REDUCING THE EFFECT OF NETWORK DELAY ON TIGHTLY COUPLED INTERACTION

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the Degree of Master of Science

In the Department of Computer Science

University of Saskatchewan

Saskatoon

By

Dane Stuckel

Keywords: Real-time groupware, CSCW, network delay, coordination theory

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

ABSTRACT

Tightly-coupled interaction is shared work in which each person's actions immediately and continuously influence the actions of others. Tightly-coupled interaction is a hallmark of expert behaviour in face-to-face activity, but becomes extremely difficult to accomplish in distributed groupware. The main cause of this difficulty is network delay – even amounts as small as 100ms – that disrupts people's ability to synchronize their actions with another person. To reduce the effects of delay on tightly-coupled interaction, I introduce a new technique called Feedback-Feedthrough Synchronization (FFS). FFS causes visual feedback from an action to occur at approximately the same time for both the local and the remote person, preventing one person from getting ahead of the other in the coordinated interaction. I tested the effects of FFS on group performance in several delay conditions, and my study showed that FFS substantially improved users' performance: accuracy was significantly improved at all levels of delay, and without noticeable increase in perceived effort or frustration. Techniques like FFS that support the requirements of tightly-coupled interaction provide new means for improving the usability of groupware that operates on real-world networks.

ACKNOWLEDGEMENTS

There are many people that helped me complete this thesis, and I am grateful to everyone. The most gratitude goes to my supervisor, Carl Gutwin, who taught me how to conduct research, how to write, and how to organize my thoughts coherently. To David Pinelle, Tadeusz Stach, and Miguel Nacenta, thank you for encouraging me and offering me advice throughout these last two years. I would also like to thank the members of my graduate committee: Lorin Elias, Ralph Deters, and Ian McQuillian. Thank you to everyone who participated in my many studies and experiments. Finally, I would like to thank my parents for being proud of me. Thank you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
CSCW	Computer Supported Collaborative Work
FFS	Feedback-Feedthrough Synchronization
IDF	Immediately Displayed Feedback
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

CHAPTER 1

INTRODUCTION

Tightly-coupled interaction is collaborative activity in which each person's actions immediately and continuously influence the actions of others. Tightly-coupled interaction is common in real-world group activity: for example, one person handing an object to another must watch the other person's approaching hand and adjust their own hand's movement to match it; similarly, two people playing a fighting game must time their moves to the fraction of a second based on what the other person is doing.

People perform tightly-coupled interaction in activities where they rely on timing, precise movement, and coordination, which is the hallmark of expert performance in groups. For example, dancers move in tandem with their partners, and in sports, players synchronize their movements during play. However, the precision timing and coordinated movement required to perform tightly-coupled activities becomes difficult in real-time distributed groupware. Real-time distributed groupware is software that allows people to work together over a network at the same time. Real-time distributed groupware is often affected by network latency, which makes it difficult for people to coordinate their actions because the information that shows what other people are doing arrives late on remote clients. Although users are able to cope with small and consistent amounts of delay, they become confused when the information they receive is unpredictable or inconsistent. One main problem for tightly coupled interaction is that local events are often displayed immediately, whereas remote events that occur at the same time are displayed only after being sent over the network.

Network delay is unavoidable on the Internet, so users of groupware systems need a way to perform tightly-coupled work in the presence of delay. In this thesis I look specifically at the problem of improving human coordination in the presence of network

delay, and I introduce a technique that can reduce the effect of delay on tightly-coupled interaction.

1.1 Research Problem

The problem addressed in this thesis is that it is difficult to perform tightly-coupled interaction in the presence of network delay. Tightly-coupled interaction occurs when actions by individuals are closely coordinated, and in which each person's actions immediately and continuously influence the actions of others. In tightly-coupled activities, the actions of individual people are based on the feedback from others. The precision timing and movement required to perform tightly-coupled activities is extremely difficult in distributed groupware, largely because of the presence of network delay.

I use the following definitions:

- *Feedback* is the visual information displayed to the local user as a result of their actions – for example, as a person controls an avatar, feedback is the corresponding movement of the avatar on screen;
- *Feedthrough* is the same visual information as feedback, but displayed to the other members of the group, which means that the information must travel across the network before it can be displayed.
- *Immediate Feedback* is a display scheme where feedback from local actions is displayed immediately and feedthrough is displayed as it is received from the network. This means that actions that occur at the same time on remote clients are displayed shortly after local events, which disrupts tightly-coupled work.

Delay makes it difficult for people to coordinate their actions, because the visual feedthrough information that shows what other people are doing arrives late. The group's actions become disorganized because of the immediate feedback display policy that is used by most real-time distributed groupware.

1.2 Solution

I address the problem of network latency's effect on tightly-coupled interaction by re-synchronizing local feedback with remote feedthrough information. This technique is called Feedback-Feedthrough Synchronization (FFS), and it works by matching the display of local feedback with the arrival of information from remote sites by holding back the local delivery for a time period equal to the network delay between the clients. This means that local feedback and remote feedthrough for an action are displayed at approximately the same time on all clients. Thus both local and remote actors can react to that information at the same time, just as they would in a face-to-face environment.

The consequence of holding local user feedback is that the user's controls may seem less responsive, because users also rely on feedback from their own previous actions to inform their actions in the future. A key element of the evaluation was to determine if the local feedback latency caused the task to become more difficult than immediately displayed feedback with similar amounts of network delay.

1.3 Steps in the Solution

To develop FFS, I used an iterative process that first explored the effect of network delay on tightly-coupled interaction, and then developed a technique to improve users' ability to cope with network delay.

1. Definition: I developed a usable definition of tightly coupled interaction, which is collaborative activity in which individual actions are closely coordinated, and in which each person's actions immediately and continuously influence the actions of others. By identifying components and environmental constraints of tightly coupled work from the literature, I reduced Neisser's (1976) cognitive framework to an action-feedback cycle. The action-feedback cycle can be used to describe the details of tightly-coupled work both with and without delay.

2. Exploration of Delay Effects: I explored tightly-coupled activities and determined the effects that delay has on tightly-coupled interaction by using informal usability testing. Network latency was simulated between group members as they performed various tightly-coupled tasks in prototype groupware systems, and I evaluated their performance and the comments they made during these activities. The early

exploratory studies used custom-built systems to explore the domain of tightly-coupled interaction in groupware.

3. *Development of FFS Technique*: Using my evaluation of the effects of visual delay from the previous step, Feedback-Feedthrough Synchronization was designed to reduce the difficulty of users performing tightly-coupled interaction. I developed the idea of delaying the visual feedback of the user's own local actions by the same amount of delay currently on the network.

4. *Prototype Construction:* I built a prototype that implemented FFS in a networked distributed groupware game. In the game, one player drove a fire truck and collected water, while other player controlled the aim of the water turret to put out a fire. This task was designed such that tightly-coupled interaction was required in order for the participants to succeed.

1.4 Evaluation

I tested the effectiveness of FFS in a study where participants performed a tightly-coupled task with increasing amounts of simulated network delay. The results indicate that when FFS was used, people's accuracy improved significantly. When there was no delay, groups were able to perform the task with an average of 95% accuracy. At low levels of delay (100 ms), those using FFS were able to perform the task with 94% accuracy, while those receiving immediate feedback were only able to perform the task with 84%. The effect was more dramatic with larger amounts of delay: At 500 ms, those using FFS were able to perform the task with 56% accuracy, while those receiving immediate feedback were only able to perform the task with 33% accuracy.

Although tasks with more network delay were rated by participants as significantly more difficult, using FFS did not result in a change in perceived effort or performance. These results suggest that from the user's perspective, in this particular task, the difficulty of compensating for delayed local feedback is not significantly different from compensating for network delay. Although participants thought the difficulty of the task was the same regardless of the display scheme, they performed significantly better while using FFS.

1.5 Contributions

The main contribution of this thesis is the introduction of the FFS technique. FFS improves people's ability to work together using real-time distributed groupware. Actions and activities that are difficult to perform with a certain level of delay can now be performed in situations with much higher delay. This allows many types of activities in group work to happen successfully that could not happen before. In addition, even though some of these activities are subtle and small (e.g., passing an object to another person), the ability to support tightly-coupled interaction will help distributed groupware to feel more like face-to-face work. My results show the potential of FFS as a way of dealing with delay in groupware, and as a way of improving the usability of real-time groupware that supports tightly-coupled interaction.

There are also several secondary contributions of this work:

- I confirmed the results of previous studies, which demonstrate the negative effect of network delay on tightly-coupled collaboration.
- I described a new way to model tightly-coupled work: the action-feedback cycle.
- I designed a task that can be used for the future study of FFS.
- I built a groupware networking library, called GTC.

1.6 Outline of Thesis

Chapter Two reviews related work on real-time distributed groupware. It explores the type of communication relevant to tightly-coupled interaction, coordination theory, and the design of real-time distributed systems. Finally, it explains existing strategies for dealing with network delay, and it illustrates the need for a more user-oriented approach.

Chapter Three explains the basic concepts behind FFS. The action-feedback cycle is used to explain the process of tightly-coupled interaction, and then it is used to illustrate why network delay is a problem in real-time distributed groupware. Then I explain FFS, and why it helps users collaborate.

Chapter Four discusses the implementation of the technique, as well as the applications and components I created for my research. This chapter includes a detailed

feature analysis and description of the GTC networking library, the exploratory platform Spacewar Duo, and the Fire Truck application that was used in the formal study.

Chapter Five describes three exploratory studies that contributed to my research. These studies helped to explore the domain of tightly-coupled interaction as well as the effects of network delay.

Chapter Six reports on the design and methodology and results of the formal study. The design and the task are described in detail, along with the procedure and physical apparatus used in conducting the study. I also analyze the users' quantitative performance for statistical significance, present the users' subjective perceptions of effort, and report on user comments during the trial.

Chapter Eight discusses the implications of my work. It examines possible explanations for my results, and explores design questions related to this thesis.

Chapter Nine summarizes the research of this thesis by discussing its contributions, and several possible future directions for further work in the area.

CHAPTER TWO

RELATED WORK

The FFS technique is based on research in four areas of study: Groupware, Coupling, Coordination, and Delay. These topics will be covered in the following sections.

2.1 Groupware

Groupware is software that helps people work together. It is useful for performing collaborative tasks that involve planning, designing, and teamwork. Groupware has been divided using two main concepts: the amount of time it takes to pass information, and the distance over which people are communicating (Figure 2.1).

The first difference among groupware is whether the software supports synchronous or asynchronous communication. *Asynchronous* groupware allows people to communicate over a range of time, such as through email, blogs, or forums. *Synchronous* groupware passes information over a relatively short amount of time, where information is sent and received almost immediately. Synchronous communication is sometimes called "real-time" communication, because the speed of information is almost as fast as communicating face-to-face. Examples of synchronous communication are voice chat, or collaborative virtual environments, or real-time video games.

The second major dimension that determines how groupware is used and built is the distance between the participants, where there is a distinction between co-located and distributed groupware. *Co-located* groupware helps people work together in the same physical location. The focus of co-located groupware is to allow a group of people who are gathered together to collaborate more effectively than they would be able to otherwise. Examples of co-located groupware are single-display, large-display and tabletop systems designed for groups. *Distributed* groupware allows people to work together over a distance. Collaboration becomes difficult when people are separated by a distance, since the standard forms of communication from face-to-face interactions are

unavailable, so people are limited to the types of communication that are provided by the software (Baecker, 1993; Crowley, 1990; Hill, 1992; Patterson, 1991; Dewan, 1991). Examples of distributed groupware are shared white boards (Greenburg, Haynes, and Rada, 1995), collaborative writing tools (Sharples, 1993), or view sharing systems (Greenberg, 1990).

This thesis is concerned with real-time distributed groupware, which is groupware that allows people in different places to synchronously work together (top-right of Figure 2.1). This kind of groupware is used in fast-paced collaboration, such as the rapid manipulation of virtual objects, conversation, and conflict resolution.

	Same Place	**Different Place**
Same Time	**Face to Face Interactions** • Face-to-face conversation • Public computer displays • Electronic meeting rooms • Group decision support systems	**Remote Interactions** • Telephone • Shared view desktop conferencing systems • Desktop conferencing with collaborative editors • Video conferencing • Media spaces
Different Time	**Ongoing Tasks** • Post-it note • Team rooms • Group displays • Shift work groupware • Project management	**Communication and Coordination** • Mailed letter • Vanilla email • Asynchronous conferencing bulletin boards • Structured messaging systems • Workflow management • Version control • Meeting schedulers • Cooperative hypertext & organisational memory

Figure 2.1. Examples of group work (Johansen 1988 in Baecker, 1995, p. 742)

2.2 Coupling

Coupling is a subjective measure of how much interaction is required between people as they work together toward some goal. Olson and Teasley (1996) describe coupling as a unidimensional scale between loose and tight, which is determined by how much interaction is required between the participants for clarification or persuasion, and how immediate a response is needed. Gutwin (1997) describes coupling as the amount of work that one person can do before they need to interact with another. Collaborative group work is described as either loosely or tightly coupled, depending on the task.

Loosely-coupled interaction is often fluid or dynamic, where the group will gather to complete a portion of a task, and then separate moments later to work on individual parts of the overall process (Pinelle, 2003). Olson and Teasley (1996) describe loosely-coupled work as activities where people need to be aware of others' actions, but do not require immediate feedback or clarification. People could perform parts of the work independently and in parallel, becoming more tightly-coupled when there are conflicts between their tasks. Churchill and Wakeford (2001) use the term 'mobility' instead of coupling, but suggest that loose mobility implies that people do not act in synchrony with each other, and that communication is asynchronous or intermittent.

Tightly-coupled interaction is collaborative activity in which individual actions are closely coordinated, and in which each person's actions immediately and continuously influence the actions of others.

> At one extreme, tightly coupled work involves two or more people whose work is directly dependent on each other, and their work typically involves a number of interactions to compete the task. Immediate interaction helps them to communicate clearly or to negotiate some resolution.
>
> (Olson and Teasley, 1996)

There are two main elements of tightly-coupled interaction: the interdependence between their actions, and the feedback and feedthrough between group members.

In general, a group's interdependence is determined by how much group members depend on one another. Coordinating your own actions with the actions of other people

is difficult when there is a high degree of interdependence between group members, because it is more likely that your work will interfere with the work of others. To work in a tightly-coupled fashion, a group needs to understand what is happening in the shared environment and what each person is doing, so that people can avoid undoing, redoing, or damaging the work of someone else.

Feedback and feedthrough are the means by which people determine what they should do next and how they should adjust their actions. Feedback is the visual information displayed to the local user as a result of their actions – for example, as a person controls an avatar, feedback is the corresponding movement of the avatar on screen; feedthrough is the same visual information, but displayed to the other members of the group, which means that when using distributed groupware, the information must travel across the network before it can be displayed.

This thesis is only concerned with tightly-coupled tasks, because network delay mainly interferes with the synchrony of feedthrough information from other clients. Tasks with no feedthrough information will not gain any benefit from FFS.

2.3 Coordination

Malone and Crowston (1990) define coordination as an "act of managing interdependencies between activities performed to achieve a goal." In other words, coordination is a process of organizing your own actions with the actions of others. For example, when passing a ball in sports, two teammates are trying to coordinate their actions. The player that is passing the ball should know where the receiving player will be, if the receiving player is ready or preparing to catch the ball, and if the path is clear between the passer and the receiver. Equally, the receiver should know what direction the ball will be coming, and when the ball will arrive. If the passer or the receiver can communicate these things to the other, then it will be easier to pass the ball between them. When the players are visible to each other, in the same arena or field, then most of this information is available implicitly, because the players can see what their teammates are doing and where they are.

Thompson (1967) identifies three types of coordination: standardization, which is the development of rules and routines to guide work practices; planning and scheduling,

where timing and order of work are specified; and mutual adjustment, where group members must monitor and respond to other group members' activities through ongoing communication. Many coordinated activities make use of all three types of coordination, so aiding the user in any of these three types of coordination will often make coordination easier overall. For example, passing a ball can make use of all three kinds of coordination. Standardization consists of the rules, regulations, and the local team practices that guide player behaviour. Planning and scheduling involves choosing a game plan or a play before the pass occurs. Finally, mutual adjustment takes place in the midst of the pass itself, as the players adjust and refine their actions to compensate for the slight variations in the task resulting from performing the action in a real environment. Of the three types of coordination, this thesis is only concerned with mutual adjustment, because that is the only type that relies on communication between the group members during the task.

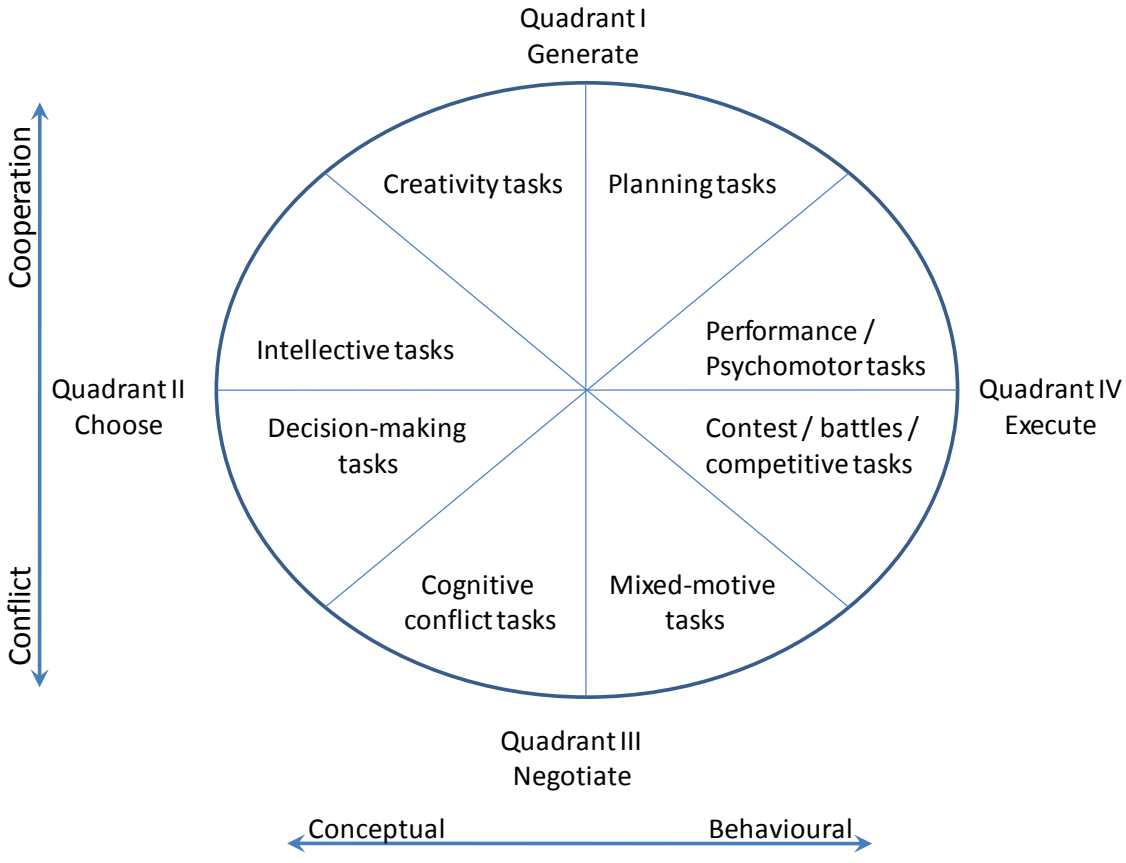


Figure 2.2. McGrath's Circumplex

Coordination has been studied in both the physical and social sciences, and includes a broad variety of work. This thesis is only interested in a small component of the theory of coordination, which is that pertaining to tightly-coupled interaction. The coordination that tightly-coupled interaction requires is more concerned with low-level cognitive functions than of planned, long-term cooperation. McGrath (1991) illustrates the difference between various cooperative tasks by dividing them into eight distinct types, which he arranges as a circumplex (Figure 2.2). Tightly-coupled interaction falls under Quadrant IV of McGrath's circumplex, which includes psychomotor tasks and competitive tasks. The main challenge of tightly-coupled interaction is acting correctly at the proper moment in time, so these tasks are behavioural instead of conceptual, and involve immediate conflict or collaboration. The desired result of this kind of activity is generally already known before the group begins the task, and the measure of success is how well the task was executed.

The other tasks in McGrath's circumplex, which include creative design, planning, and decision-making, do not benefit as much as psychomotor and competitive tasks, since their results are measured differently. For example, a team might gather to design a layout for a magazine cover, where their results are measured by the placement of features, attractiveness to customers, and by various other metrics that relate to the quality of their design. However, these types of tasks still contain small moment-to-moment tightly-coupled interactions that are needed to achieve their overall goal, such as passing pens and paper back and forth, and avoiding interference with another's work.

2.3.1 Coordination Mechanisms

Coordination mechanisms are the means by which people coordinate their actions. For example, people may coordinate their actions through prearranged agreements, such as deciding to meet somewhere at a certain time, or planning some sort of activity together beforehand. People may also use contextual or societal rules to coordinate their actions, such as always driving on the right side of the road, or paying taxes. In tightly-coupled interaction, the most frequently-used coordination mechanism is communication between the participants. There are two main ways in which people communicate during tightly-coupled activities: explicit and implicit communication.

Explicit communication occurs when a participant actively gives information to others in the group, such as by consciously speaking or gesturing during a conversation (Segal, 1994). For example, a person may announce his presence in a room by telling the occupants that he is there. However, explicit communication is too slow for tightly-coupled interaction, because it takes time to transmit the information and to interpret what the other person is trying to say, especially for language-based communication such as speech, sign language, or textual writing.

Implicit information is given passively by a participant's presence or by the results of their actions. For example, the sound of the door opening and someone stepping through may inform others that someone else is entering the room. In the case of tightly coupled activities, much of the information needed to coordinate a group's actions is given implicitly, either because it is too slow to communicate the information explicitly, or the information may be too hard to describe in a purely explicit manner (speech, gesturing, pointing). Most of the information that we use in everyday life is implicit, because we get this information "for free" from simply being aware of our environment, using our five senses (Daft, 1986). Implicit information is often faster than explicit information to relay between group members, because it does not require a conscious sending of information by anyone in the group (Segal, 1994). If someone wishes to know something that is available implicitly in their environment, then they can perceive it directly, without needing to have any sort of conversational exchange with other group members. The chief advantage of implicit information is the speed at which it can be exchanged, and speed is necessary to complete time-dependent activities.

It is possible to send implicit information in an explicit way. If people know that they are being watched, they may act in a way that indicates or even exaggerates their intention or purpose. In this way it is possible to intentionally communicate thoughts or ideas to other people in order to coordinate their actions. For example, imagine a group playing a collaborative multiplayer game, such as a first-person shooter, and that the group is approaching a T-intersection. If someone knows that they are being watched by a teammate (perhaps they are moving in front of the others), that person may move closer to the left or right wall to indicate her future direction of travel.

2.4 Network delay and its effects on collaboration

Delay is common in real-world distributed applications because information must be transmitted across a network and processed at the other end before it can be displayed. There are two main types of delay – latency and jitter. *Latency* is the time that elapses between an event and its display on another system in the group. This results in a person's actions in the shared environment being seen after they actually occur. *Jitter* is the variation in latency due to changing network traffic conditions and processing loads. Jitter causes a remote user's actions (e.g. moving a telepointer) to appear jerky, with the result that they may become difficult to predict.

Delays can have severe effects on collaboration – on coordination, communication, and understanding of the shared situation. Delay can make turn-taking difficult to negotiate (Park and Kenyon, 1999), can hinder social locking protocols, and can cause inconsistencies that lead to confusing roll-back actions (Mauve, Vogel, Hilt, and Effelsberg, 2004). Delay may also cause users to disagree over the timing or simultaneity of key events (Vaghi, Greenhalgh, and Benford, 1999). People may experience different orderings of events with implications for causality, such as missed causal links or wrongly inferred dependencies (Brun, Safaei, and Boustead, 2006).

There has been a large amount of research to determine how network delay affects users of distributed groupware. It is widely accepted that high levels of network delay can lower user performance in real-time activities, and it is also known that users experiencing very low levels of network latency find the delay almost unnoticeable. Several studies use games as the application of interest. For example, in a virtual ping-pong game, users did not seem to perceive less than 150 ms of delay, and the delay did not seem to affect the users in any way (Vaghi, Greenhalgh, and Benford, 1999). With 500 ms of delay, however, users had great difficulty hitting the ball, and there were verbal misunderstandings between the players about the state of the world. In another study of an overhead-view car racing game, 50 ms of delay was considered acceptable, while delay greater than 100 ms was not (Pantel, 2002). In studies of the online real-time strategy game WarCraft III, players discovered it was relatively easy to cope for network delay up to 500 ms, but delay greater than 800 ms degraded the user experience dramatically (Sheldon, 2003). Finally, first-person shooter games are often more

susceptible to network delay. In these games, latencies as low as 100 ms can significantly reduce user performance, and latency greater than 150 ms feels extremely sluggish for players (Beigbeder, 2004). Jitter has also been shown to have significant effects on people's ability to predict others' movement (Gutwin, 2001), but does not always affect users' perceptions in game environments (Quax et al., 2004).

Games are sensitive to delay for many reasons, but one is that games often involve tightly-coupled activities. For example, people often need to carry out tasks such as hitting a moving target, or acting at the same time as another person. Delayed visual information can confuse or disorient users, causing them to make more errors and take more time to complete a task.

2.5 Techniques for Dealing with Network Delay

A number of strategies have been proposed to assist users in dealing with the various problems caused by network delay. These techniques come from research into distributed simulations, databases, networking, and groupware; however, none of the solutions are ideal for the problem of supporting tightly-coupled work in shared workspaces. I look at four strategies below: hiding delay, adding delay, revealing delay, and decoupling users.

2.5.1 Hiding Delay

There are many techniques designed to mask network delay, promoting the illusion of a perfectly synchronized environment. Prediction is one main strategy (Aggarwal et al., 2004). There are several techniques for predicting the actions of other users – essentially making educated guesses about what the other users are doing until an update from the network arrives (Li, Li, and Lau, 2004; Pantel and Wolf, 2002). Prediction models work well in systems where users' actions are limited or regular. If a prediction model is perfect, then it appears that there is no delay at all; however, good prediction is difficult to achieve.

Dead-reckoning (Aggarwal et al, 2004) is a simple prediction model that is often used to cope for missing positional information. The dead-reckoning technique sends the absolute position and velocity of shared objects, along with the current time, to the other

clients. The receiving clients use that information to determine where the objects would be if they continued along the same trajectory. A more complex method of dead-reckoning uses a Kalman filter to predict the object's current location (Azuma and Bishop, 1995).

Local Perception Filters (Smed, 2003) take a different approach: they warp the shared environment to hide the effects of network delay. For example, bullets might appear to slow down as they approach your avatar, but speed up as they near enemy ships. Visual tricks can allow network messages to reach remote clients before the messages need to be displayed on their screens.

Time Warp (Jefferson, 1985) is a method for recovering from delayed information. Clients allow actions to proceed optimistically, but remember the previous states of the shared environment, and execute a rollback if they receive late messages. This method can be optimized for real-time applications by taking periodic snapshots (Mauve, 2004), or by maintaining current and delayed versions of the shared environment (Cronin, 2004).

However, mechanisms like TimeWarp were designed to maintain consistency between data, not between users. Although Time Warp has been recommended for fast-paced action games like first-person shooters (Cronin, 2004), the use of rollback can be distracting and confusing. The users may see the locations of shared objects suddenly change, or discover that the world is not how it was a moment ago. The users experience a gap in the ordering of events because their clients did not show the events leading up to the current state. Users must be able to make sense of what is happening in the application, which requires not just consistent data for the current moment, but also consistent data across a span of time. People rely on the ordering of events and timely feedback to understand the results of their actions and the actions of others.

2.5.2 Adding Delay

Some techniques deal with network delay by adding latency to various points in the distributed system. Adding delay to the receiver side in the form of a buffer is a common method for dealing with jitter (Znati, Simon, and Field, 1995), but there are also ways of using extra delay to address problems of latency.

Bucket Synchronization (Gautier, 1999), *Delta-causality Control* (Ishibashi, 2006), and *Local Lag* (Mauve, 2000) are techniques that ensure everyone in the group experiences the same amount of delay, which achieves fairness and consistency between the clients. All of these techniques include time information in the messages that they send to other clients, either by separating messages into turns or by including actual timestamps. Clients save the messages that they receive, using the time information to replay the messages in the appropriate order. These three techniques add extra delay to all of the clients to achieve fairness and synchronization, and to make everyone as slow as the slowest client.

These solutions may add additional delay, because the cut-off time for late messages must be significantly past the amount of average delay on the network; otherwise, a large portion of received messages would be discarded because of network jitter (See Figure 2.3). For example, if the cut-off time is exactly the same as the average amount of delay on the network, then half of all received messages would be thrown away. However, adding more delay to the already significant amount of delay experienced on the Internet is not an ideal solution for real-time applications like games or telepointers. One solution to avoid increasing the delay, proposed by the authors of Local Lag, is to combine their technique with another such as Time Warp, which would correct the state of the shared environment using the late messages instead of throwing them away.

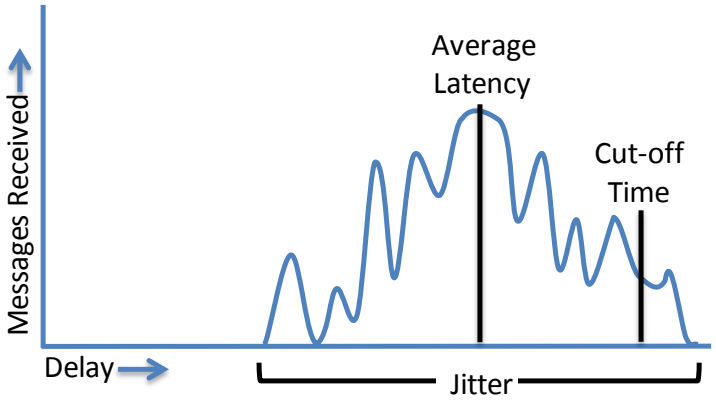


Figure 2.3. The cut-off time for these techniques must be greater than the average message delay to receive more than 50% of the messages.

2.5.3 Revealing Delay

Users have the ability to adapt to a changing environment, and this ability can be used to deal with delay. Revealing the current network conditions gives the user a chance to adapt their behaviour appropriately; there are several methods that have been suggested for revealing delays to the user.

Indicators can display the current amount of delay that is affecting the user. Whether by indicating the overall status of the network, or by revealing the delay between individual clients, it has been shown that users are able to perform better when they are more informed (Gutwin, 2004; Fraser, 2000). *Decorators* can show past states, the current amount of latency or jitter, predicted future states, or any other sort of information to help users work more easily in distributed groupware (Gutwin, 2001).

The Echo technique (Chen et al., 2007) is a user-oriented technique designed to reveal the current network conditions to the local user. They use an additional ghost object to represent the local position of a shared object, which is affected immediately by local actions. The spatial difference between the ghost object and the shared object reveals the amount of delay on the network.

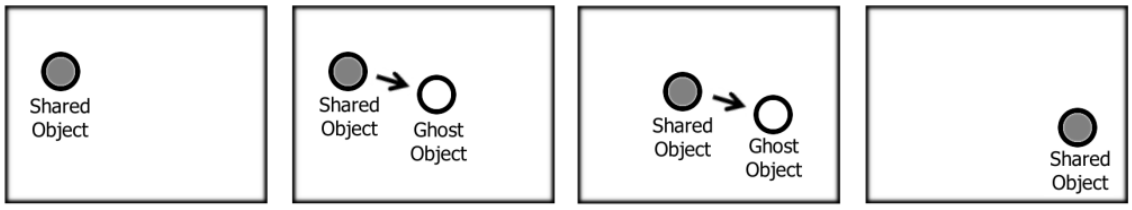


Figure 2.4. The ball is being moved across the screen by the user. In the Echo technique, the ghost object represents immediate feedback while the shared object is updated from the network.

Mechanisms for revealing delay to the user can also be more subtle. Metaphors such as environmental weather can be used to tell users the state of the network, with bad weather indicating unfavourable network conditions (Oliveira, 2003). Users might expect that acting in a rainstorm would be more difficult, and therefore adjust their behaviour accordingly.

2.5.4 Decoupling Users

A final strategy for dealing with delay is to avoid the problem altogether. Limiting the amount of interaction between the users, or the speed at which they interact, may improve the quality of inter-user actions in the presence of delay (Vaghi, 1999). For example, the actions of a slow-moving tank are easier to predict than a fast-moving jeep. Similarly, quick games where users interact with each other instantaneously are impractical to play over the Internet, whereas slower-paced games support online play more easily. For example, Street Fighter III (Capcom, 1999), where players are constantly reacting and counter-reacting to their opponent, would suffer significantly from even small amounts of network delay, whereas fighting games like Dead or Alive 4 (Team Ninja, 2005) that feature slower game play, queued actions, and longer character animations are played online without much difficulty. Commercial games have been studied to determine how much they are affected by network delay, and a large variance has been found between genres (Claypool, 2006), and within the genres (Dick, 2005), but the suggested cause is the degree to which users are able to affect each other.

CHAPTER 3

FEEDBACK-FEEDTHROUGH SYNCHRONIZATION (FFS)

FFS is a solution to the problem of supporting tightly-coupled interaction in the presence of network latency. In the following sections, the action-feedback cycle will be used to explain the theory behind the technique, and then questions about its applicability in real-world groupware will be discussed.

3.1 Action-Feedback Cycle

The action-feedback cycle can be used to describe the process of working in tightly-coupled interaction. Tight coupling is a process in which participants (both local and remote) use the information produced by each person's actions as input to their future decisions. For the local person, this information is feedback; for the remote person, it is feedthrough. Feedback and feedthrough, and the times at which they are displayed, are important because people rely on temporal information to inform them about what other people are doing. Figure 3.1 (left) shows a diagram of the action-feedback cycle for a single user, and (right) unwraps the circular diagram to show individual actions and their feedback over time. The general process of tightly-coupled interaction can then be represented as in Figure 3.2: the figure adds an arrow to show the feedback information also moving to the remote person as feedthrough (note that there is no network delay shown in this example).

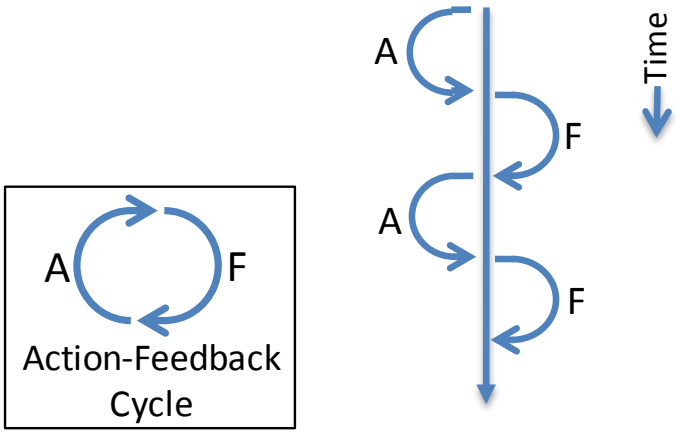


Figure 3.1. The action-feedback cycle (left) and the cycle unwound over time (right).

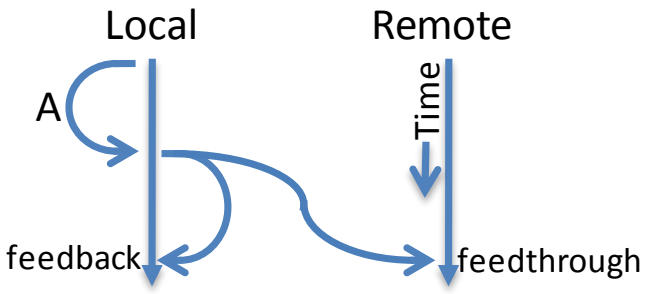


Figure 3.2. The action-feedback cycle between two clients

This representation can show how network delay affects tightly-coupled interaction (Figure 3.3, left). The diagram shows several iterations of the action-feedback loop for the local user, and shows that the feedthrough from the first action arrives late at the remote site. At time T_1 in the diagram (the dashed line), it can be seen that the local user has carried out a second action by the time that the remote user sees the results of their first action – therefore, they now have different views of the state of the world. This is the result of an *immediate feedback policy*, where feedback from the local user is displayed immediately while remote feedthrough information is delayed by network latency.

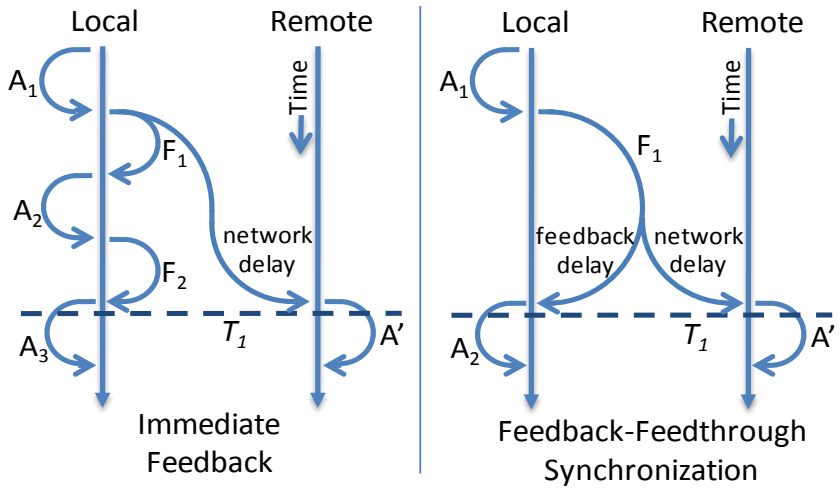


Figure 3.3. The effects of delay when using immediate feedback (left) and when using FFS (right)

3.2 The FFS Technique

Unlike the immediate feedback policy, FFS does not immediately display feedback to the local user. Instead, it displays local feedback at the same time that the feedthrough is displayed at the remote site. The timing is accomplished by monitoring the current network latency, and so will not be exact; but the idea of FFS is to display both the local feedback from a particular action, and that action's remote feedthrough, at approximately the same time. This is shown in Figure 3.4 at right. Delaying the local user's feedback makes an important difference: since the local user needs the feedback to determine their next action, they do not move ahead until the remote user also sees the visual result of their last action. This means that both the local and remote users start their next actions at the same time, and based on the same view of the state of the world. It is the timing of the two people's actions that is important in successful coordination, and FFS prevents either of them from getting ahead of the other.

FFS is a variant of the Local-Lag technique (Mauve, 2004) that is designed to synchronize users in tightly-coupled interaction, instead of maintaining data consistency between clients. Unlike Local Lag, FFS synchronizes users by only delaying local feedback, without affecting network traffic or other clients.

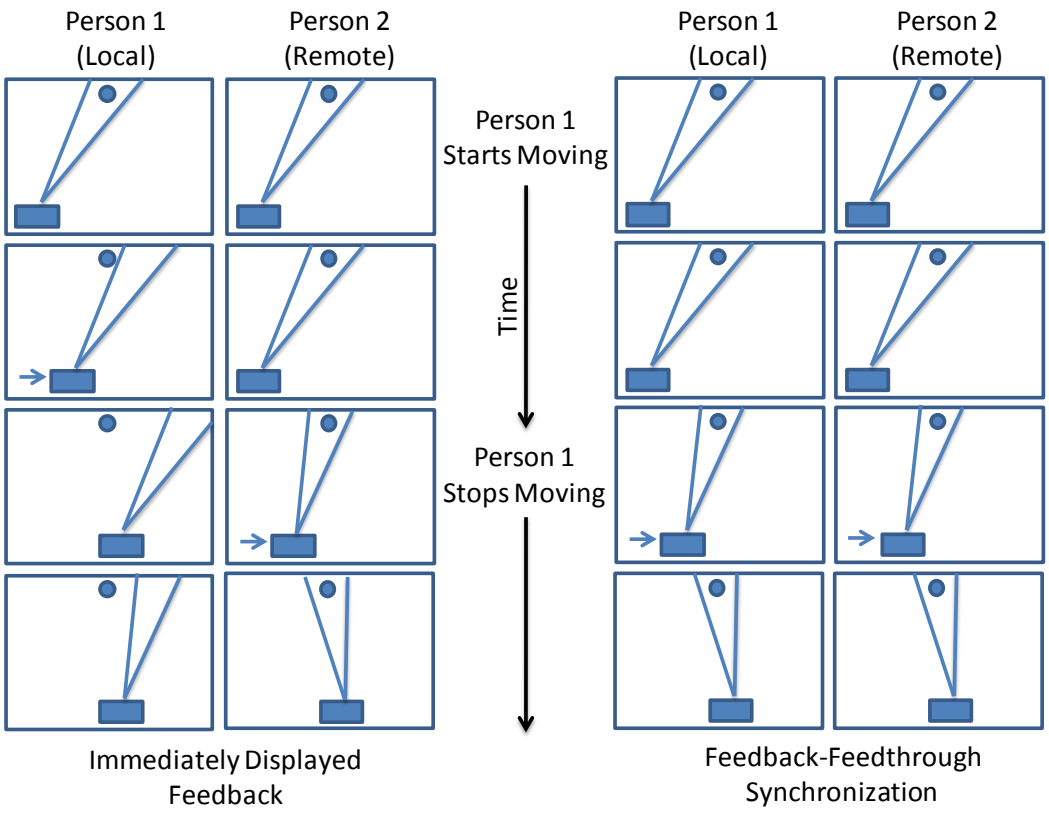


Figure 3.4. Network delay causes actions to be displayed at a later time on remote clients. Adding local feedback delay causes actions to be displayed at approximately the same time on both the local and remote machines.

Figure 3.4 shows an example of the difference between immediate feedback and FFS, using a very simple game scenario. In the game, the local user moves a fire truck (the rectangle at the bottom of each screen) left and right; the remote user moves the direction of the water spray towards a fire (the circle at the top of each screen). The left side of the figure shows what happens with immediate feedback. The local user starts moving to the right in frame 1, and immediately sees their own feedback (frame 2). However, this feedback has not yet reached the remote user, and so they do not adjust the water angle. Assuming that the local user's machine determines the score, the players are now losing valuable points – but the remote user has no idea that anything is wrong. This discrepancy does not happen when FFS is applied (right side of Figure 3.4). When the local user starts moving, the local machine holds the feedback until it also arrives at the remote site. This means that both people see the truck start moving at the same time (in

frame 3), and means that the remote user's actions (based on their view of the world) remain correct on both machines.

FFS works by changing the location, in the process of coordinated action, where delay must be dealt with by the users. Whereas immediate feedback requires that users deal with delay in the coordination of actions, FFS moves the requirement to the local controller – that is, the user is compensating for less-responsive controls, rather than attempting to imagine what is happening on the other person's screen. The benefit is that users cope for the invisible network delay by coping for the visible feedback delay. Since users are able to interact with and experience the different type of delay with less difficulty, they can perceive the results of their actions as others on the network would see them, and they develop a better understanding of how to adjust their behaviour to compensate.

In addition, FFS ensures that both users' screens show (at least approximately) the same view of the world, which aids in communication about the task. One of the main problems with divergent views is that people have difficulty discussing the world (Gutwin and Greenberg, 1998), and FFS allows people to be much more confident that the other person will understand what they say and their references to objects in the world.

3.3 Local Effects for Users

Although FFS improves group performance in tightly-coupled tasks, it introduces latency between when an action is performed and when the user perceives its local feedback, which causes users to subtly change their behaviour. Jay, Glencross, and Hubbold (2007) use the impact-perceive-adapt model to predict user behaviour based on the amount of latency between users' actions and their feedback:

- Impact threshold [25 ms]: Performance errors increase significantly, but users are unaware of feedback latency.
- Perception threshold [50 ms]: Users become aware of feedback latency, but they do not adapt; Performance errors continue to rise.
- Adaptation threshold [100 ms]: Users adapt to feedback latency by slowing down, and errors are reduced.

The adaptation threshold is the upper limit for the perception of immediate causality, which is the point at which users first begin to feel they need to adapt their behaviour because of the feedback latency (Card, Moran, and Newall, 1983). At latencies greater than the adaptation threshold, there is a linear relationship between user speed and feedback latency, which is that users slow down their actions to compensate for the amount of delay between their actions and their feedback (MacKenzie and Ware, 1993).

Users adjust their behaviour differently depending on whether they are using continuous or discrete control mechanism. For example, mouse movements can be adjusted over a continuous range of values, whereas a keyboard is limited to a discrete set of keys. A mouse can be moved forward at various speeds to adapt to the user's current situation, but pressing the up key on the keyboard is a distinct, singular command, so it cannot be adjusted in the same way. Instead of adjusting the direction and speed of their input, keyboard users adjust their behaviour by changing the timing of their actions. The main effect is that discrete controls are more predictable than continuous ones, and therefore less susceptible to delay.

Although FFS allows the users to perform significantly better at tightly-coupled interaction, the application may be seen as lower quality because of the perceived reduced responsiveness of the controls. The feedback latency that FFS introduces to users' actions is similar to the feedback latency in early groupware systems, where actions were sent to a centralized server and the server responded with a single view of the shared environment for everyone (Garfinkel, 1994). The visual latency between users' input and feedback on such systems made interactions with the shared applications frustratingly slow and difficult to use. However, the network latency between users only disrupts actions that are tightly-coupled, and tightly-coupled actions are only a small subset of what a user may do in an application. In order to avoid recreating the sluggish interaction found in centralized, shared view systems, FFS should only be applied to tightly-coupled activities where group performance and timing are valued more than instantaneous feedback. Methods of switching the technique on and off are discussed in section 3.6.

3.4 Designing for Individuals or Groups

Designing groupware is difficult because the requirements of individual work often contradict the requirements of working in a group. In the physical world, the trade-off between designing for the individual or the group is determined by an environment where people can interact with each other directly. The same kind of direct interaction is unavailable to users of distributed groupware, so they have to rely on the forms of interaction that the software designer decided to provide.

However, providing solutions for the group can detract from individual work. For example, single-user applications tend to support symbolic or indirect input, but such versatile interaction techniques also leave very little evidence about who performed the action, its result, or that it has occurred. Conversely, limiting the user's actions or providing too much information about others' activities can distract or interrupt users who are trying to focus on their task (Gutwin and Greenburg, 1998). A balance must be established between what is needed by individual users and what is required by the group.

3.5 Comparison to Other Techniques

Groupware networking techniques are often defined either by how the technique corrects errors caused by network delay or by where data is buffered. FFS differs from other techniques because it only buffers data for the local client. The data that is transmitted across the network is not affected at all, because clients only delay their own local feedback. This is different from techniques like Timewarp, which records all actions by every client such that it can recalculate the state of the shared environment after remote messages are received. FFS only buffers local actions, because it only modifies the local client. FFS is also different from Bucket, Delta-causality, and Local Lag techniques which synchronize everybody to match the latency of the slowest client, in order to achieve fairness and consistency. FFS is intended to synchronize the users, not their clients, so only tightly-coupled interaction needs to be synchronized.

The closest pre-existing technique to FFS is the Echo technique by Chen et al. (2007), but their task is too different from the one chosen in this thesis to objectively compare results (for reasons discussed in Section 8.4). However, FFS demonstrates a

similar improvement in performance, without additional ghost objects and without significantly raising the users' perceived difficulty of the task.

3.6 Transitioning between Immediate Feedback and FFS

Although there is a clear performance benefit to using FFS in tightly-coupled activity, it is important to minimize the burden that it puts on the user. Therefore, correctly identifying tightly-coupled actions and isolating them from actions that are independent can improve the user experience. The difference between tightly-coupled and independent actions is that actions that are tightly-coupled depend on the timing of actions by remote users, so the best way to determine if an action is tightly-coupled is to determine the potential for conflict between the users of the system. If an action could interfere with the results of an action by another user, then network delay may cause a conflict between what the users intended and their results.

An action's potential for conflict depends on the rules of the shared environment, and are therefore highly application specific. Ideally, software designers determine an action's potential for conflict as they are designing their application, but there may be ways to automate the process. For example, if there is collision detection between avatars, the potential for conflict may depend on the distance between them in the environment. However, FFS should be applied to actions in a consistent and predictable manner. Users are good at interacting with predictable systems, but randomly or unexpectedly changing from immediate feedback to FFS or vice versa will not provide much benefit to the users.

3.7 FFS for Groups Larger than Two

The simplest implementation of FFS reveals the network latency between the local host and a single remote client, but when there is more than one client involved, it is not clear how much FFS should delay local user feedback. If network latency varies widely within a group, then it is impossible to synchronize local feedback delay with all of the clients at the same time. In groups with many people, each client needs to determine some value to delay local feedback that preserves the performance benefit of FFS.

There are several ways that FFS could be applied between multiple people. A good minimum effort strategy is to use the average latency of all the clients involved in the tightly-coupled interaction as the delay used for FFS. A better strategy might be to separate users into small tightly-coupled groups, and then only apply FFS between members of their group in a way that is consistent with the rules of the shared environment. It is important to make the grouping obvious such that the results of their actions are consistent, because otherwise the benefit from the effect will be lost, and the users will become frustrated.

Fortunately, the vast majority of tightly-coupled interactions involve only two people. Although people often participate in collaborative activities while in large social groups, tightly-coupled work between group members is usually limited to one-to-one interactions. For example, papers are passed from hand to hand, and dancers move in tandem with their partner. In tight hallways, people pass by each other one at a time, moving in and out of tightly coupled interaction with a series of individuals. If FFS is only applied between users who are affecting each other in a tightly-coupled manner, then there may not be a need to scale FFS to handle groups larger than two people.

CHAPTER FOUR

IMPLEMENTATION

Several components and applications were developed in support of this thesis. Each study was conducted on a custom-built system designed to test my hypotheses. Spacewar Duo, the Ball-Bouncing game, and the Fire Truck game were all designed to study the problem of multiperson control. GTC is a toolkit for developing distributed groupware, and was used to build the prototypes, as well as numerous distributed groupware projects unrelated to this thesis. In addition, certain components of these systems contain important design choices for future work in testing user performance in real-time distributed groupware, such as how artificial network delay was applied, as well as details about my implementation of FFS.

4.1 GTC

The Groupware Toolkit was a networking library built to alleviate the difficulty of creating groupware. It is a thin wrapper for the networking classes of C#, called GTC. Features that I implemented were:

Multiple server connections

GTC supports many types of network architectures. Since servers can support multiple clients and clients can connect to multiple servers, GTC allows centralized as well as peer-to-peer and service-based network architectures (Figure 4.1).

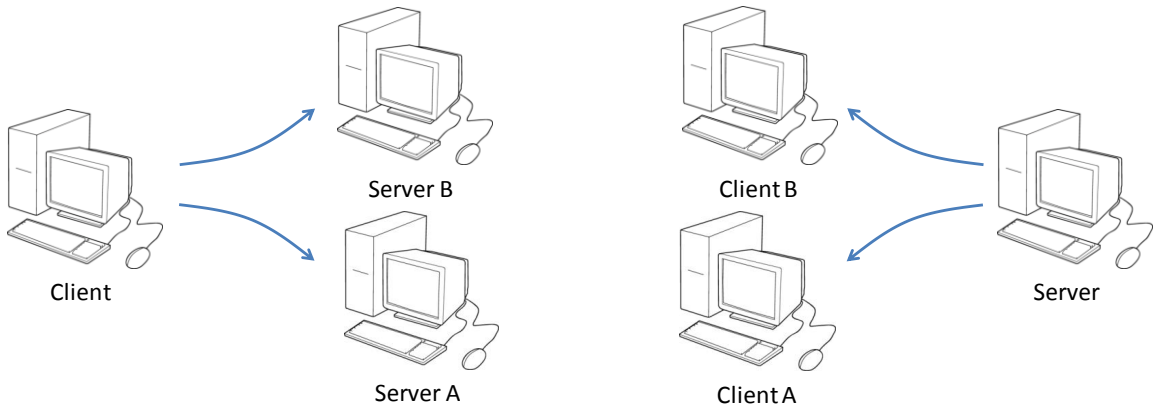


Figure 4.1. Each client can support multiple servers

Streams

Distributed groupware applications relay various types of information between their clients. For example, Avatar movements, chat messages, voice data, status messages, and telepointer positions need to be sorted and distributed to different parts of the application, but the variety of the information and the effort to maintain and debug the source code is troublesome for the application programmer. GT uses the metaphor of streams to separate different kinds of information for the various parts of the groupware program. The stream metaphor allows parts of an application to send and receive messages on their own channel. For example, there could be an avatar movement stream, a chat message stream, a voice data stream, and each message from the network is passed directly to the part of the application that needs it.

Shared Dictionaries

A keyed dictionary is a data structure that allows data to be indexed with a key. Just as numbers are used to reference entries in an array, objects or strings can be used to reference data in a keyed dictionary. In groupware, keyed dictionaries are useful for storing properties of objects, or for retrieving an object by its name. GT included a type of keyed dictionary called a shared dictionary. A shared dictionary pretends that its contents are the same on every client, but in truth, it is a networking construct that distributes changes to the dictionary between the clients.

Streaming Tuples

Streaming tuples are used to share any grouping of primitive data types between clients, such as telepointer or avatar locations. The tuples are designed to be used like regular primitive types, but share their values with the other clients. This feature includes rate control, such that data is only sent to the other clients if its values have been changed, and if a set amount of time has passed since the last time its values were sent.

Variable reliability

Clients can dynamically switch between sending via TCP or UDP. The reliability of messages can be determined on a per-message level by specifying which protocol should be used to send it. TCP is a slow but reliable connection where data that is lost on the network is resent. UDP is a fast but unreliable connection because it does not resend lost data, but it is useful for real-time applications such as games or streaming media.

GTC normally maintains a TCP connection between each client and server. If an application begins to send messages via UDP as well, then a UDP connection is established alongside the TCP connection automatically. Regardless of the protocol that is used for a particular message, they are still sorted by streams, so components of an application can switch between TCP and UDP without changing anything else.

High Precision Timestamps

Standard timers, such as System.TickCount, System.Timers, and System.Windows.Form.Timer, do not have high enough resolution for FFS, 3D animation, networking timestamps, or data logging. I created a high precision, high resolution timer class that receives timestamps from Kernel32.dll, which is a low-level Windows driver.

Aggregation and Rate Control

GTC supports message aggregation. A message can either be sent immediately, or it can be attached to the next message that will be sent on the network. It also supports rate control to limit the number of messages that are sent over a period of time. Rate control uses aggregation to control when messages are sent, and allows messages to be sent more efficiently, because more than one message can be included in a network packet. Message aggregation and rate control in GTC also support message priorities,

such that messages from a certain stream can be sent before messages from another. However, messages can only be aggregated and controlled on a per-message level, so byte-level rate control is not automatically supported.

Limited session management

GTC generates an event on the server when clients connect or disconnect from it. Session events can be passed between clients and servers voluntarily via streams, just like objects, strings or byte arrays. Servers have complete control over session management, and can choose to send and receive session events in any manner they wish. Session events are handled like the other types of streams in GTC (objects, strings, byte arrays), and the methods for sending and receiving session events are consistent with those other data types as well.

Stability

A GTC server will never quit. It will drop clients that cause errors or exceptions, and in the worst case, a server will completely restart itself. Instead of failing and exiting on an application error, a server will throw an error event, and then will recover and continue on as normal. An application can subscribe to error events, and thus receive a report of all of the problems occurring in the server during runtime. One of the methods used to recover from an error is to drop the offending client, but clients are able to reconnect to servers with a single method call.

Network Latency Auto-Detection

The library automatically determines the current level of network latency between each client and server. After a certain interval (the default is twenty seconds), a message containing a timestamp is sent to another client or server, and then it is sent back. The difference between the timestamp in the returning message and the current time is the turnaround time for information between the endpoints. The turnaround time divided in half is a measure of the network latency experienced by that message. Averaging the last several such measurements will determine the approximate average latency currently on the network. GTC calculates this automatically so that applications may know the latency between any client and server.

4.2 FFS Implementation

The goal of the technique is to hold local feedback to mimic what would have happened if the information had travelled over the network. FFS only changes the local user's view in relation to his or her own input, so information travelling to remote clients is sent normally, and information from remote clients is displayed normally as well.

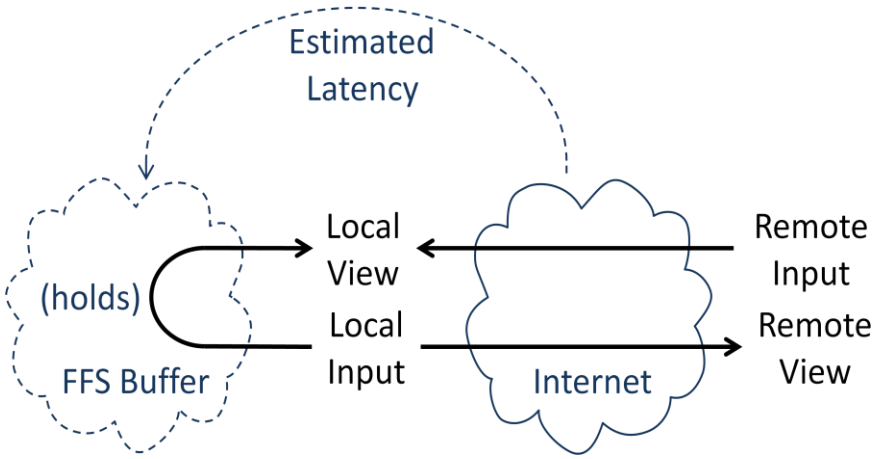


Figure 4.2. Diagram of a simple implementation of Feedback-Feedthrough Synchronization

A simple implementation of FFS relies on two components: one to determine the current latency on the network, and another to buffer local feedback (Figure 4.2).

The current latency on the network can be determined by 'pinging' the other client, which is a process of measuring the amount of time it takes to send and receive messages from the other person.

Buffering local feedback is accomplished by saving the feedback in a dynamic list until it is time to give it back to the user. Whenever a client sends a user's action to other clients over the network, it also saves the action and the current time in a buffer. When the timestamp of any of the actions in the list is less than the current network latency subtracted from the current time, then that action is added to the local view.

4.3 Artificial Delay

As mentioned in chapter three, there are two types of network delay: latency and jitter. I modified my applications to inject latency between the network layer and the rest of the software, but I deliberately chose not to inject jitter as well. Artificially adding

jitter to an application in a way that accurately models real-world jitter on the Internet is difficult. The behaviour of jitter on a network is dependent on many factors, and the different jitter conditions would add another independent variable to the study. Beyond that reason, most jitter can be eliminated in exchange for latency by buffering network traffic on the receiver, and secondly, users do not notice small amounts of jitter, but are heavily affected by latency.

I injected latency into my applications by buffering the data that was received by the clients. A certain amount of network latency was achieved by determining the current network latency, and then buffering the received messages such that the total amount of delay between the clients was equal to the desired amount of delay.

Buffering was achieved using a list structure, where list items were sorted by the time they were added to the list. When the oldest item in the list was older than the amount of time the application wished to buffer, then that item was removed and given back to the application.

4.4 Prototypes

There were three prototype applications: Spacewar Duo, the Ball-Bouncing game, and the Fire Truck game. These prototypes helped explore the area of multiperson control, tightly-coupled interaction, and tightly-coupled interaction with network delay.

4.4.1 Spacewar Duo

Spacewar Duo was a framework for studying the limits of multi-person control. It was a modified version of the 1962 classic Spacewar, made to run with multiple players on distributed clients. It was built in Java, and used basic AWT/Swing and Java2D components.

In Spacewar Duo, a simple triangle drifted around a static space on the screen. A single user could control the triangle, which represented a spaceship, either by applying thrust or by rotating its orientation, and they could fire bullets in the direction of the ship's orientation. There is no little friction in Spacewar's environment, so when the user applied thrust to accelerate, the fastest way to slow down was to turn around and apply

thrust in the opposite direction. It used the GT library (built in Java) to connect clients together, and allowed multiple ships from distributed clients to exist in the same space.

To study multi-person control, I gave users the ability allow two players to control a single ship. When players began the game, they were prompted to choose between being a pilot or a gunner. Those who chose to be a pilot were given a ship, and they could fly around and shoot as normal. Those who chose to be a gunner were randomly assigned to a ship with a pilot. Ships with gunners gained a visible turret and the pilot of the ship lost the ability to shoot, but they had the advantage of being able to rotate their turret to fire in any direction they wished.

The interaction between the pilot and the gunner was made to be tightly-coupled, such that each player heavily affected their partner with their actions. When the pilot moved or turned the ship, the turret was moved or rotated along with it. When the gunner rotated the turret, the ship was turned a quarter of the rotation in the opposite direction, and when the gunner fired, the recoil threw the ship off-course.

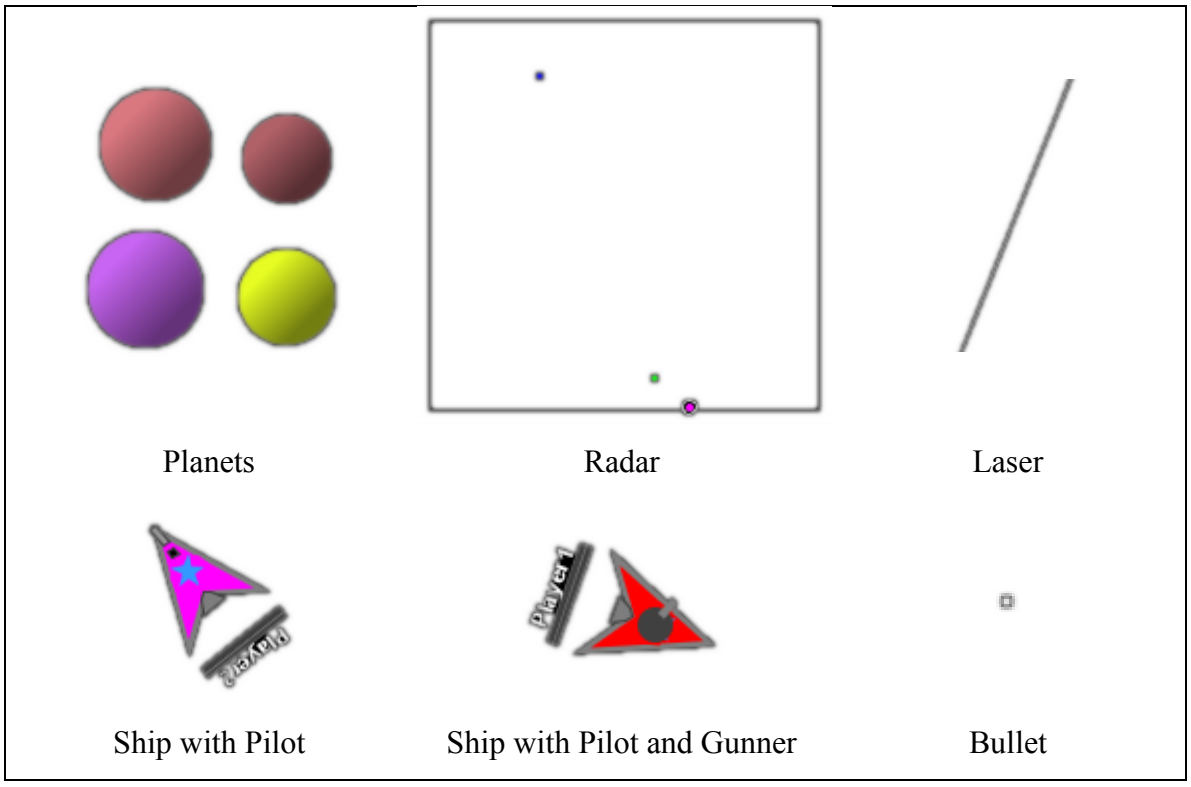


Figure 4.3. Graphics used in Spacewar Duo (black background is inverted).

There was a race course version of Spacewar Duo as well, in which two players travelled through a maze and shot stationary circular targets. The correct direction of travel was indicated by flashing red arrows, and the walls were represented by large yellow blocks. When a ship touched a wall block, their ship slowed to 25% of its normal speed. The turret shot lasers, which crossed the screen immediately instead of moving with momentum as bullets did. When a target was successfully hit by a bullet, the circles turned pink. When all the targets were hit, then the players had completed the maze and the trial was over.

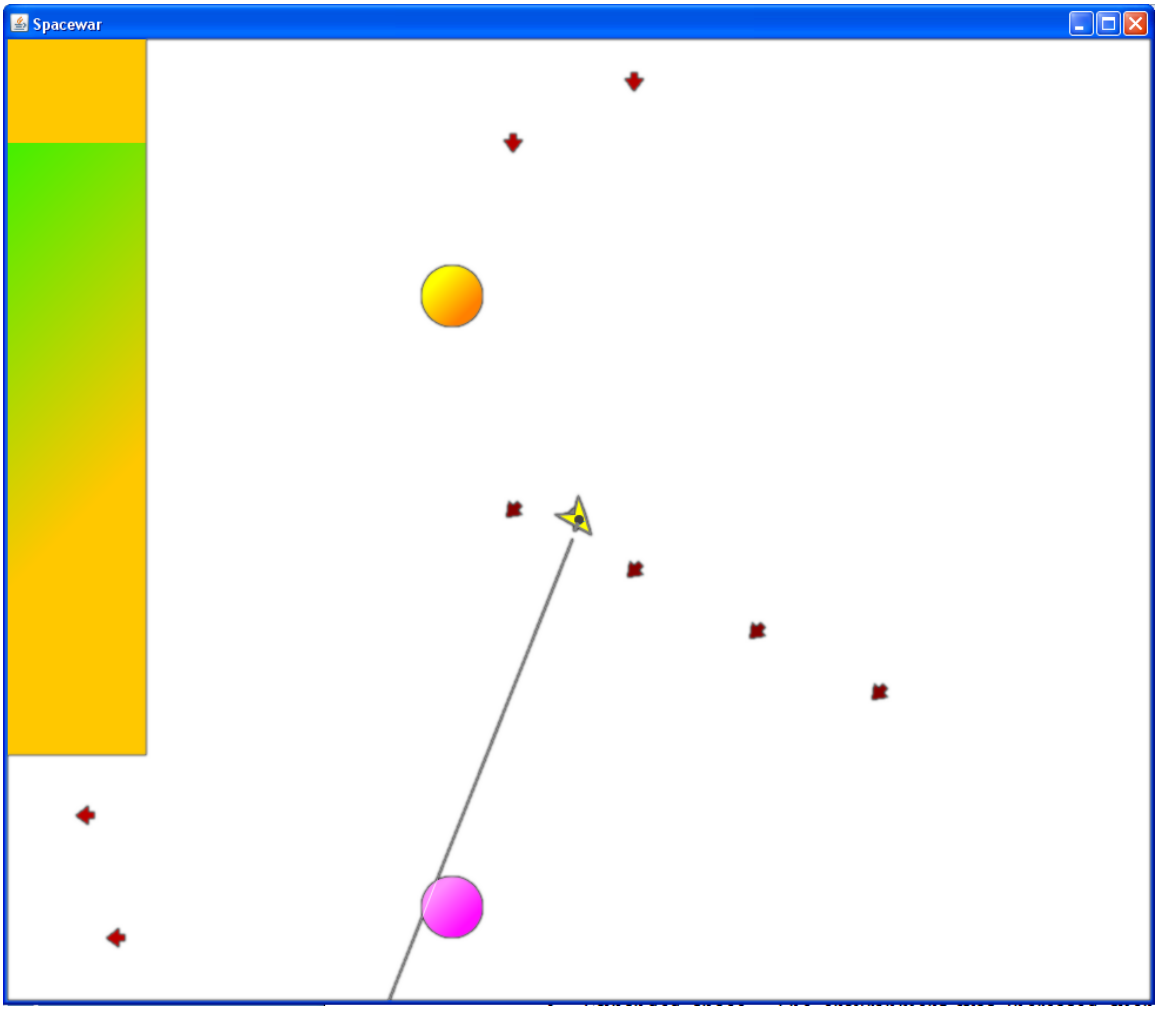


Figure 4.4. Screenshot of Spacewar Duo Race (black background is inverted)

4.4.2 Ball-Bouncing Game

A third multiplayer game was prototyped in Macromedia's Flash. It was a shared-screen two-player game that ran on a single computer. In this two player game, each player would bounce a ball over to the other player's side of the screen while simultaneously trying to catch the ball being bounced toward them (Figure 5.3). Players controlled the buckets with the keyboard. The balls were opposite colours (red and blue), which matched the colours of the buckets that were supposed to catch them. The players used the arrow keys and the W, A, S, D keys to control the red and blue buckets, respectively. Their shared score was displayed at the top center of the screen in large text. Players received 20 points for catching a ball, but they lost 20 points if they missed one.

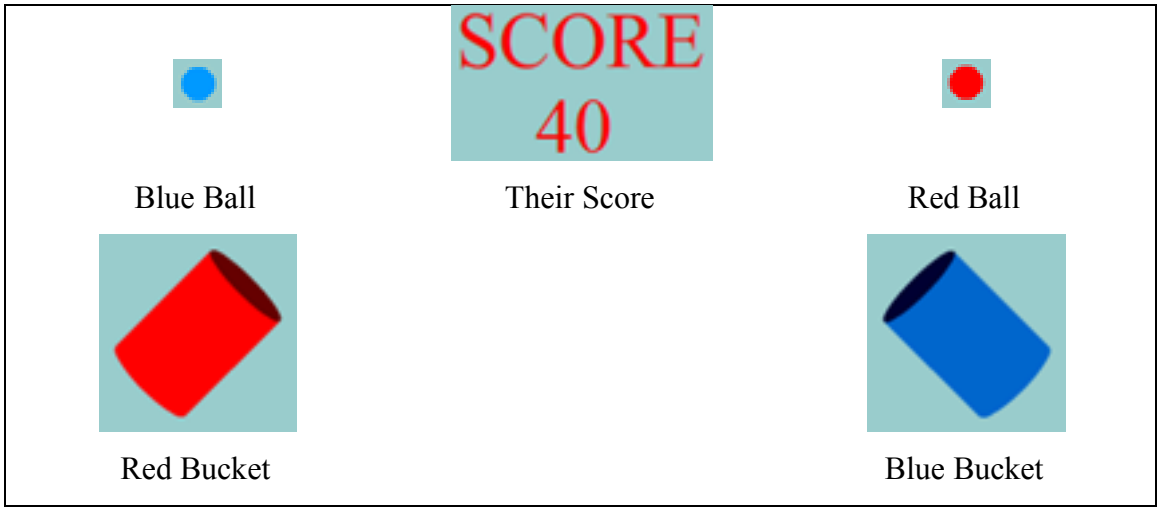


Figure 4.5. Graphics used in Ball-Bouncing Game.

4.4.3 Fire Truck

The Fire Truck game was the application that was used as a prototype for FFS and for the formal study. It was written using DirectX and C#, the networking between the clients used the GTC library, and the graphics were made in MS Paint.

A session in the Fire Truck game only supported two clients, which connected directly to each other to create minimal network delay. The messages that were sent between the clients consisted of both streaming updates and discrete events. The fire truck's position, the current angle of the water stream, and the status of the fire were

streamed between the clients. The current level of artificial delay and the command to begin the trial were sent as discrete messages at the appropriate times in the study.

The goal of the game was to keep the fire out for as long as possible. The fire truck collected water by moving in front of a water tank. During the trial, water decreased at a rate of 250 units per second, and collecting a water tank increased the fire truck's water supply by 600. The five potential positions for the water tanks were dispersed evenly across the screen, with the furthest tanks being 400 pixels from the

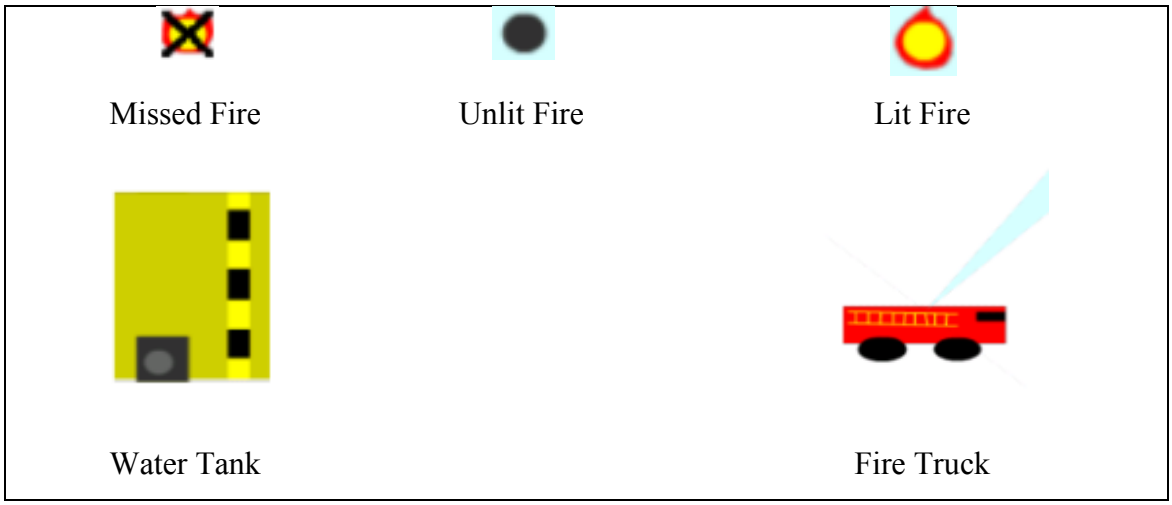


Figure 4.6. Graphics used in Fire Truck game

center of the screen. The order of appearance for the water tanks was randomized, but each trial used the same seed for the random number generator. The fire was 500 pixels above the truck, placed near the top of the screen, and it had three possible states: unlit, lit, and missed. The fire would be unlit as long as the fire truck contained water and the water stream was accurately aimed at the fire. However, the fire would become animated and lit if the fire truck did not contain any water, and a small 'X' would appear on the fire whenever the stream was missing the target (Figure 4.6).

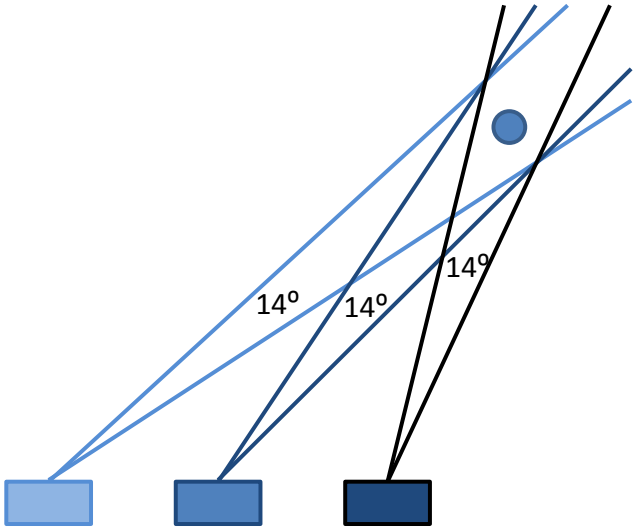


Figure 4.7. Participants had to aim within seven degrees of the target.

The target was considered hit when the current angle of the water stream and the current angle between the fire truck and the fire were within seven degrees of each other. Therefore, the difficulty of hitting the target was approximately the same from any point along the path of the fire truck's travel.

Artificial delay was injected between the clients in order to simulate network latency. The desired level of network delay was achieved by adding artificial delay to the real network latency already present on the network, as detailed in section 4.3. The real network latency between the clients was determined automatically by GTC, and artificial delay was injected in the same manner as in Spacewar Duo. The desired level of delay could be increased or decreased at run-time via keyboard commands. During the study, the level of artificial delay in each trial was determined automatically according to a preset series.

Data logging was done in a separate thread from the rest of the application. The main application passed text data to the logging thread, and when the thread had received more than 5120 bytes, it wrote the data to the hard disk. When the application exited, the logging thread would write the remaining data to the log file.

The clients accessed two different log files over the course of the application's lifetime. The fire truck's position on the screen, the angle of the water stream, the angle of the fire truck to the fire, the amount of water currently in the truck, and the total amount of time elapsed since the program started were all recorded into the first log file.

The second log file contained the results of each trial, which were formatted to be imported by Microsoft Excel or analyzed by |-stat (http://oldwww.acm.org/perlman/stat/). The results of each trial consisted of the users' total accuracy for each level of delay, as well as their accuracy during each ten second interval of the trial.

There were two display policies implemented in the Fire Truck game: immediate feedback, and FFS. When using the immediate feedback policy, the clients display the feedback from the local users' actions immediately (diagrammed in Figures 4.8 and 4.10). Client A is controlling the position of the fire truck with the keyboard, and Client B is controlling the angle of the water with the mouse. If Client A changes the position of the fire truck, then the new position of the fire truck is sent to the other client and the graphic of the fire truck is immediately moved on Client A's screen. Latency causes the position information to arrive at Client B a short time later, which means that the two clients change the positions of their truck graphics at different times.

When using FFS, local feedback was buffered for a short time before it was displayed (Figures 4.9 and 4.11). Client A is controlling the position of the fire truck with the keyboard, and Client B is controlling the angle of the water with the x-axis of the mouse. If Client A changes the position of the fire truck, then the new position of the truck is sent to the FFS buffer and to the other client. The FFS buffer holds the position information for the same amount of time that it would take for the information to reach Client B over the network. After that amount of time has passed, both Client A and Client B changes the position of the local truck graphic to match the input from the user. Input from Client B is buffered in the same manner.

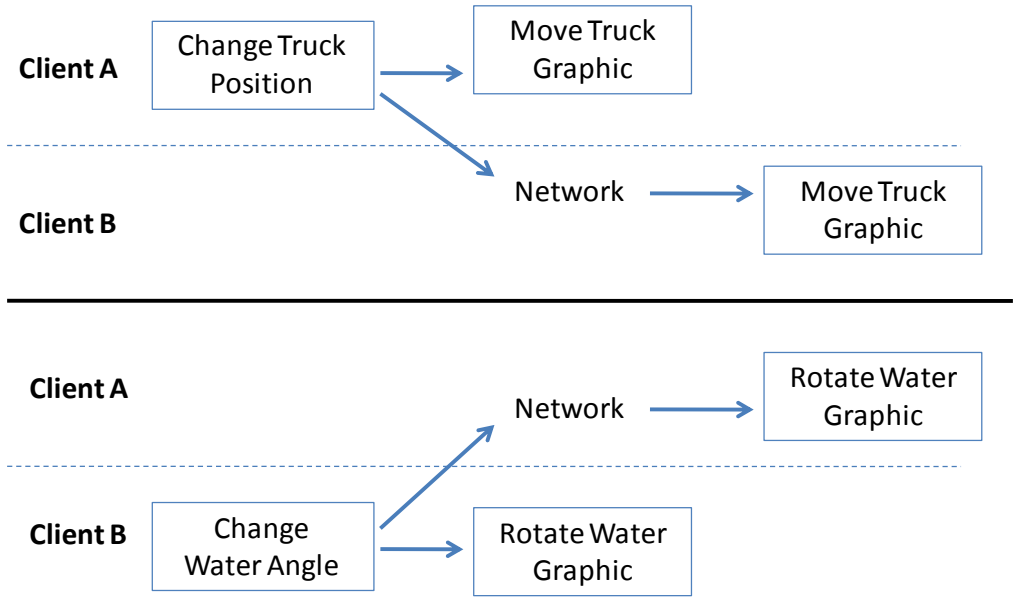


Figure 4.8. The flow of information while using an immediate feedback policy

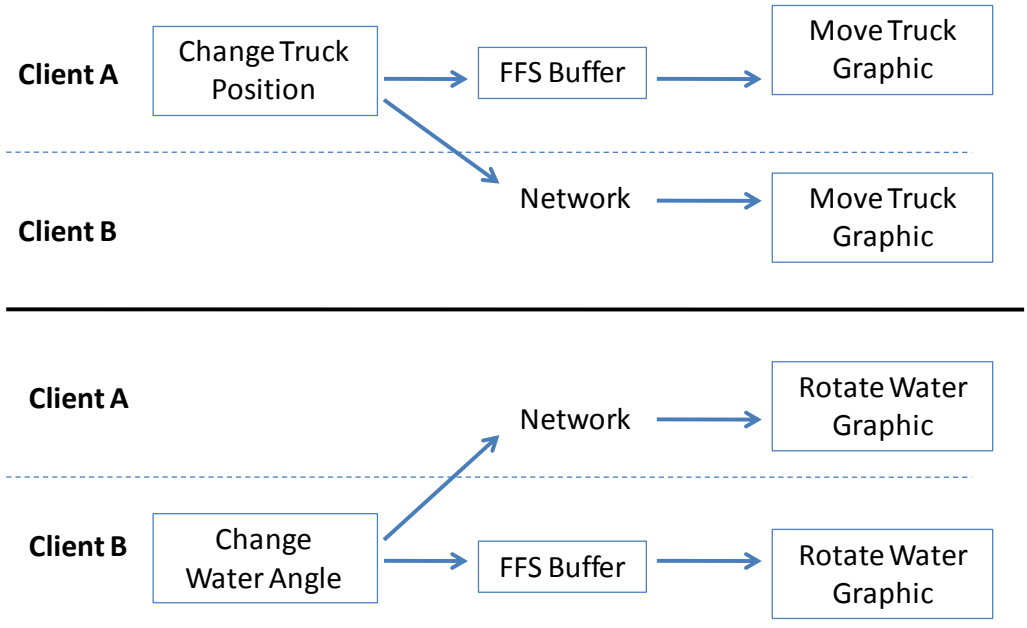


Figure 4.9. The flow of information while using FFS

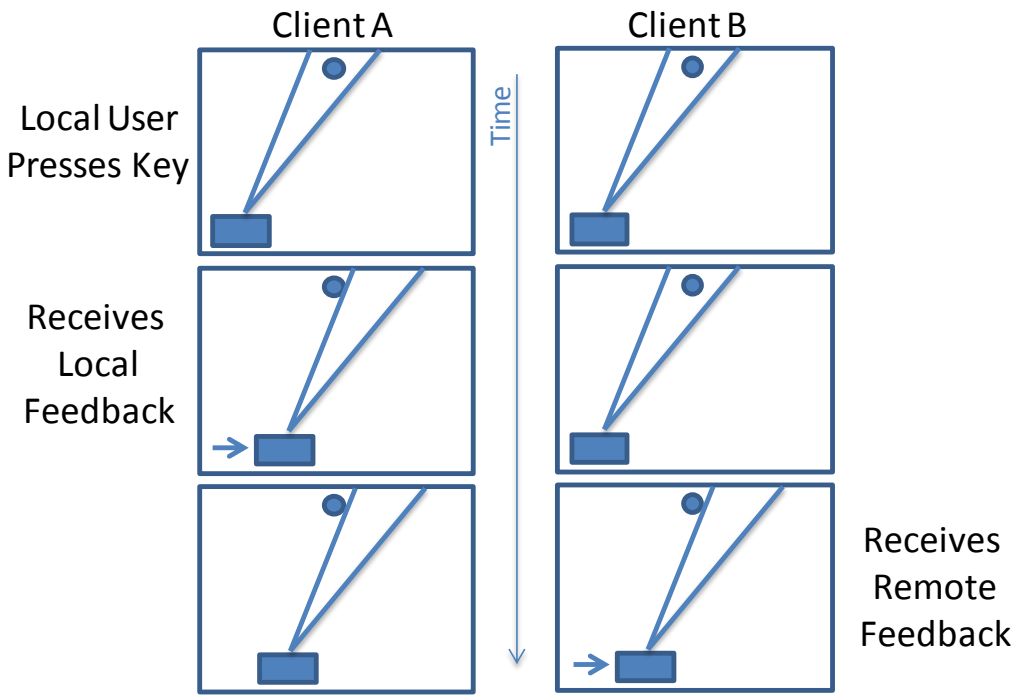


Figure 4.10. The feedback is shown at different times

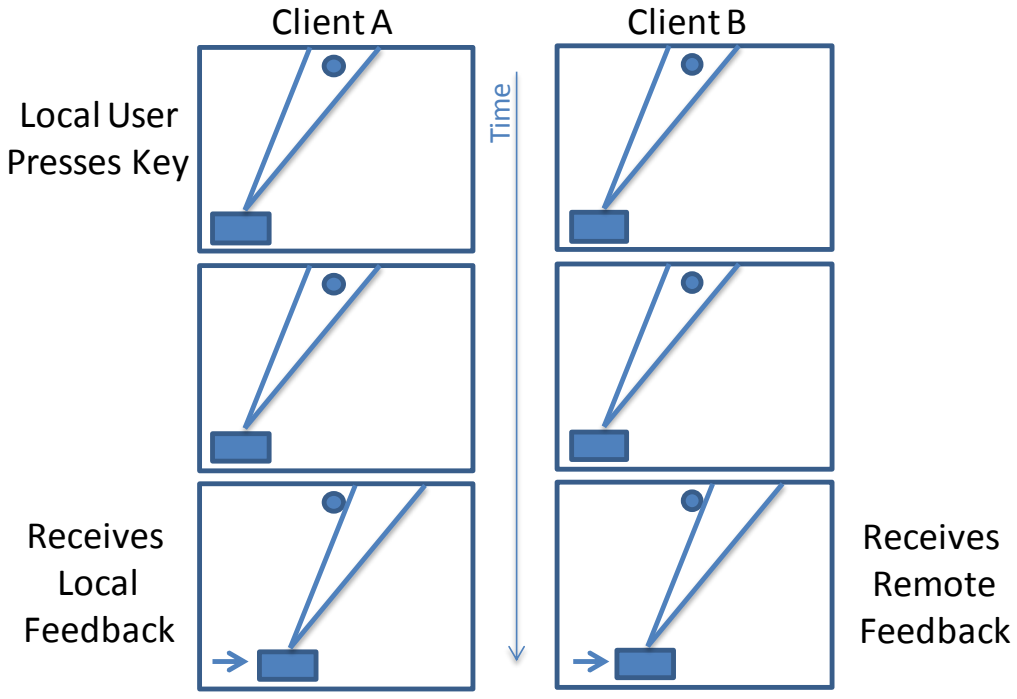


Figure 4.11. The feedback is shown at approximately the same time

CHAPTER FIVE

EARLY EXPLORATORY STUDIES

Three exploratory studies took place before a formal evaluation was run. The goal of these studies was to find a task that could properly test the limits of real-time distributed collaboration, which included discovering how users behaved during tightly-coupled interaction, and how they compensated for network delay during activities that were tightly-coupled.

5.1 Spacewar Duo

The goal of the first two exploratory studies was to determine how difficult it was for pairs of users to work together in a tightly coupled activity. Specifically, the purpose was to discover what strategies users would adopt to cope with the tightly-couple play, and what problems the users seemed to struggle with. These studies used variations of a game called Spacewar Duo. In Spacewar Duo, two players were put in control of one ship. One player had to pilot the ship while the other player controlled a gun turret on top of it (see Section 4.4.1). The test was specifically set up such that the actions of one player immediately affected the other in predictable ways. When the pilot rotated the ship, the turret rotated along with it. When the turret fired, the recoil pushed the ship off course. And when the turret rotated, the ship would be turned in the opposite direction with reciprocal rotational force.

The first exploratory study was played in the style of a 'deathmatch' (Figure 5.1), where each pair of players in a ship continually tried to shoot the other ships without getting shot themselves. In this task, players were to play for approximately one hour, but many preferred to play for longer. Gaming sessions generally lasted between one to three hours.

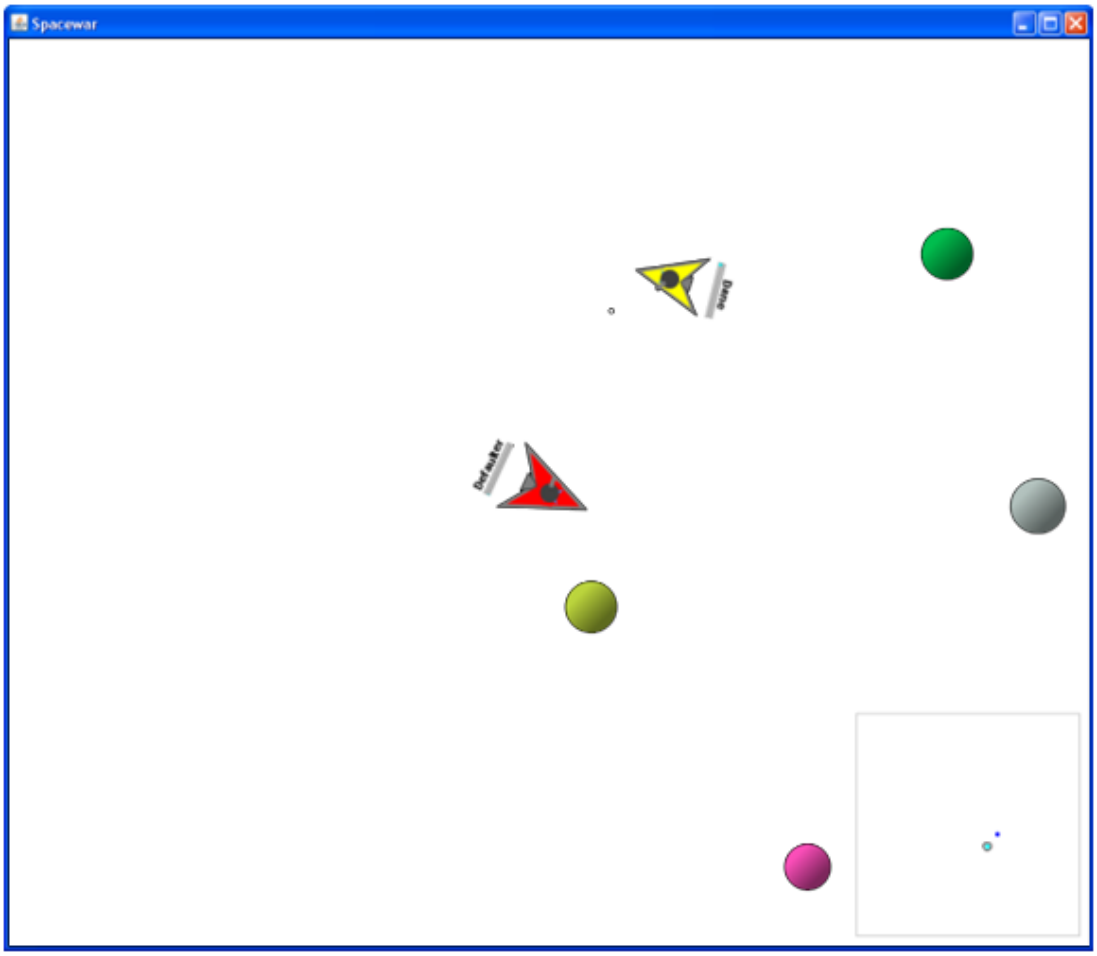


Figure 5.1. Two players worked together in a deathmatch-style multiplayer game (solid black has been inverted to white).

The second exploratory study was the similar to the first, but various levels of artificial network latency were injected into the clients. The goal of the second exploratory study was to determine how network delay changed the strategies that users adopted, and how users tried to compensate for the delay. It was important to be able to measure the performance and actions of the individual users, so the test was refined from the first study to use only one pair of users at a time. The task was changed from a multiplayer deathmatch to a simple navigation task. The players had to navigate through a large maze while shooting stationary yellow targets, completing the task as fast as possible. The maze was not difficult; there was only a single path of travel, and flashing red arrows guided the players to the end. The task took approximately two to six minutes (Figure 5.2).

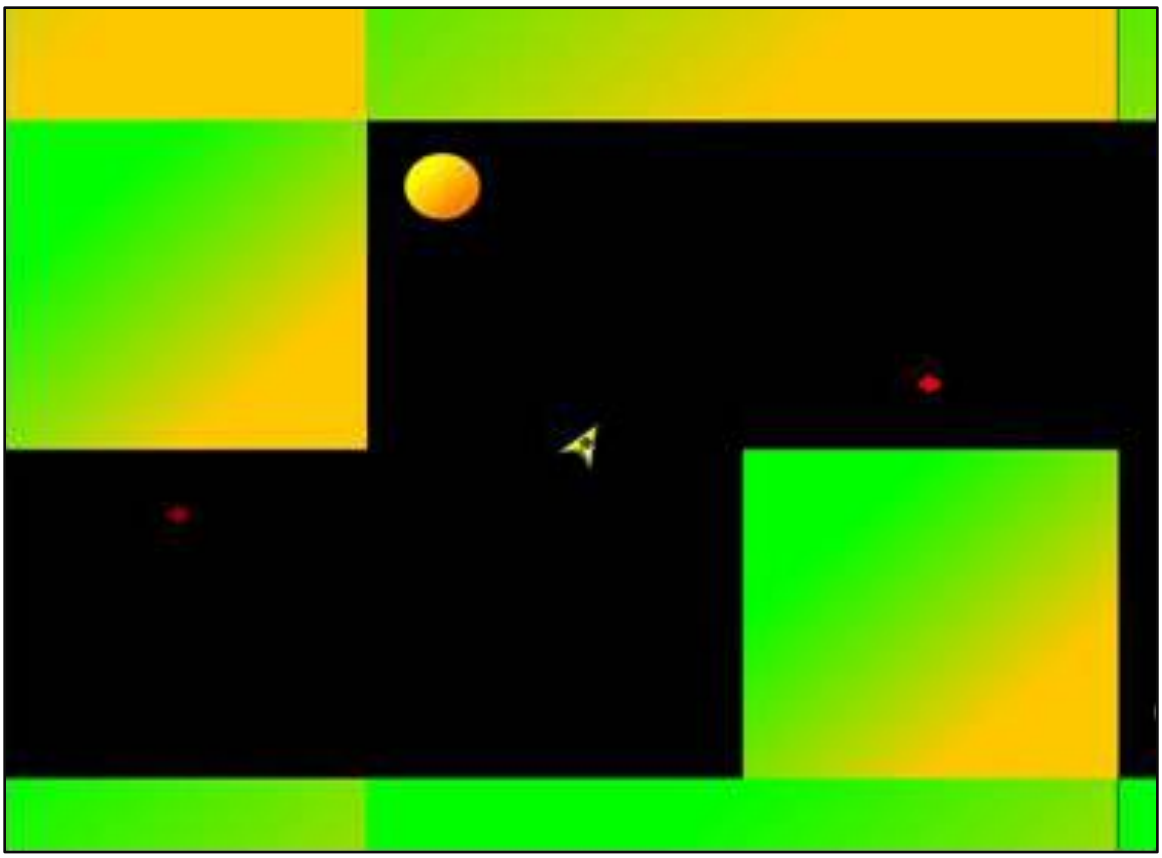


Figure 5.2. Two player teams navigated a single-track maze. Their goal was to shoot all of the targets (yellow circles) as quickly as possible.

The behaviour of the players in both of these studies was generally the same. It was determined that if the players could divide the task into parts that they would almost always chose to do so, even at the cost of performance. The unfortunate choice of setting the game in a frictionless outer space contributed to a "turn-taking" strategy. The pilot would aim the spaceship such that it would drift in the general direction of their target, then the gunner would aim and fire the turret. When the ship drifted too far away from their target, the gunner would stop firing and the pilot would start to move the ship around again. The turn-taking strategy was remarkably resilient against network delay, but the turns were long enough that it was questionable whether the task was tightly-coupled enough for the purpose of my topic.

However, one main contribution of these studies was revealing the presence of cyclic overcompensation in the users' behaviour. Cyclic overcompensation is a problem

where both users try to compensate for the actions of their partner at the same time in a similar way. When their actions are merged, the combined compensation is too much, so the users both try to compensate in the opposite way, which causes the same problem in the opposite direction. Network delay between distributed clients tends to exaggerate cyclic overcompensation, because the users do not realize that their partner is compensating as well until feedback is received from the network. Cyclic overcompensation was a significant problem for users in Spacewar Duo, and it muddied the quantitative results of these early studies to the point where only the comments and general performance of the users were useful in evaluation.

5.2 Ball-bouncing Game

The goal of the third exploratory study was to explore another kind of task to determine if it would be better suited for a study about tightly-coupled interaction. In this two player game, two balls would fall from the top of the screen, and each player would bounce one ball over to the other player's side of the screen while simultaneously trying to catch the ball being bounced toward them (Figure 5.3 and Section 4.4.2). Six participants were recruited locally to determine if the ball-bouncing task would be tightly-coupled enough for a formal study.

Once again, users preferred to divide the task between the two of them. When asked where they were focusing their attention, users commented that they were focusing only on their own side of the screen, even when the game physics were set to a very slow or extremely fast pace. Although both players were looking at the same display, there was no verbal effort to work together, nor did the players seem to know what the other was doing.

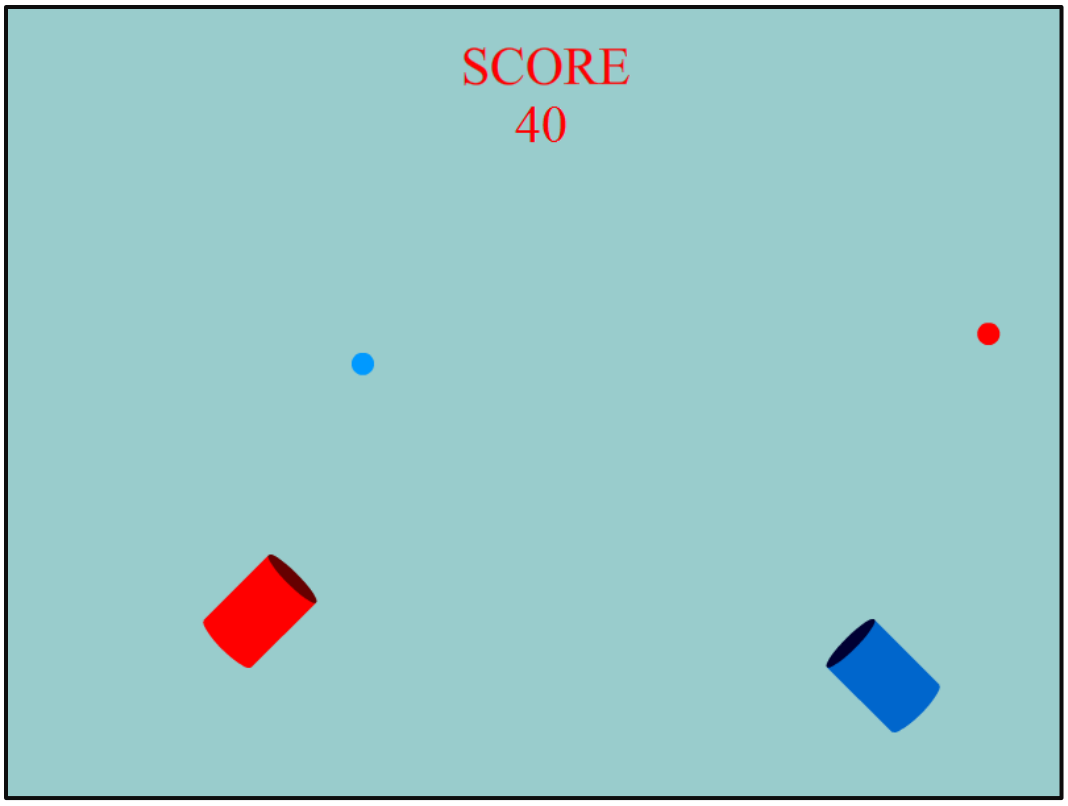


Figure 5.3. In this two player game, each player would bounce a ball over to the other player's side of the screen while simultaneously trying to catch the ball being bounced toward them.

5.3 Summary

These three exploratory studies helped determine the task used in the formal study, as well as how it was evaluated. In general, it was found that users would almost always choose to divide the task between them if they could. Therefore, the task for the formal study should have users working together in a way that is impossible to separate, such that the task studies tightly-coupled work instead of how effectively users were able to decouple their actions.

Although metrics such as each player's completion time or the number of kills deaths during play been used as performance metrics in studies about the effect of network delay on user performance (Park and Kenyon, 1999; Beigbeder et al., 2004; Sheldon et al., 2003), these early studies showed that those metrics are also heavily influenced by other factors, such as the participants' skill in the task, how effectively they

were able to work together, and random chance. The number of errors made by the participants was a better indicator of the effect of network delay on their performance, but it was difficult to evaluate user errors in an application such as Spacewar Duo or the Ball-Bouncing game for two reasons: the chaotic and unconstrained nature of the shared environment, and complexity of users' interactions.

First, the free movement of their avatars in all three of the studies allowed for a wide variety of situations that could affect the users' behaviour. A consistent evaluation of tightly-coupled interaction is impossible in tasks that allow too much variety, because different situations elicit different responses from the users, which significantly changes their performance.

Second, it is difficult to evaluate users' performance at any point in time when both users are interfering with each other. Mistakes and adjustments tend to bounce back and forth between them, causing effects such as cyclic overcompensation. Rather than using a metric that requires a task that takes a long time to complete, errors by the participants should be clearly discernable and immediate, such that the effect of delay on their performance can be more directly evaluated.

CHAPTER SIX

EVALUATION

A formal study was carried out to determine if FFS improved users' ability to cope with visual delay while working together through distributed groupware.

6.1 Goals

Overall, this study explored the limitations of tightly-coupled collaboration with distributed groupware, and determined if FFS allowed users to cope more easily with network delay. This study examines the following questions:

- How much does network delay affect users' performance?
- Does FFS help users perform better?
- Does FFS make the task easier or harder to perform?

6.2 Study Design

There are two independent variables in this study: feedback display policy, and the amount of latency between the users. There were two display policies: immediately displayed feedback, and FFS. There were six levels of latency: 0, 100, 200, 300, 400, and 500 milliseconds. The dependent variable was the users' accuracy during the task, which was the amount of time they were successfully hitting the target.

6.3 Task

The task was a simple multiplayer game. The participants controlled either the driver or the turret operator of a 2D fire truck (described in Section 4.4.3), and their task was to work together with their partner to aim a stream of water at a small fire for as long as possible.

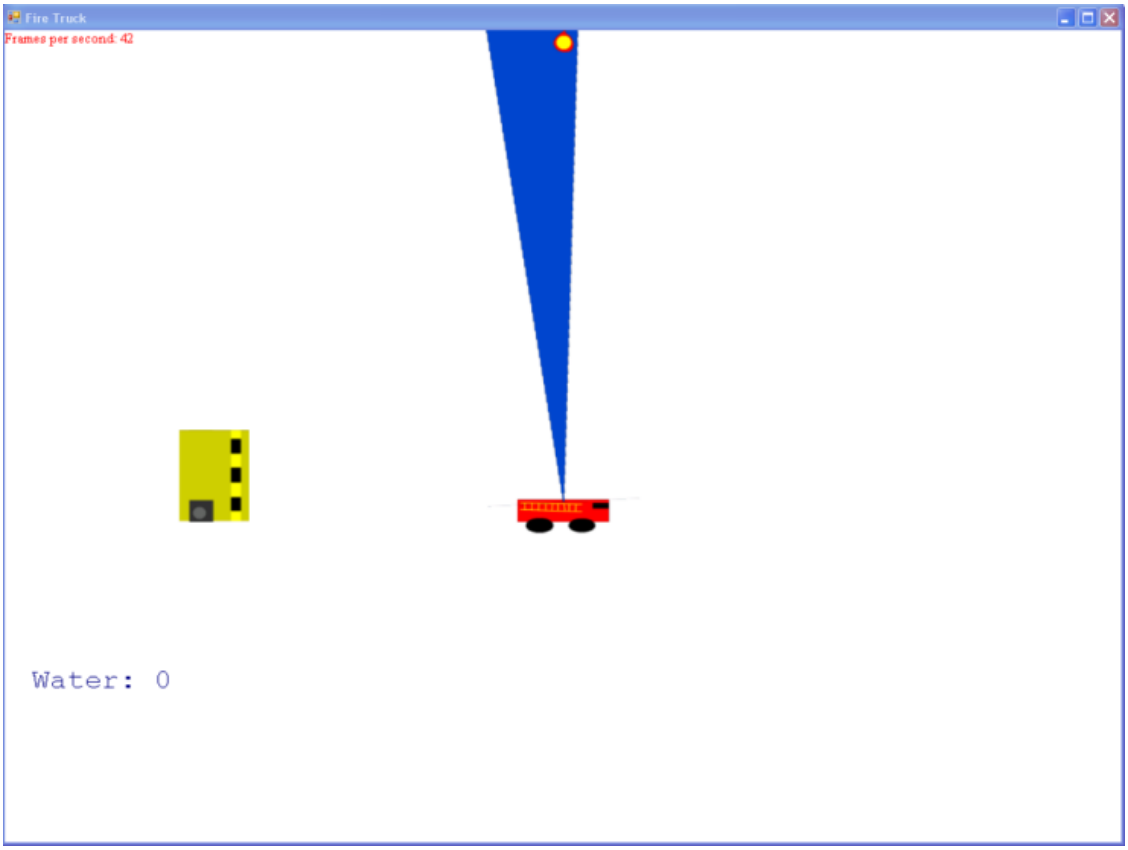


Figure 6.1. One user moved the truck left and right across the screen, while the other user controlled the angle of the stream of water.

The task was a simplified example of a collaborative interaction between two users. The users worked together to complete a simple goal in a real-time distributed application. During each trial, visual delay was injected between the participants to determine what the effect would be on their performance. To simplify the results of this study such that they could be more easily understood, this task was designed such that while both participants worked together to complete the task, only one of the participants' actions interfered with the other. The generalizable task for the users was to work together to maintain a goal state for the length of a trial. Their goal was always achievable at any moment in time so that it was immediately obvious at what times during the task coordination broke down between the two users, because they will fall out of the goal state.

The driver of the fire truck used the arrow keys on the keyboard to move the truck left and right. The driver's job was to collect water tanks by driving in front of them with

the fire truck. When a water tank was collected, its water was added to the fire truck's supply, and a new water tank appeared somewhere else on the screen.

The other user controlled the turret with the mouse. The goal of the turret operator was to aim the truck's stream of water at the fire. However, if the fire truck had no water left, the spray from the turret would not be enough to put out the fire. Therefore, both of them had to work together to complete the task.

During the experiment, network latency was artificially adjusted to various levels to determine what effect that amount of delay had on user performance. The different levels of latency were chosen based on previous research (described in Section 2.4). Network jitter was non-existent or insignificant on the dedicated network, so the actual amount of delay at any particular level of network latency was relatively constant during the task. Participants were informed that what they saw on their screen would not be the same as their partner's because it took time for information to travel between their computers. They were also informed that communication would be very important in this task, and that they were free to talk to each other as much as they wanted, and that comments about the system and their strategies were encouraged.

The networking model used for the Fire Truck game was designed to simulate common networking architectures, including those that are peer-to-peer or centralized, by focusing on a specific user, the water turret operator. From the viewpoint of the user, there are two types of information that are feeding into a client involved in a tightly-coupled activity: the actions or feedback of individual members, and the combined result of the actions of the entire group. The actions or feedback from individuals inform the new actions of the local user, allowing them to compensate or adjust their own behaviour in response. The combined feedback for the entire group shows the local user how successful their actions were.

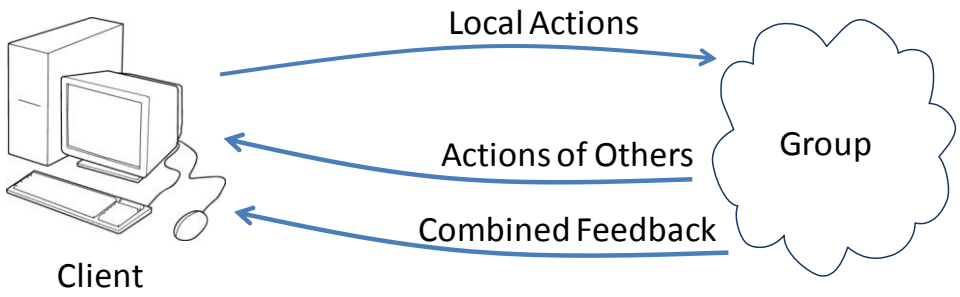


Figure 6.2. Representation of information flowing to and from the water turret operator.

There were two clients in the study, but the relationship between the users' movements was only one-directional. The turret was attached to the fire truck, but the driver of the fire truck was not directly affected by the movements of the water turret. If the local user is the water turret operator in Figure 6.2, then the movements of the fire truck driver represent the actions of others that are affecting the local user. The combined group feedback, which is the product of the local user's actions combined with the actions of the rest of the group, is represented as the state of the fire. A lit fire meant that the group was failing at the task, while an unlit fire meant that they were successfully hitting the target with water.

Network latency affects tightly-coupled tasks in two independent ways: it delays the actions or feedback from other clients, and it delays the group's combined feedback. These delays are independent because in the process of completing the task, local users react to the actions or feedback from other users, and after they react they determine if their actions improved the group's combined feedback. In tightly-coupled tasks, the information is streaming between the clients, so the users are constantly trying to adapt their behaviour for the actions of other clients, while using the combined feedback for the group to determine how good they are doing.

6.4 Procedure

Twenty-four participants, seven female and seventeen male, were recruited from the University of Saskatchewan. Participants ranged in age from 18 to 33 years (mean of 22.4 years). All were regular computer users (minimum of ten hours per week, mean of 32.25 hours per week), and twenty reported that they had played multiplayer online

games before. Thirteen described themselves as highly experienced players of multiplayer online games. Eight of the thirteen highly experienced players reported that they have experienced significant network delay in games, two reported that they have never experienced significant network delay, and the remaining three were unsure or did not answer the question. Of the seven inexperienced players who had played multiplayer online games before, four reported that they had experienced significant network delay in games, two reported had they have never experienced significant delay, and the remaining participant did not answer the question.

Participants carried out the task in groups of two. At the start of the session, the participants were read a script describing the task and what they were to complete. The total time to complete a session was approximately one hour. Each session was divided into two separate runs: each run consisted of six trials, and each trial lasted two minutes. The first thirty seconds of every trial was considered training time, to allow the users to familiarize themselves with the task during each level of delay. There was a short break between each trial, during which participants filled out a NASA TLX form, describing their experience with the trial they had just completed. In each session, one run used FFS while the other did not, and the order of the runs was changed between sessions to diminish the effect of the users learning the task.

The first trial in each run had a minimum amount of network delay, so that the users could learn the task quickly. However, every successive trial had a progressively larger amount of network delay, so the users could knowledgeably adapt their behaviour to compensate.

6.5 Apparatus

Two computers were used, each with a monitor set at 1024 by 728 pixel resolution, an optical mouse and keyboard, and the machines were connected via ethernet cable to a dedicated router. There were 13 to 26 milliseconds of real latency between the clients at all times. The participants were in adjacent cubicles, and there was a physical separation between them such that they could not see the other person or the other person's monitor (Figure 6.3). The application was custom built in C# using GTC (described in Section 4.1) and DirectX (http://msdn.microsoft.com/directx/). Users'

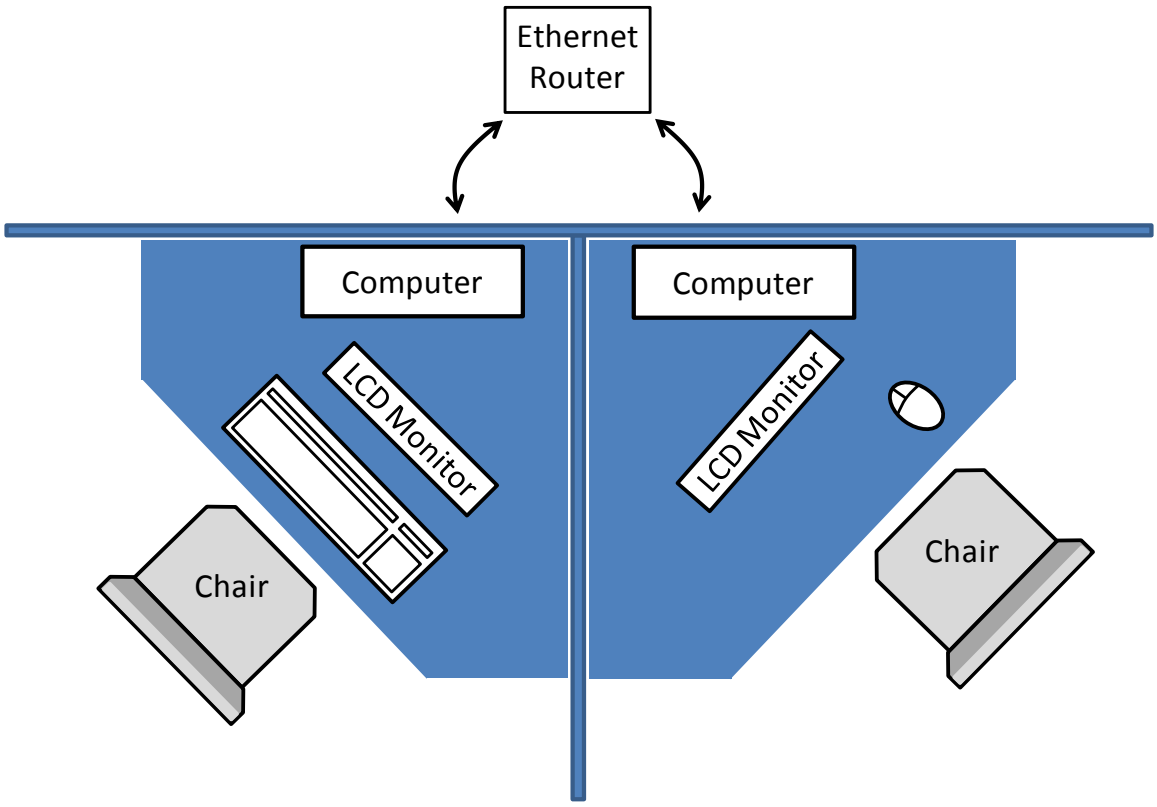


Figure 6.3. Diagram of apparatus.

screens were updated and network messages were handled approximately 60 times per second. Each user's device inputs and positions were recorded every screen update, along with the current angle of the turret and the position of the fire truck.

6.6 Results

The results are organized by two main factors. First, I will report the effects of increasing delay on performance, and then discuss the effects of using FFS. Second, I will compare people's perception of effort between immediately displayed feedback and FFS.

6.6.1 Effects of Delay Amount

Performance in the task was measured in terms of accuracy – the fraction of the trial that groups were able to maintain the goal state (i.e., keep the stream of water on the fire).

Overall, accuracy decreased as network delay increased. As can be seen in Figure 6.3, average accuracy over all conditions dropped from nearly 95% at the lowest level of delay, to less than 50% with 500ms delay. A 2x6 ANOVA showed a significant main effect of delay amount ($F_{5,45}$=76.38, p<0.001). This confirms the results of previous studies that user performance in coordinated activities is directly affected by delay.

6.6.2 Effects of Using FFS

The 2x6 ANOVA also showed a significant main effect of the use of FFS ($F_{1,9}$=77.38, p<0.001). Figure 6.4 shows that the overall difference between immediately displayed feedback and FFS for all levels of delay was more than 18%.

In addition, there was a significant interaction between delay amount and feedback display policy ($F_{5,45}$=7.904, p<0.001). The interaction arises because (as expected) there was no difference between immediately displayed feedback and FFS at the lowest level of delay, and the difference was small at 100ms. The advantage of FFS is greater at higher levels of delay (see Figures 6.5 and Table 6.1). The largest difference between categories was at 200 ms of delay, which is also the category with the largest drop in performance for immediately displayed feedback. The average performance with immediately displayed feedback dropped 34% between 0 and 200 ms, whereas performance dropped only 7% when using FFS.

6.6.3 Perception of Effort

Users' subjective perception of effort was measured using NASA TLX questionnaires. NASA TLX is a standardized questionnaire that allows users to rate the mental and physical demand of the task, as well as their frustration, effort, and performance, on a 1 to 7 scale (see Figure 6.6).

The NASA TLX results showed that the users believed that the tasks consistently became harder as the visual delay increased: 2 x 6 ANOVAs on each of the aspects rated on the NASA TLX by the users showed a significant result in every case except for physical effort for the driver (Table 6.2). However, there were no significant results found between users using FFS and those who were not: 2 x 6 ANOVAs on each of the aspects rated on the NASA TLX by the users did not show any significance (Table 6.3).

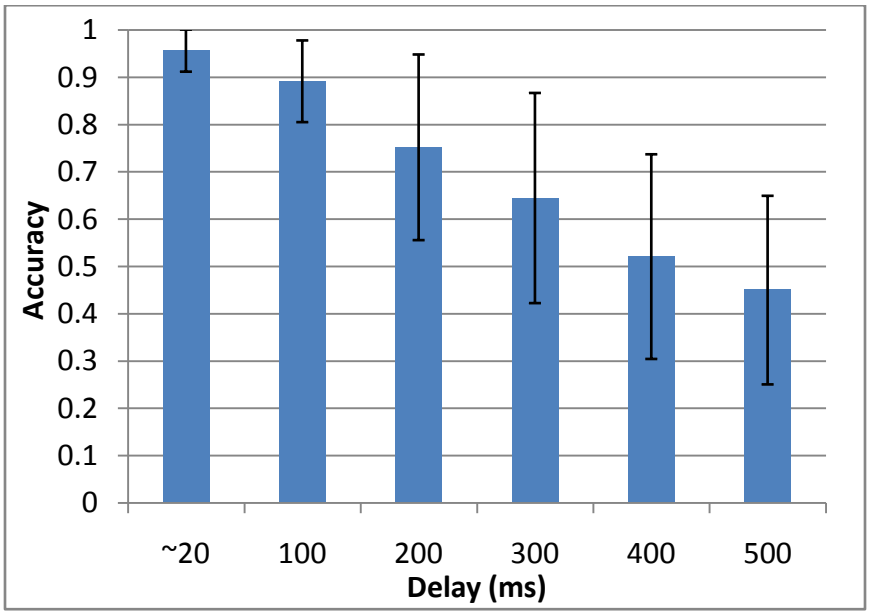


Figure 6.4. Mean accuracy over all delay amounts; Error bars represent 95% confident interval.

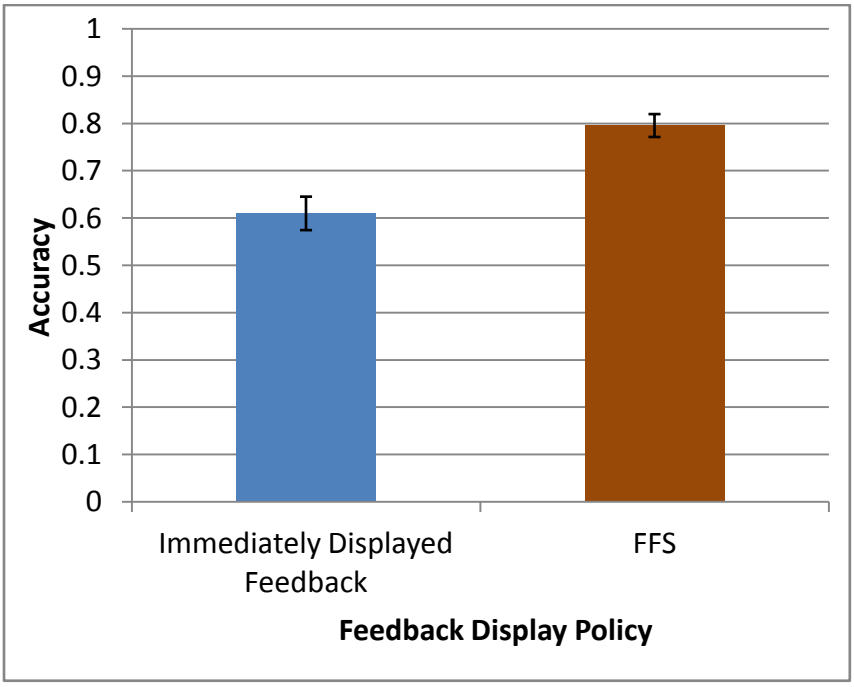


Figure 6.5. Mean accuracy between feedback display policies; Error bars represent standard error.

Table 6.1. Results from the collaborative task, represented graphically in the chart above; IDF represents immediately displayed feedback.

Delay (ms)	Accuracy (%)	
	IDF	FFS
~20	0.9571	0.9565
100	0.8424	0.9415
200	0.6167	0.8882
300	0.5175	0.7727
400	0.3878	0.6546
500	0.3392	0.5616

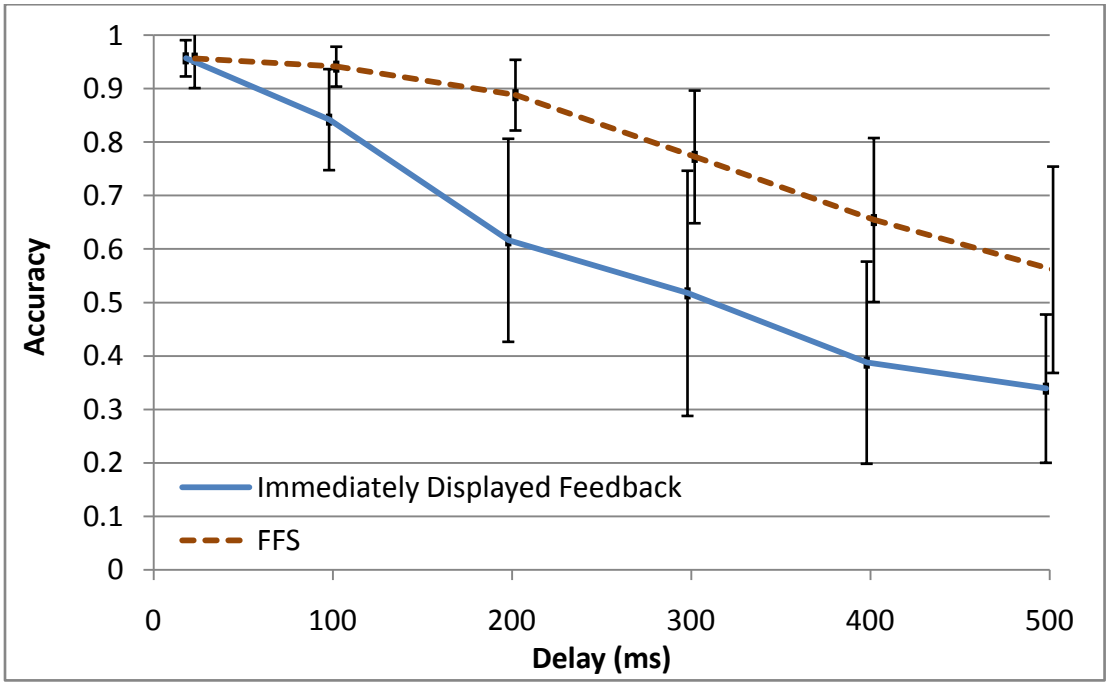


Figure 6.6. Mean accuracy in the collaborative task, by delay amount and feedback display policy; Error bars represent 95% confidence intervals..

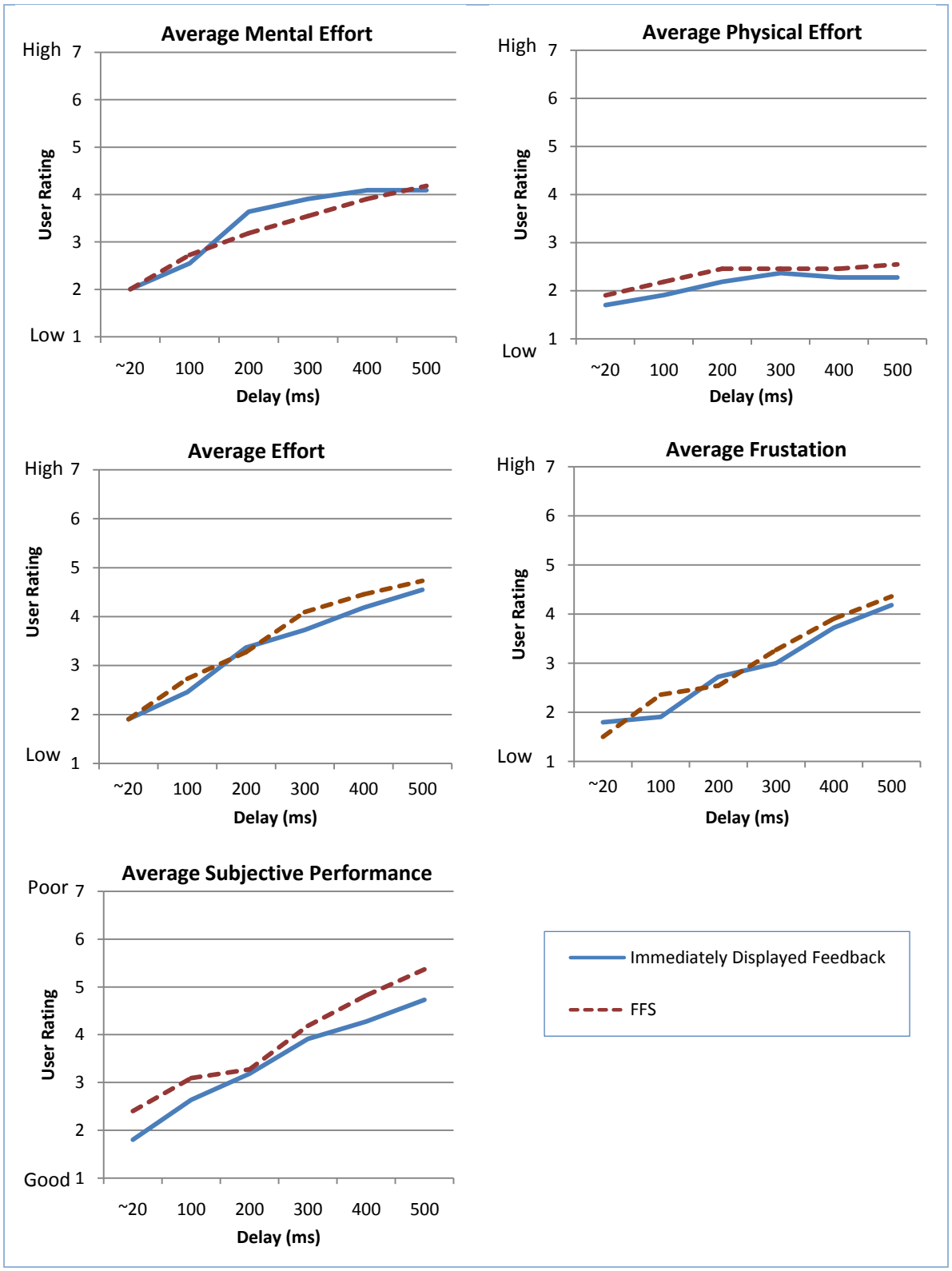


Figure 6.7. Average NASA TLX scores rated by users.

Table 6.2. AVONA results for NASA TLX: difficulty vs. delay; Non-significant results are highlighted.

	Driver	Turret
Mental	$F_{5,45}$=12.59, p<0.001	$F_{5,45}$=19.043, p<0.001
Physical	$F_{5,45}$=1.558, p=0.191	$F_{5,45}$=4.543, p=0.002
Effort	$F_{5,45}$=4.866, p=0.001	$F_{5,45}$=19.565, p<0.001
Performance	$F_{5,45}$=7.259, p<0.001	$F_{5,45}$=11.992, p<0.001
Frustration	$F_{5,45}$=7.212, p<0.001	$F_{5,45}$=21.266, p<0.001

Table 6.3. AVONA results for NASA TLX: immediately displayed feedback vs. FFS; All results are non-significant.

	Driver	Turret
Mental	$F_{1,9}$=0.855, p=0.379	$F_{1,9}$=3.418, p=0.098
Physical	$F_{1,9}$=1.328, p=0.279	$F_{1,9}$=0.989, p=0.346
Effort	$F_{1,9}$=0.254, p=0.254	$F_{1,9}$=0.380, p=0.553
Performance	$F_{1,9}$=0.227, p=0.645	$F_{1,9}$=1.338, p=0.227
Frustration	$F_{1,9}$=0.730, p=0.415	$F_{1,9}$=0.021, p=0.887

CHAPTER SEVEN

DISCUSSION

This study explored the domain of tightly-coupled interactions with various levels of network delay, and introduced a technique to reduce the effects of network delay on group performance. The results of the formal study show the following:

- I confirmed previous studies, showing that network delay significantly impairs tightly-coupled interactions in distributed real-time groupware.
- I discovered that FFS reduces the effect of network delay on tightly-coupled interaction.
- I found that users' perception of task difficulty did not significantly change between immediate feedback and FFS at similar levels of delay.

The remainder of this section will give possible explanations for my results and answer potential questions about the technique. I will also discuss how my results can be generalized for groupware designers, and suggest future directions for my work.

7.1 Explanation of Results

7.1.1 The Difference between Immediately Displayed Feedback and FFS in Tightly-coupled Interaction

People were able to cope for network delay more effectively with FFS than without it. There are three probable reasons for the improvement: feedback became synchronized between the clients, users became aware of the delay, and the controls slowed users down.

Synchronizing feedback between the clients increased the predictability of users' actions, because it restored the causality of local and remote events in the shared environment. For example, when playing the fire truck game with immediate feedback,

one client may show the water hitting the fire while the other client shows the water missing the fire completely. With FFS, both clients show the same feedback at approximately the same time, so the users can agree on whether they are hitting the fire or not.

When users became aware that there was latency between performing an action and their feedback, they began to adapt their behaviour to cope for what they perceived as less responsive controls. The latency between users' controls and their feedback was the same as the current amount of latency on the network, so when the users adapted their behaviour to compensate for the less responsive controls, they also coped for the delay on the network.

Another possible contribution to the users' performance is that the latency between their controls and local feedback slowed the users down. Slow users are more predictable than fast ones, which could allow them to work together as a better team. For example, in the Fire Truck game, users' performance with FFS was better than with immediately displayed feedback, even though the users were slowed down.

7.1.2 The Subjective Perception of Effort

There were no significant differences found when comparing subjective perception of effort scores for immediate feedback and FFS. As the network delay increased, all of the users confirmed that the task seemed to require more effort in every way (except physically). However, users rated the average effort of immediate feedback and FFS as approximately the same at any particular level of delay.

At high levels of delay in this task, there seems to be a balanced trade-off between users not understanding what to do with immediately displayed feedback, and users losing local control with FFS. Users could easily perform their part of the task when they were using immediately displayed feedback, but they became confused or frustrated when their feedback showed that the group was not succeeding, such as when they saw actions from their partner that did not make sense. The effort of performing the task with immediate feedback was due to the fact that users had to learn how to cope with network delay without any indication of what they were doing wrong. When using FFS, users

could easily tell what they were doing wrong, but it was more difficult to perform their individual tasks because of the local feedback latency.

7.2 Design Questions

7.2.1 Does feedback latency from FFS have to be obvious to the user?

Techniques like Timewarp try to hide network delay from the user, but when the state of the environment is corrected to account for late information that has been delayed because of network latency, then the sudden changes are a surprise to the user. Local Perception Filters try to gradually correct for latency by manipulating the relative speeds of objects between users' avatars, but it assumes that users' are using avatars that are separated by a distance that must be transversed before they can affect each other and as avatars move closer together and become more tightly-coupled, hiding network delay with this strategy eventually becomes impossible. In tightly-coupled interaction, users influence each other immediately and continuously as they perform their task, so deceiving the user about the state of the environment or about what their partner is doing is detrimental to any sort of organized collaboration. FFS allows users to perform tightly-coupled interaction with real-time distributed groupware because it reveals the hidden delay to the users; it is necessary to make the delay explicit, because the users are the ones that are coping with it.

One way to hide the delay without losing the benefit of FFS is to immediately display contextually appropriate feedback to acknowledge user input, while still delaying the results of their actions (Figure 7.1). For example, a character avatar could appear to wind up her leg before kicking a ball, a moving spaceship could take a short time to completely change its momentum, or a car could squeal its tires before accelerating. The interval of time that FFS adds to the local feedback is often small, and this technique may work well because many video games and GUIs already apply immediate feedback to actions with delayed results, such as through animations and visual effects.

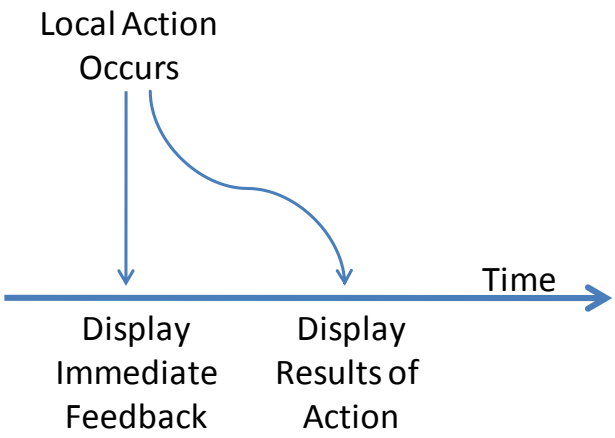


Figure 7.1. Disguising FFS in animations or visual effects can maintain the feeling of immediate control, while the results of local actions are still delayed.

7.2.2 How can these results be generalized for designers?

There are two ways that these results can be generalized for designers: how network delay affects user performance in tightly-coupled tasks, and how FFS is able to improve group performance in tightly-coupled tasks.

First, various kinds of tasks, ranging from real-time multiplayer games to a variety of isolated activities (such as those discussed in Chapter two), have been studied and each evaluation has determined that delay does indeed affect the participants' performance. However, previous studies (Chen et al., 2007; Park and Kenyon, 1998) used performance metrics such as completion time or users' kill-to-death ratio; these metrics are highly dependent on the application and the parameters of the system, so any comparison between their techniques or tasks is difficult. Small changes to the rules of the game or the physics of the shared environment could change their results drastically, even though the task is largely the same. My results are more generalizable than previous work because they are more directly connected to whether or not the user knew how to correctly adjust their actions at any point in time during the task, regardless of the specific application or the parameters of the system. Since many tightly-coupled tasks occur in real-world software, my results can be used to predict how user performance will degrade at certain levels of network delay.

Second, these results show how FFS can improve user performance in tightly-coupled interaction. The study task was an implementation of the general action-feedback cycle that can occur in many collaborative situations, such as object handoff, true shared manipulation (e.g., two people moving an object simultaneously), shared aiming, fine-grained turn taking, and numerous game applications, so using FSS in other tightly-coupled tasks should provide similar results.

7.2.3 What amount of network or feedback delay is too much?

Network delay is a significant problem for real-time distributed groupware on the Internet, so groupware designers should know the operating limits for their applications. FFS is a technique that allows users to work together more effectively with network delay, but it does not reduce the amount of delay that users experience. At a certain level of delay tightly-coupled work becomes impossible, and users try to compensate by decoupling the task. In Park and Kenyon's tightly-coupled task (1999), users resorted to a move-and-wait strategy, where they would make a small movement and then wait for their clients to resynchronize from the network. In Spacewar Duo, users would take turns controlling the spaceship, passing control between the pilot and the gunner. These strategies allow users to compensate for the large amounts of network delay, but they are not very efficient because users spend time a large amount of time waiting, instead of contributing toward the task. Therefore, determining the amount of delay that users can cope with while still performing tightly-coupled interaction is important for establishing what network conditions are acceptable for any particular real-time distributed groupware application.

During the task, there were two distinct responses as users were exposed to large amounts of network delay. With immediately displayed feedback, users had to imagine how the delay was affecting them in order to adjust their actions, but with large amounts of delay, they became confused and did not know what to do. With FFS, users could see how network delay was affecting their actions, but with large amounts of delay, it was difficult to correctly control their actions because of the large amount of feedback latency. These responses generally occurred at 400 or 500 milliseconds of delay, and are the direct result of excess network or feedback delay on the users.

7.2.4 How will network jitter affect FFS?

FFS was not tested with different levels of network jitter. During the Fire Truck task, participants were exposed to relatively consistent amounts of network delay. There were very small amounts of real network jitter (~13 ms) between the clients because the two computers were connected via a dedicated Ethernet router. Real-world applications may be exposed to high levels of jitter because of router delays, lost and retransmitted packets, or variance in the network. Network jitter has been found to significantly impair people's ability to predict others' actions, so it may adversely affect user performance in tightly-coupled tasks (see Section 2.4).

However, the amount of jitter on a network is usually very small compared to latency, particularly on local-area networks or broadband Internet. Additionally, high amounts of jitter can be exchanged for network latency by buffering data on the receiving client. Future work will determine the effect of network jitter on user performance in tightly-coupled tasks.

7.2.5 Is revealing network delay always a good idea?

The advantage of FFS is that it allows users to cope with network delay instead of trying to hide it from them, which lets users to perform tightly-coupled interaction more easily. However, forcing the users to cope with network delay has some potential drawbacks. Notably, placing the burden of coping with network delay could increase the effort of using the application, and the perceptible network delay could cause the application to be seen as lesser quality than applications that hide network delay from the users.

One of the most important and surprising results from my study is the lack of a significant difference of the perceived difficulty of the task between immediately displayed feedback and FFS. If the effort of coping with network delay and feedback delay is the same when performing tightly-coupled tasks, then the performance benefit when using FFS comes without a cost to the user.

In real-time distributed groupware, both network delay and feedback delay are seen as things to be avoided. Network delay is a natural result of the distance between distributed users as well as packet-switching and processing delays, so despite the best

efforts of researchers in this area, it will probably never completely disappear. The techniques for hiding network delay discussed in Chapter two try to simulate an experience as if there were no delay at all, but those techniques also assume that user interaction is a rare and intermittent thing, such that the results of their interaction can be corrected or limited without introducing large inconsistencies. Tightly-coupled interaction requires constant feedback from the participants, so strategies that try to deceive the user into believing that there is no delay could instead frustrate their attempts at collaboration. However, revealing network delay to the users (such as with FFS) could lower their opinion of the software, because they could see it as lesser quality than one that is able to hide network delay successfully. In truth, FFS is revealing the real timing of users' actions as seen on remote clients, since the feedback from their actions is being delayed by the same amount as the current delay on the network. Since users can see the results of network delay on their actions, tightly-coupled activities will benefit from FFS, even though users' opinion of the application may be seen as worse because of the perceptible delay.

7.2.6 How will users' behaviour change now that they can perform tightly-coupled interaction?

Research into tightly-coupled interaction may be related to similar research in tabletop displays, where users collaborate in a face-to-face setting with digital artefacts. Tabletop groupware systems allow people to work together in a tightly-coupled manner, because they can directly see and interact with the other users. These systems allow users to use the same collaboration techniques that they use in the real world, without interference from network delay. Since there are few examples of tightly-coupled interaction in distributed groupware, tabletop collaboration may be the best example of what natural, distributed, tightly-coupled work should be.

Many of the principles of tabletop displays already apply to distributed groupware. There are strategies for handling conflicts between users, for adjusting the display of information for individual points of view, and for negotiating between individual tasks and shared work. However, one area that has not been explored yet in distributed groupware is user territoriality. When using tabletop displays, users tend to

divide the tabletop into personal and shared workspaces. Since people dislike reaching into the space directly in front of another user sitting at the table, users tend to use the space in front of them as a personal working area for their current task. In future work, it would be interesting to determine if the concept of personal space transfers to distributed groupware when users are able to interact in a tightly-coupled manner.

7.2.7 How difficult is FFS to implement?

Using FFS does increase the complexity and the development time of the system. The software delays only local feedback while remote actions are displayed immediately, so the logistics of what to draw on the screen can be difficult. The simple solution of directly buffering the input cannot always be used in applications that also send the results of their input (i.e., kills, movement resulting from in-game physics, conflicts) because the results are not known until the input is applied to the shared environment. Implementing FFS in such a system is not impossible, but designing a systematic way to implement FFS in complex applications is an area of future work.

7.3 Summary

The results of the study show that FFS can significantly improve group performance, with no similar increase in effort, in tightly-coupled tasks. There are three probable reasons for the improvement in group performance: feedback was synchronized between the users, users were made aware of the network latency between them, and the feedback latency slowed users down and made them more predictable.

Additionally, designers should recognize that:

- The feedback delay must be apparent to the user such that they can compensate for it, but it may be possible to hide the local delay.
- FFS does not reduce the amount of delay on the network, but allows users to cope with it more easily in tightly-coupled tasks
- Revealing the network delay may lower users' opinion of the local responsiveness, but there was no significant difference between their perceived effort of coping with immediately displayed feedback and the FFS technique.

- User behaviour may change as a result being able to perform tightly-coupled interaction.
- Implementing feedback latency for only local actions may be difficult in complex applications.

This research is mainly a demonstration of the benefit of synchronizing user feedback, but it also demonstrates the effect of network and feedback latency on the user. Therefore, two main lessons that software designers should take from this thesis: minimize your application's overall latency, and synchronize user feedback for tightly-coupled tasks.

CHAPTER EIGHT

CONCLUSIONS

Tightly-coupled interaction is common in many kinds of face-to-face activity, but it is difficult to perform in real-time distributed groupware. The main problem is network latency that disrupts people's ability to coordinate their actions. To reduce the effects of network delay on tightly-coupled interaction, I introduced a technique called Feedback-Feedthrough Synchronization. FFS displays user feedback at the same time for local and remote participants, preventing one person from getting ahead of the other in coordinated interaction. An experiment found that FFS had substantial positive effects on group performance, and that the effect grew larger with increasing delay.

7.4 Contributions

The main contribution of Feedback-Feedthrough Synchronization is that it raises the threshold of usability for real-time distributed groupware. Tightly-coupled tasks can now be performed at higher levels of delay, and users can work together more easily than they could before. FFS is a technique that synchronizes user feedback to enhance their understanding of the effect that network delay has on their actions. It is able to reduce the effect of 100 – 200 ms delays that are common on the Internet, improving users' ability to perform tightly-coupled interaction. As a result, groupware can be more like face-to-face interaction, where people can interact in real-time and in close collaboration.

There are four secondary contributions of this thesis:

- I built a groupware networking library, GTC, that reduces the difficulty of creating distributed groupware.
- I showed that the action-feedback cycle is a way to model tightly-coupled interaction with or without network delay.
- I developed a tightly-coupled task that can be used for the future study of FFS and tightly-coupled behaviour.

- I confirmed previous studies, which show that network delay has a significant negative effect on tightly-coupled interaction.

8.1 Future Work

FFS demonstrates the benefit of considering the effect that network techniques have on real-time group activities, and opens the way for further analysis into how network techniques affect user performance. For example, the Timewarp technique allows groupware to correct for late information to maintain a consistent shared environment across all of the clients, but the sudden corrections to the shared environment could be confusing for the user and reduce their performance, especially in tightly-coupled tasks. The Local Lag technique introduces additional delay to clients to make all of them as slow as the slowest client, which synchronizes both the clients and the users, but the additional delay may reduce a group's performance more than it helps. These popular techniques were designed to maintain consistency and fairness between the clients, but the techniques were not designed with user performance or usability in mind. The results from my study show that further evaluation of networking techniques could improve our understanding of groupware usability, and contribute better methods of improving user performance.

This research also opens the way for analysis of how network techniques affect users' perception of the task, as well as their perception of their own performance. Many networking techniques change the state of the shared environment or the way that it is perceived, which as a result changes how the users interact with the environment. Different kinds of feedback, such as feedback from other users or feedback about the state of the task, changes people's perception of the task. The difference between FFS and immediately displayed feedback is how the users changed their behaviour as a result of the different timing of local feedback. If small changes to the local feedback can change people's performance to such a degree, then other networking techniques will probably change how users behave as well.

Another area of future work is finding the best way to transition between FFS and immediately displayed feedback. This is one of next steps in implementing FFS in real-world software, so that users can work on individual tasks without interference from

a technique meant for only tightly-coupled interaction, because adding feedback latency to users working alone would have negative effects. Possible examples of ways to implement transitions between FFS and immediately displayed feedback would be user-triggered mode switches, automatically detecting when actions affect other users, or gradual adaptation in relation to the proximity between users' avatars.

Since FFS is a technique that only affects information on the local client, it would be useful for large-scale applications, such as massively multiplayer online games. These types of games allow thousands of people to play together, often connecting the players through a centralized server. FFS could be used to allow tightly-coupled interaction between the players without increasing the required bandwidth of the application or the complexity of the server, because this technique only changes the user's own feedback on the local client.

There is a limit to the amount of delay that users can cope with. In the Fire Truck game, users complained when the network latency exceeded 400 – 500 ms of delay with both FFS and immediately displayed feedback. However, the complaints between the two conditions were different: with FFS the users felt like they had lost control, and with immediately displayed feedback they complained that they did not know what to do. It may be possible to mix the conditions together to raise the threshold of user adaptability by only applying a portion of the network latency as feedback latency (and therefore only a portion of FFS), or by limiting feedback latency to an amount less than 400 milliseconds.

Lastly, it would be valuable to know what specific tasks receive the most benefit from FFS, and how such tasks can be identified. Although I have provided a definition for tightly-coupled interaction, the degree to which tasks are considered tightly coupled is not well known. Future work should classify different kinds of tightly-coupled interaction, such that these types of tasks are better understood.

REFERENCES

Aggarwal, S., Banavar, H., Khandelwal, A., Mukherjee, S., and Rangarajan, S. 2004. Accuracy in dead-reckoning based distributed multi-player games. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support For Games (Portland, Oregon, USA, August 30 - 30, 2004). NetGames '04.* ACM Press, New York, NY, 161-165.

Azuma, R. and Bishop, G. 1995. A frequency-domain analysis of head-motion prediction. In *Proceedings of the 22nd Annual Conference on Computer Graphics and interactive Techniques* S. G. Mair and R. Cook, Eds. SIGGRAPH '95. ACM, New York, NY, 401-408.

Baecker, R. M. 1993. *Readings in Groupware and Computer-Supported Cooperative Work: Assisting Human-Human Collaboration*. 1st ed. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

Baecker, R. M., Grudin, J., Buxton, W. A. S., Greenberg, S. 1995. Readings in Human-Computer Interaction: Towards the Year 2000 (Second Edition) Morgan Kaufmann Publishers, Inc. San Francisco, CA, USA.

Beaudouin-Lafon, M. and Karsenty, A. 1992. Transparency and awareness in a real-time groupware system. In *Proceedings of the 5th Annual ACM Symposium on User interface Software and Technology* (Monteray, California, United States, November 15 - 18, 1992). UIST '92. ACM Press, New York, NY, 171-180.

Beigbeder, T., Coughlan, R., Lusher, C., Plunkett, J., Agu, E., and Claypool, M. 2004. The effects of loss and latency on user performance in unreal tournament 2003®. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support For Games* (Portland, Oregon, USA, August 30 - 30, 2004). NetGames '04. ACM Press, New York, NY, 144-151.

Brun, J., Safaei, F., and Boustead, P. 2006. Managing latency and fairness in networked games. *Commun. ACM* 49, 11, 46-51.

Card, S., Moran, T., and Newell, A. 1983. The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 1983.

Cronin, E., Filstrup, B., Kurc, A. R., and Jamin, S. 2002. An efficient synchronization mechanism for mirrored game architectures. In *Proceedings of the 1st Workshop on Network and System Support For Games* (Bruanschweig, Germany, April 16 - 17, 2002). NetGames '02. ACM, New York, NY, 67-73.

Crowley, T., Milazzo, P., Baker, E., Forsdick, H., and Tomlinson, R. 1990. MMConf: an infrastructure for building shared multimedia applications. In *Proceedings of the 1990*

ACM Conference on Computer-Supported Cooperative Work (Los Angeles, California, United States, October 07 - 10, 1990). CSCW '90. ACM Press, New York, NY, 329-342.

Dabrowski, J. R. and Munson, E. V. 2001. Is 100 Milliseconds Too Fast? In *CHI '01 Extended Abstracts on Human Factors in Computing Systems* (Seattle, Washington, March 31 - April 05, 2001). CHI '01. ACM, New York, NY, 317-318.

Daft, R. L. and Lengel, R. H. 1986. Organizational information requirements, media richness and structural design. *Manage. Sci.* 32, 5 (May. 1986), 554-571.

Dewan, P. and Choudhard, R. 1991. Flexible user interface coupling in a collaborative system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems: Reaching Through Technology* (New Orleans, Louisiana, United States, April 27 - May 02, 1991). S. P. Robertson, G. M. Olson, and J. S. Olson, Eds. CHI '91. ACM Press, New York, NY, 41-48.

Dewan, P. and Choudhary, R. 1991. Primitives for programming multi-user interfaces. In *Proceedings of the 4th Annual ACM Symposium on User interface Software and Technology* (Hilton Head, South Carolina, United States, November 11 - 13, 1991). UIST '91. ACM Press, New York, NY, 69-78.

Dick, M., Wellnitz, O., and Wolf, L. 2005. Analysis of factors affecting players' performance and perception in multiplayer games. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support For Game*s (Hawthorne, NY, October 10 - 11, 2005). NetGames '05. ACM Press, New York, NY, 1-7.

Dix, A., Finlay, J., Abowd, G., and Beale, R. 1998. Chapter 13: Groupware. *Human Computer Interaction,* 2nd Edition, Prentice Hall. 463-508.

Ellis, C. A., Gibbs, S. J., and Rein, G. 1991. Groupware: some issues and experiences. *Commun. ACM* 34, 1 (Jan. 1991), 39-58.

Funkhouser, T. A. 1995. RING: a client-server system for multi-user virtual environments. In *Proceedings of the 1995 Symposium on interactive 3D Graphics* (Monterey, California, United States, April 09 - 12, 1995). SI3D '95. ACM, New York, NY, 85-ff.

Fussell, S. R., Kraut, R. E., and Siegel, J. 2000. Coordination of communication: effects of shared visual context on collaborative work. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work* (Philadelphia, Pennsylvania, United States). CSCW '00. ACM Press, New York, NY, 21-30.

Gautier, L. 1999. End-to-end transmission control mechanisms for multiparty interactive applications on the Internet. *Proc. of IEEE Infocom 1999, volume 3.*

Gaver, W. 1996. Affordances for interaction: The social is material for design, *Ecological Psychology, vol. 8, no. 2*, 112-129.

Gergle, D., Kraut, R. E., and Fussell, S. R. 2006. The impact of delayed visual feedback on collaborative performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Montréal, Québec, Canada, April 22 - 27, 2006). R. Grinter, T. Rodden, P. Aoki, E. Cutrell, R. Jeffries, and G. Olson, Eds. CHI '06. ACM Press, New York, NY, 1303-1312.

Gutwin, C. 1997. *Workspace awareness in real-time distributed groupware.* PhD Thesis, Dept of Computer Science, University of Calgary, Alberta.

Gutwin, C. 2001. The Effects of Network Delay on Group Work in Shared Workspaces, *Proceedings of the European Conference on Computer-Supported Cooperative Work*, 299-318.

Gutwin, C., Benford, S., Dyck, J., Fraser, M., Vaghi, I., and Greenhalgh, C. 2004. Revealing Delay in Collaborative Environments, *Proceedings of the ACM Conference on Human Factors in Computing Systems*, 503-510.

Gutwin, C. and Greenberg, S. 1996. Workspace Awareness for Groupware. *Proceedings of the Conference on Human Factors in Computing Systems.* Vancouver, 208–209.

Hill, R.D., Brinck, T., Rohall, S.L., Patterson, J.F. and Wilner, W. 1994. "The Rendezvous Architecture and Language for Constructing Multi-User Applications." *ACM Transactions on Computer-Human Interaction,* 1(2), 81-125, June.

Jay, C., Glencross, M., and Hubbold, R. 2007. Modeling the effects of delayed haptic and visual feedback in a collaborative virtual environment. *ACM Trans. Comput.-Hum. Interact.* 14, 2 (Aug. 2007), 8.

J. S. Steinman, J. W. 1998. Scalable distributed military simulations using the SPEEDES object-oriented simulation framework. *Proc. of Object-Oriented Simulation Conference '98*, 3-23.

Kraut, R. E., Fussell, S. R., Siegel, J. 2003. Visual information as a conversational resource in collaborative physical tasks. *Human Computer Interaction*, vol. 18, 13-49.

Kraut, R. E., Gergle, D., Fussell, S. R. 2002. The use of visual information in shared visual spaces: informing the development of virtual co-presence, *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, New Orleans, Louisiana, USA.

Lambert, L. 1978. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7): 558-565.

Li, F. W., Li, L. W., and Lau, R. W. 2004. Supporting continuous consistency in multiplayer online games. In *Proceedings of the 12th Annual ACM international Conference on Multimedia* (New York, NY, USA, October 10 - 16, 2004). MULTIMEDIA '04. ACM, New York, NY, 388-391.

Luff P. K. & Heath, C. C. 1999. Surveying the Scene: The monitoring practices of staff control rooms. In *Proceedings of People in Control: An international conference on human interfaces in control rooms, cockpits and command centres*, Noyes, J. and Bransby, M. (eds.), University of Bath, UK, IEE Press. 1-6.

MacKenize, I. S., & Ware, C. 1993. Lag as a determinant of human performance in interactive systems. *Proceedings of the ACM Conference on Human Factors in Computing Systems - INTERCHI '93*. New York: ACM, 488-493.

McGrath, J. E. 1984. *Groups: Interaction and performance.* Englewood Cliffs, NJ: Prentice-Hall, Inc.

McGrath, J.E. 1991. "Time, Interaction, and Performance (TIP): A Theory of Groups," *Small Group Res.*, 147-174.

Malone, T.W., Crowston, K. 1990 What is coordination theory and how can it help design cooperative work systems? *Proc. CSCW'90*, ACM Press, 357–370.

Malone, T. W. and Crowston, K. 1994. The interdisciplinary study of coordination. *ACM Comput. Surv.* 26, 1 (Mar. 1994), 87-119.

Mauve, M. 2004. How to keep a dead man from shooting. *Proc. of the 7th International Workshop on Interactive Distributed Multimedia Systems*, 199-204.

Nova, N. 2002. Awareness Tools : Lessons from Quake-Like. In *Proceedings of "Playing with the Future Conference"*, Manchester, UK.

Nova N., Wehrle, T., Goslin, J., Bourquin, Y. & Dillenbourg, P. 2003. The Impacts of Awareness Tools on Mutual Modelling in a Collaborative Video-Game. In J. Favela and D. Decouchant (Eds.). *Proceedings of the 9th International Workshop on Groupware*, Autrans France, September 2003, 99-108.

Pantel, L. and Wolf, L. C. 2002. On the impact of delay on real-time multiplayer games. In *Proceedings of the 12th international Workshop on Network and Operating Systems Support For Digital Audio and Video* (Miami, Florida, USA, May 12 - 14, 2002). NOSSDAV '02. ACM Press, New York, NY, 23-29.

Pantel, L. and Wolf, L. C. 2002. On the suitability of dead reckoning schemes for games. In *Proceedings of the 1st Workshop on Network and System Support For Games* (Bruanschweig, Germany, April 16 - 17, 2002). NetGames '02. ACM Press, New York, NY, 79-84.

Patterson, J.F. 1991. "Comparing the Programming Demands of Single-User and Multi-User Applications." In *Proceedings of the UIST'92 Symposium on User Interface Software and Technology*, November 11-13, Hilton Head, South Carolina, ACM Press, 87-94.

Park, K. S. and Kenyon, R. V. 1999. Effects of Network Characteristics on Human Performance in a Collaborative Virtual Environment. In *Proceedings of the IEEE Virtual Reality* (March 13 - 17, 1999). VR. IEEE Computer Society, Washington, DC, 104.

Pinelle, D. 2003. A Groupware Design Framework for Loosely-Coupled Workgroups. *Proceedings of the European Conference on Computer-Supported Cooperative Work*.

Oliveira, Manuel. 2003. Perceptual Network Metaphors: Involving the User in the End-to-End Argument. Available online at http://www.cl.cam.ac.uk/~jac22/out/mo-d.pdf, 2007.

Olson, G.M., Olson, J.S. 2000. Distance matters. *Human-Computer Interaction*, 15(2-3), 139-178.

Olson, J. S. and Teasley, S. 1996. Groupware in the wild: lessons learned from a year of virtual collocation. In *Proceedings of the 1996 ACM Conference on Computer Supported Cooperative Work* (Boston, Massachusetts, United States, November 16 - 20, 1996). M. S. Ackerman, Ed. CSCW '96. ACM, New York, NY, 419-427.

Robert S. Allison, J. E. 2004. Effects of network delay on a collaborative motor task with telehaptic and televisual feedback. *Proceedings of the 2004 ACM SIGGRAPH international Conference on Virtual Reality Continuum and Its Applications in industry.* ACM Press, New York, NY, 375-381.

Salvador, T., Scholtz, J., and Larson, J. 1996. The Denver model for groupware design. *SIGCHI Bull.* 28, 1 (Jan. 1996), 52-58.

Scott, W.R. 1987. Organizations: rational, natural, and open systems, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ.

Segal, L. D. 1994. Effects of Checklist Interface on Non-verbal Crew Communications. Western Aerospace Laboratories, Inc. 1611 Mays Avenue, Monte Serrano, CA, 17-18.

Sheldon, N., Girard, E., Borg, S., Claypool, M., and Agu, E. 2003. The effect of latency on user performance in Warcraft III. In *Proceedings of the 2nd Workshop on Network and System Support For Games* (Redwood City, California, May 22 - 23, 2003). NetGames '03. ACM Press, New York, NY, 3-14.

Shneiderman, B. and Plaisant, C. 2004. Designing the User Interface: Strategies for Effective Human-Computer Interaction: Fourth Edition, Addison-Wesley Publ. Co., Reading, MA.

Smed, J., Niinisalo, H., and Hakonen, H. 2005. Realizing the bullet time effect in multiplayer games with local perception filters. *Comput. Networks* 49, 1 (Sep. 2005), 27-37.

Sohlenkamp, M. and Chwelos, G. 1994. Integrating communication, cooperation, and awareness: the DIVA virtual office environment. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work* (Chapel Hill, North Carolina, United States, October 22 - 26, 1994). CSCW '94. ACM Press, New York, NY, 331-343.

Stefik, M., Foster, G., Bobrow, D. G., Kahn, K., Lanning, S., and Suchman, L. 1987. Beyond the chalkboard: computer support for collaboration and problem solving in meetings. *Commun. ACM* 30, 1 (Jan. 1987), 32-47

Tang, J. C. 1991. Findings from observational studies of collaborative work. *International Journal of ManMachine Studies*, vol. 34, 143-160.

Thompson, J.D. 1967. Organizations in action. McGraw-Hill, New York.

Quax, P., Monsieurs, P., Lamotte, W., De Vleeschauwer, D., and Degrande, N. 2004. Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game. In *Proceedings of 3rd ACM SIGCOMM Workshop on Network and System Support For Games* (Portland, Oregon, USA, August 30 - 30, 2004). NetGames '04. ACM Press, New York, NY, 152-156.

Vaghi, I., Greenhalgh, C., and Benford, S. 1999. Coping with inconsistency due to network delays in collaborative virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology* (London, United Kingdom, December 20 - 22, 1999). VRST '99. ACM Press, New York, NY, 42-49.

Whittaker, S., & O'Conaill, B. 1997. The role of vision in face-to-face and mediated communication. In *K. Finn, A.Sellen & S. Wilbur (Eds.) Video-Mediated Communication*. Mahwah, NJ: Erlbaum, 23-49.

Yutaka Ishibashi, M. N. 2006. Subjective Assessment of Fairness among Users in Multipoint Communications. *Proceedings of the 2006 ACM SIGCHI international Conference on Advances in Computer Entertainment Technology*. Hollywood, California: ACM Press, 69.

Znati, T., Simon, R., Field, B. 1995. A network-based scheme for synchronization of multimedia streams. In *Proc. Internat. Workshop Multimedia Synchronization*, Tysons Corner, VA, 12-20.

APPENDIX

EXPERIMENTAL MATERIALS

Post Experiment Questionnaire (Exploratory Study / Spacewar Duo)

1. Did you notice any latency between you and the gunner? Circle your answer.

   Yes No

2. If yes, rate how badly you think it affected your performance as a team. Place an X beside your answer.

   _____ None
   _____ Slightly harder, but coping was easy
   _____ Noticeably harder, and coping for the delay was a challenge
   _____ Significantly harder, and coping for the delay was near impossible
   _____ The task was altogether impossible

Explain the reasons for your ranking.

3. If yes, rate how badly you think it affected your performance as an individual. Place an X beside your answer.

   _____ None
   _____ Slightly harder, but coping was easy
   _____ Noticeably harder, and coping for the delay was a challenge
   _____ Significantly harder, and coping for the delay was near impossible
   _____ The task was altogether impossible

Explain the reasons for your ranking.

Consent Form (Spacewar Duo)

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF SASKATCHEWAN
INFORMED CONSENT FORM

Research Project: The effect of network delay on collaboration
Investigators: *Carl Gutwin*, Department of Computer Science (966-8646)
Dane Stuckel, Department of Computer Science (966-2327)

This consent form, a copy of which has been given to you, is only part of the process of informed consent. It should give you the basic idea of what the research is about and what your participation will involve. If you would like more detail about something mentioned here, or information not included here, please ask. Please take the time to read this form carefully and to understand any accompanying information.

In this study you will be asked to carry out an activity in a multiplayer game. You will either be asked to be the pilot or the gunner of a spaceship, and your task will be to work together with your partner to shoot all of the targets in a maze as fast as possible. At the end of the study, we will ask you to provide us with feedback on the techniques.

The session will take up to 60 minutes to complete. You will receive a $10.00 payment for your participation.

At the end of the session, you will be given more information about the purpose and goals of the study, and there will be time for you to ask questions about the research.

The data collected from this study will be used in articles for publication in journals and conference proceedings.

As one way of thanking you for your time, we will be pleased to make available to you a summary of the results of this study once they have been compiled (they will be made available on the HCI web site, hci.usask.ca). This summary will outline the research and discuss our findings and recommendations.

All of the information we collect from you (data logged by the computer, observations made by the experimenters, and your questionnaire responses) will be stored so that your name is not associated with it (using an arbitrary participant number). Any write-ups of the data will not include any information that can be linked directly to you. The research materials will be stored with complete security throughout the entire investigation. Do you have any questions about this aspect of the study?

You are free to withdraw from the study at any time without penalty and without losing any advertised benefits. Withdrawal from the study will not affect your academic status or your access to services at the university. If you withdraw, your data will be deleted from the study and destroyed. In addition, you are free to not answer specific items or questions on questionnaires.

Your continued participation should be as informed as your initial consent, so you should feel free to ask for clarification or new information throughout your participation. If you have further questions concerning matters related to this research, please contact:

Your signature on this form indicates that you have understood to your satisfaction the information regarding participation in the research project and agree to participate as a participant. In no way does this waive your legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. If you have further questions about this study or your rights as a participant, please contact:

- Dr. Carl Gutwin, Associate Professor Dept. Computer Science (306) 966-8646 gutwin@cs.usask.ca
- Office of Research Services University of Saskatchewan (306) 966-4053

Participant's signature: _____
Date: _____
Investigator's signature: _____
Date: _____

A copy of this consent form has been given to you to keep for your records and reference. This research has the ethical approval of the Office of Research Services at the University of Saskatchewan.

Post Experiment Questionnaire (Formal Experiment / Fire Truck)

1. Personal Information:
   Gender □ Male □ Female

   Age _____

   University major or occupation_____

2. How many hours a week, on average, do you spend working with computers?

   □ 0-4 □ 4-12 □ 12+

3. Have you ever played a multiplayer computer or video game before?

   □ Yes □ No

If yes, describe the ones that you have used and how much experience you have with using each.

4. Have you ever had to synchronize closely with others in activities before?

   □ Yes □ No

If yes, name the activities and how closely you had to coordinate with others in each.

5. If any of your work and/or school activities require significant hand-eye coordination, please list them here (e.g. using tools, manipulating small objects). Next to each entry, please estimate how much time you spend on the activity per week.

6. If any of your leisure activities require significant hand-eye coordination, please list them here (e.g. computer games, sports, knitting, painting, model building, etc.). Next to each entry, please estimate how much time you spend on the activity per week.

Consent Form (Fire Truck Game)

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF SASKATCHEWAN
INFORMED CONSENT FORM

Research Project: The effect of network delay on collaboration
Investigators: *Carl Gutwin*, Department of Computer Science (966-8646)
Dane Stuckel, Department of Computer Science (966-2327)

This consent form, a copy of which has been given to you, is only part of the process of informed consent. It should give you the basic idea of what the research is about and what your participation will involve. If you would like more detail about something mentioned here, or information not included here, please ask. Please take the time to read this form carefully and to understand any accompanying information.

In this study you will be asked to carry out an activity in a multiplayer game. You will either be asked to control the driver or the turret of a fire truck, and your task will be to work together with your partner to keep a small fire drenched in water for as long as possible. At the end of the study, we will ask you to provide us with feedback on the system.

The session will take up to 60 minutes to complete. You will receive five dollars for every half hour of the study that you complete. At the end of the session, you will be given more information about the purpose and goals of the study, and there will be time for you to ask questions about the research.

The data collected from this study will be used in articles for publication in journals and conference proceedings.

As one way of thanking you for your time, we will be pleased to make available to you a summary of the results of this study once they have been compiled (they will be made available on the HCI web site, hci.usask.ca). This summary will outline the research and discuss our findings and recommendations.

All of the information we collect from you (data logged by the computer, observations made by the experimenters, and your questionnaire responses) will be stored so that your name is not associated with it (using an arbitrary participant number). Any write-ups of the data will not include any information that can be linked directly to you. The research materials will be stored with complete security throughout the entire investigation. Do you have any questions about this aspect of the study?

You are free to withdraw from the study at any time without penalty and without losing any advertised benefits. Withdrawal from the study will not affect your academic status or your access to services at the university. If you withdraw, your data will be deleted from the study and destroyed. In addition, you are free to not answer specific items or questions on questionnaires.

Your continued participation should be as informed as your initial consent, so you should feel free to ask for clarification or new information throughout your participation. If you have further questions concerning matters related to this research, please contact:

Your signature on this form indicates that you have understood to your satisfaction the information regarding participation in the research project and agree to participate as a participant. In no way does this waive your legal rights nor release the investigators, sponsors, or involved institutions from their legal and professional responsibilities. If you have further questions about this study or your rights as a participant, please contact:

- Dr. Carl Gutwin, Professor Dept. Computer Science (306) 966-8646 gutwin@cs.usask.ca
- Office of Research Services University of Saskatchewan (306) 966-4053

Participant's signature: _____
Date: _____
Investigator's signature: _____
Date: _____

A copy of this consent form has been given to you to keep for your records and reference. This research has the ethical approval of the Office of Research Services at the University of Saskatchewan.

Post Experiment Questionnaire (Formal Experiment / Fire Truck)

1. Personal Information:
   Gender □ Male □ Female

   Age _____

   University major or occupation_____

2. How many hours a week, on average, do you spend working with computers?

   □ 0-4 □ 4-12 □ 12+

3. Have you ever played a multiplayer computer or video game before?

   □ Yes □ No

If yes, describe the ones that you have used and how much experience you have with using each.

4. Have you ever had to synchronize closely with others in activities before?

   □ Yes □ No

If yes, name the activities and how closely you had to coordinate with others in each.

5. If any of your work and/or school activities require significant hand-eye coordination, please list them here (e.g. using tools, manipulating small objects). Next to each entry, please estimate how much time you spend on the activity per week.

6. If any of your leisure activities require significant hand-eye coordination, please list them here (e.g. computer games, sports, knitting, painting, model building, etc.). Next to each entry, please estimate how much time you spend on the activity per week.

Multiplayer Collaboration Workload Assessment (NASA TLX)

Task 1

Mental Demand

Low **High**

Physical Demand

Low **High**

Temporal Demand

Low **High**

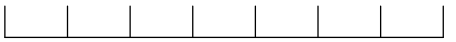
Effort

Low **High**

Performance

Good **Poor**

Frustration



Low **High**

Task 2

Mental Demand

Low **High**

Physical Demand

Low **High**

Temporal Demand

Low **High**

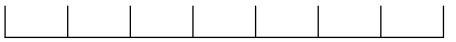
Effort

Low **High**

Performance

Good **Poor**

Frustration



Low **High**

Task 3

Mental Demand

Low **High**

Physical Demand

Low **High**

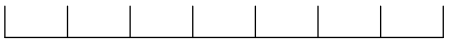
Temporal Demand

Low **High**

Effort

Low **High**

Performance



Good **Poor**

Frustration

Low **High**

Task 4

Mental Demand

Low **High**

Physical Demand

Low **High**

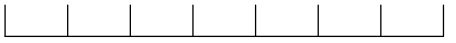
Temporal Demand

Low **High**

Effort

Low **High**

Performance



Good **Poor**

Frustration

Low **High**

GLOSSARY

Feedback is the visual information displayed to the local user as a result of their actions – for example, as a person controls an avatar, feedback is the corresponding movement of the avatar on screen.

Feedthrough is the visual information displayed to the local user as a result of actions by remote users, which means that the information must travel across the network before it can be displayed.

Feedback-Feedthrough Synchronization (FFS) is a display scheme where feedback from local actions is held for a length of time equal to the current estimated amount of delay on the network, such that an action seems to occur at approximately the same time on all of the clients involved in tightly-coupled interaction.

Immediately Displayed Feedback is a display scheme where feedback from local actions is displayed immediately and feedthrough is displayed as it is received from the network. This means that actions that occur at the same time on remote clients are displayed shortly after local events, which disrupts tightly-coupled work.

Transmission Control Protocol (TCP) is a reliable protocol for transmitting data over a packet-based network, but it is slower than UDP. TCP connections retransmit lost data, and limit the amount of information sent on the network in order to share a network's bandwidth fairly between clients.

User Datagram Protocol (UDP) is an unreliable protocol for transmitting data over a packet-based network. It does not retransmit lost data, but it is popular for streaming or real-time media.