3] Burdea G. and Coiffet Ph., VirtualReality Technology, (2nd edition), Wiley, New Jersey, 2003

[4] Brave, S. and Dahley, A. "inTouch: a medium for haptic interpersonal communication." In Proceedings of ACM CHI '97. 1997, Atlanta, GA: ACM Press: pp. 363 – 364.

[5] C., Basdogan, S., De, J., Kim, M., Manivannan, H., Kim, and M.A., Srinivasan, 'Haptics in minimally invasive surgical simulation and training', IEEE Computer Graphics and Application, Volume 24, Issue 2, Page(s):56 – 64, 2004.

[6] Broll, W., "Interacting in Distributed Collaborative Virtual Environments", Proceedings of IEEE the Virtual Reality Annual International Symposium (VRAIS'95), IEEE Computer Society, 1995, pp. 148-155.

[7] Buttolo P., Oboe R. and Hannaford B., "Architectures for Shared Haptic Virtual Environments", Computer and Graphics, Vol. 21, pp. 421-429, 1997.

[8] Oakley, I., Brewster, S. and Gray, P. Can You Feel the Force? An investigation of haptic collaboration in shared editors, EuroHaptics 2001.

[9] 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.

[10] Iglesias, R., Casado, S., Gutierrez, T., Garcia-Alonso, A., Yap, K. M., Yu, W., and Marshall, A. 2006. A Peer-to-peer Architecture for Collaborative Haptic Assembly. In Proceedings of the 10th IEEE international Symposium on Distributed Simulation and Real-Time Applications (October 02 - 04, 2006). DS-RT. IEEE Computer Society, Washington, DC, 25-34.

[11] X. Shen, J. Zhou, N.D. Georganas, "Evaluation Patterns of Tele-Haptics", Proc. IEEE Canadian Conference on Electrical and Computer Engineering, Ottawa, ON, Canada, May 2006.

[12] Hubbold, R. J. 2002. Collaborative stretcher carrying: a case study. In Proceedings of the Workshop on Virtual Environments 2002 (Barcelona, Spain, May 30 - 31, 2002). W. Stürzlinger and S. Müller, Eds. ACM International Conference Proceeding Series, vol. 23. Eurographics Association, Aire-la-Ville, Switzerland, 7-12.

[13] Wai Yu, Rima Tfaily Souayed, Gordon Dodds and Alan Marshall, "Investigation of Network Issues for Distributed Haptic Virtual Environment Applications", IEEE ICTTA'04, 19-23 April 2004, Damascus, Syria.

[14] X. Shen, J. Zhou, A. El Saddik, N.D. Georganas, "Architecture and Evaluation of Tele-Haptic Environments", Proc. 8th IEEE International Symposium on Distributed Simulation and Real Time Applications (IEEE DS-RT 2004), Budapest, Hungary, October 2004

[15] S. Shirmohammadi, N.H. Woo, "Evaluating Decorators for Haptic Collaboration over Internet", Proc. IEEE Workshop on Haptic Audio Visual Environments and their Applications, Ottawa, ON, Canada, October 2004.

[16] Srinivasan, M. A. Haptics in virtual environments: Taxonomy, research status, and challenges. Computer & Graphics, 21(4):393--404, 1997.

[17] J. Marsh, M. Glencross, S. Pettifer and R. Hubbold, "A Network Architecture Supporting Consistent Rich Behavior in Collaborative Interactive Applications", IEEE Transactions on visualization and computer graphics, Vol. 12, no. 3, pp. 405-416, May 2006.

[18] S. Matsumoto, I. Fukuda, H. Morino, K. Hikichi, K. Sezaki, and Y. Yasuda, "The Influences of network Issues on Haptic Collaboration in Shared Virtual Environments", Fifth Phantom Users' Group Workshop, 2000.

[19] I. Goncharenko, M. Svinin, S. Matsumoto, Y. Masui, Y. Kanou and S. Hosoe "Cooperative Control with Haptic Visualization in Shared Virtual Environments", Proceedings of the Eighth International Conference on Information Visualization, pp.533-538,Washington, USA, 2004.

[20] Iglesias R., Casado S., Gutiérrez T., Carrillo A., Barbero J.I. and García-Alonso A., "Analysing different architectures of a distributed environment for assembly simulation", Virtual Concept, pp. 8-10, Biarritz, France (2005).

[21] Mee Young Sung, Yonghee Yoo, Kyungkoo Jun, Nam-Joong Kim, Jinseok Chae, "Experiments for a Collaborative Haptic Virtual Reality," icat, pp. 174-179, 16th International Conference on Artificial Reality and Telexistence--Workshops (ICAT'06), 2006.

[22] Roberts, D and Wolff, R. "Controlling consistency within collaborative virtual environments", Proceedings of Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications, pp. 46-52, Budapest, Hungary, 2004.

[23] J. P. Hespanha, M.L. McLaughlin and G.S. Sukhatme, "Haptic collaboration over the Internet". In M. L. McLaughlin, J. P. Hespanha, & G. S. Sukhatme (Eds.), Touch in virtual environments. IMSC Series in Multimedia. New York: Prentice Hall, 2001.

[24] R.T. Souayed, D. Gaiti, W. Yu, G. Dodds, and A.Marshall, "Experimental Study of Haptic Interaction in Distributed Virtual Environments", Proceedings of EuroHaptics, Springer-Verlag, Munich, 2004, pp. 260-266.

[25] Jung Kim, Hyun Kim, Boon K. Tay, Manivannan Muniyandi, Joel Jordan, Jesper Mortensen, Manuel Oliveira, Mel Slater and Mandayam A. Srinivasan, "Transatlantic Touch: A Study of Haptic Collaboration over Long Distance", Presence Vol. 13, Issue 3 - Special Issue: Collaborative Virtual Environments, 2004.

[26] J. Cheong, S. Niculescu, A. Annaswamy and M.A. Srinivasan, "Motion Synchronization in Virtual Environments with Shared Haptics and Large Time Delays", Proceedings of the First Joint Eurohaptics Conference and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, pp. 277-282, Pisa, Italy, 2005.

[27] Glencross, M, Pettifer, S and Hubbold, R., "A Network Architecture Supporting Consistent Rich Behavior in Collaborative Interactive Applications", IEEE Transactions on Visualization and Computer Graphics, May-June 2006.

[28] 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.

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

[30] S. Shirmohammadi, N.D. Georganas. An End-to-End Communication Architecture for Collaborative Virtual Environments. Computer Networks, Vol.35, No.2-3, Febr. 2001, pp. 351-367.

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

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

[33] Kyoung Shin Park; Kenyon, R.V. "Effects of network characteristics on human performance in a collaborative virtual environment Virtual Reality", 1999. Proceedings., IEEE, Vol., Iss., 13-17 Mar 1999, pp. 104-111.

[34] Hikichi Kenji et al. 2002. "The Evaluation of Adaptation Control for Haptics Collaboration Over the Internet". In proceedings of the International CQR Workshop 2002, Okinawa Japan, May 2002.

[35] Souayed, R. T., Gaiti, D., Pujolle, G., Yu, W., Gu, Q., and Marshall, A. 2003. Haptic Virtual Environment Performance over IP Networks: A Case Study. In Proceedings of the Seventh IEEE international Symposium on Distributed Simulation and Real-Time Applications (October 23 - 25, 2003). DS-RT. IEEE Computer Society, Washington, DC, 181.

[36] C.Z. Sun and D. Chen. Consistency maintenance in realtime collaborative graphics editing systems. ACM Transaction on Computer-Human Interaction, Vol. 9, No. 1, 2002: 1–41.

[37] Martin Mauve, "Algorithms and Protocols for Distributed Multimedia Systems", Habilitation, University of Mannheim, Mannheim, 6.11.2002

[38] Chen, L., Chen, G., Chen, H., March, J., Benford, S., and Pan. "An HCI method to improve the human performance reduced by local-lag mechanism". Interact. Comput. 19, 2 (Mar. 2007).

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

[40] D. Mills, "Simple network time protocol (SNTP) version 4 for IPv4, IPv6 and OSI," RFC 2030, IETF, 1996.

[41] C. Demichelis, P. Chimento, "Instantaneous Packet Delay Variation Metric for IPPM", Draft IETF, October 2009

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

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

[44] Open Haptics Toolkit: http://www.sensable.com/products-openhaptics-toolkit.htm. Last viewed on June 1, 2007.