HONOURS PROJECT REPORT

AfriMeet: An Internet meeting tool designed for low bandwidth and unstable network conditions

Flora Kundaeli

Supervised by: Dr. Hussein Suleman

	Category	Min	Max	Chosen
1	Requirement Analysis and Design	0	20	
2	Theoretical Analysis	0	25	
3	Experiment Design and Execution	0	20	15
4	System Development and Implementation	0	15	10
5	Results, Findings and Conclusion	10	20	15
6	Aim Formulation and Background Work	10	15	15
7	Quality of Report Writing and Presentation	10		10
8	Adherence to Project Proposal and Quality of Deliverables	10		10
9	Overall General Project Evaluation	0	10	5
Total marks		**80**		**80**

Department of Computer Science

University of Cape Town

2011

Abstract

Video-conferencing has become an important tool in the business world for communication and aiding in the decision making process. It eliminates the need for costly travel and time wasted preparing for face-to-face meetings. However, with the tremendous growth and adaptation of this technology, some areas are unable to make use of it due to its high bandwidth requirements. This report investigates techniques that can be used to reduce the high bandwidth requirements of video-conferencing systems resulting in the design and implementation of such a tool. Once the development process of the tool was complete, it was then tested and evaluated to determine if it works efficiently with low bandwidth. It was found that the bandwidth consumed by handraising was negligible compared to that consumed by screen sharing alone. Therefore such features ought to be given priority over the more bandwidth consuming features such as screen sharing or video. It was also found that for the screen sharing applications in video-conferencing systems, it is more advisable to transmit only the changes in a screen image to other participants as opposed to sending entire images. This differencing approach has been shown to greatly reduce the amount of bandwidth levels required to sufficiently support screen sharing.

Contents

Contents

Introduction 1
1.1 Motivation 1
1.2 Research questions 1
1.3 Work allocation 2
1.4 Ethical, professional and legal considerations 3
1.5 Summary outline of report 3
Background 4
2.2 Participant list 4
. 2.2.1 Participant list 4
2.2.2 Floor control 4
2.2.3 Hand raising 5
2.3 Screen sharing 6
2.4 Chat and Video 7
2.4.1 Chat 7
2.4.2 Presentations 7
2.4.3. Audio 8
2.4.4 Audio & Video 8
2.4.5 Video 8
2.5 Discussion 9
Design and implementation 11
3.1 High-Level Overview of System: Front End 11
3.2 Low-Level Implementation: Back End 13
3.2.1 Technology and tools 13
3.2.2 The HTTP Protocol 14
3.2.3 Participant List 16
3.2.3 Meeting Options 19
3.2.4 Hand Raising and Status Statistics updates 22
3.2.5 Floor control 23
3.2.6 Polling 25

3.2.7 Screen sharing 27
3.2.8 Exit application 31
3.3 Iterations 31
3.3.1 Iteration 1 31
3.3.2 Iteration 2 32
3.3.3 Iteration 3 34
Experimentation and Evaluation 35
4.1 Introduction 35
4.2 User Testing 35
4.2.1 Methodology 35
4.2.2 Experiment set-up 36
4.2.3 Participants 37
4.2.4 Results 37
4.2.5 Discussion of results 37
4.3 Performance Evaluations 39
4.3.1 Experiment 1: Bandwidth required for desktop sharing only 39
4.3.2 Experiment 2: Bandwidth required for handraisng 42
4.3.3 Experiment 3: Bandwidth required for a normal meeting 45
4.3.4 Experiment 4: Bandwidth required in an extremely active meeting (Stress Test) 48
4.3.5 Experiment 5: Bandwidth required without image differencing 52
4.4 Summary 54
Future Work and Conclusion 55
5.1 Future Work 55
5.2 Conclusion 55
Participant Consent Form 57
Experiment Questionnaire 58
References 61

List of Figures

Figure 1: Work allocation Colour is used to illustrate each team members section. The sizes of the blocks do not have any relationship to the amount of work in each section. Each section weighed the same and had the same level of complexity or difficulty. 2
Figure 2: Meeting statistics displayed at the bottom of the window 12
Figure 3: Breakdown of the desired meeting features 13
Figure 4: client server architecture 14
Figure 5: URL Encodings examples for different strings 15
Figure 6: Overview of the flow of data in system 16
Figure 7: Participant list 17
Figure 8: Data structures in the server 18
Figure 9: Creating a new meeting 20
Figure 10: Errors when creating a meeting 21
Figure 11:Snapshots of entering or changing meeting rooms 22
Figure 12: Screenshots of handraising 23
Figure 13: Changing the presenter 24
Figure 14: Screenshots of the polling feature 26
Figure 15: Flow of data during screen sharing 28
Figure 16: Computing image differences using ImageMagick The two resultant image differences appearing as black in the above diagrams are much smaller in size then the entire image (in this case B). Only image A is sent across and image B is produced through the image reconstructions. 30
Figure 17: Viewing the presenters desktop 31
Figure 18: Initial screen image as viewed from the receiver side 32
Figure 19: What the integrated system initially looked like 33
Figure 20: Improvement of previous version with the features shifted onto the above menu 34
Figure 21: Time delays in image transmission 38
Figure 22: Data transfer for the server and Peter during screen sharing 40
Figure 23: Graphs of data transfer for James and John during screen sharing 40
Figure 24: Distribution of data from the 5 screen sharing experiments combined 42
Figure 25: Total spread of dataflow at server 43
Figure 26: Bandwidth usage without screen sharing 45
Figure 27: Screen sharing tested with various applications 46
Figure 28: Screen sharing applications combined 47
Figure 29: Bandwidth required by normal meeting 48
Figure 30: Stress test for extremely active meeting 51
Figure 31: Bandwidth consumed by screen sharing without the use of image differencing 53

Chapter 1

Introduction

Business people, employers and employees alike spend approximately 20-30% of their time at work participating in meetings (Post et al., 2007). As a result, meetings have become the focus of multidisciplinary research leading to the introduction of asynchronous meeting technologies (Reidsma et al., 2007). These technologies enable individuals from different locations to participate in and contribute to meetings. One such example of a distributed meeting technology is video-conferencing, which offers the prospect of increasing participation and expertise in the making of critical decisions, reducing transportation costs, and increasing the range of tasks that employees can accomplish without physically traveling to a central office (Emanuel et al., 1995). In a typical video-conference, a camera is mounted over a computer screen and an image of the remote person(s) with whom one is communicating is displayed on an on-screen window (Monk & Watts, 1995). Communication can then take place through various forms such as audio-video, video-chat or audio-video-chat.

Most existing video-conferencing tools, however, have shortcomings and limitations. Major well known concerns involve dealing with unstable connections, low bandwidth and scalability issues. These conditions can greatly jeopardize the overall communication process, resulting in dysfunctional meetings. Without good video-conferencing tools and effective meeting discipline, Web conferencing can become annoying, redundant, unsatisfactory and waste people's time (Austin et al., 2006).

1.1 Motivation

There are multiple non-commercial and commercial video-conferencing systems that exist today, however most are designed for broadband networks and therefore do not work efficiently with low bandwidth (Cohen & Wang, 2009). Due to the high bandwidth requirements of most video-conferencing applications, when used in areas like South Africa and other developing countries where low bandwidth and unstable networks are still a problem, the product becomes unpredictable and its usability drastically drops (Egido, 1998, Causey et al., 2006). This ends up causing a negative impact on the user's experience and overall judgment of the software. As a result, countries and areas with similar network and bandwidth challenges as the ones mentioned above, are unable to use these systems, and the opportunity of using virtual meetings as a way to collaborate is ruled out.

Therefore the focus of this project is on creating a video-conferencing tool that will be able to offer the same features as other online meeting tools but that can operate with low bandwidth. The system should be able to adapt smoothly to changes in bandwidth or network conditions without causing too many distractions to the users and affecting their experience with the product. In addition, the system should be simple to set up and not require the user to install additional plugins or software.

1.2 Research questions

The main research questions tackled by this project as a whole are as follows:

- Is it possible to build an effective audio-conferencing tool that works with low bandwidth conditions?

- Is it possible to build an effective video-conferencing tool that works with low bandwidth conditions?
- Is it possible to build an effective text chat tool that can work with minimal bandwidth?
- Is the pre-loading of static data feasible with low bandwidth?
- Is it possible to build a screen sharing tool that works with low bandwidth?
- Is it possible to construct a system that manages meeting procedures (floor control, hand-raising etc.) efficiently with low bandwidth?

The last two research questions are investigated in this report. In order to answer these questions, a system was built and tested with users under this constraint. The resulting system should function reliably and give a good end user experience despite bandwidth levels. The success of the product will greatly benefit organizations in developing countries where unstable and low bandwidth connections prohibit them from participating in online meetings. In addition, various results from experimentation run during the project could be helpful for future research on development of Internet based solutions for low bandwidth and unstable Internet conditions.

1.3 Work allocation

The project was divided into three equal and distinct stand-alone sub components that would make up the whole video-conferencing system once merged. The diagram below illustrates the work allocation.

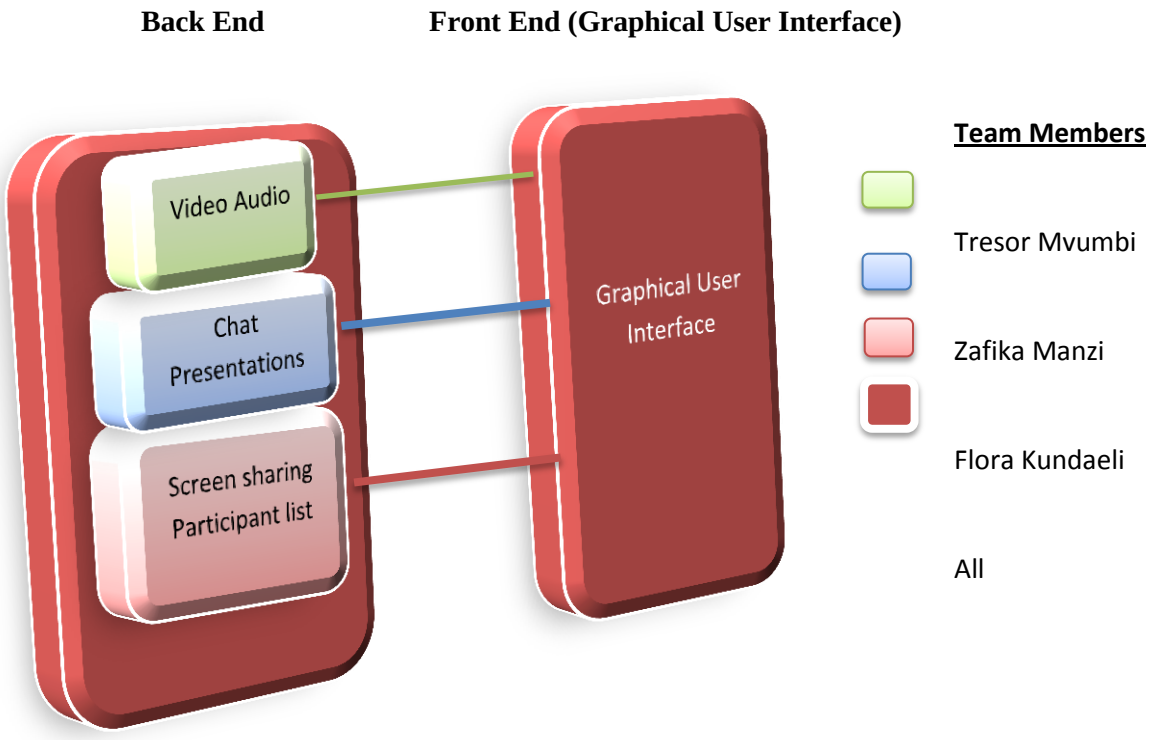


Figure 1: Work allocation

Colour is used to illustrate each team members section. The sizes of the blocks do not have any relationship to the amount of work in each section. Each section weighed the same and had the same level of complexity or difficulty.

The team members each worked on their individual sections of the system. Each back-end section (components) had its own front-end user interface. The three sections would later be integrated to form the overall system with the Graphical user interface being designed collaboratively.

The initial intention was to implement a framework for hosting components; however, this changed later on in the project and was instead replaced by the screen sharing application. This document will focus on the development of the screen sharing application, participant list, floor control, hand raising mechanism and the polling feature of the online meeting tool. The system was designed in java using a client-server approach. This is described in detail in Chapter 3.

1.4 Ethical, professional and legal considerations

Ethics clearance from the university was obtained as user testing was conducted to help in the evaluation process. There are no anticipated risks associated with the experiments. Information was provided to all test participants regarding the test procedure and confidential nature of the data collected. In addition, users were required to sign a consent form before being allowed to participate in the experiment. The system was developed using free and open source software tools, development platforms and third party libraries.

1.5 Summary outline of report

Chapter 2 covers general background information on low bandwidth video-conferencing systems. The design and implementation process of the software tool is described in Chapter 3. Chapter 4 investigates the results obtained from user testing and other evaluations. The overall summary of findings, suggested future work and conclusion are discussed in chapter 5.

Chapter 2

Background

This chapter explores the various techniques that have been adopted by previous video-conferencing systems in an effort to overcome the challenges brought about by low bandwidth problems. Not much bandwidth related work was found on the participant list, floor control, hand raising, polling and chat features. This could be because these features were not considered to be much of a bandwidth threat as they consumed insignificant amounts of bandwidth. A discussion of each feature is given below. In addition, presentations, screen sharing, audio and video in relation to low bandwidth are also discussed.

2.2 Participant list

.

2.2.1 Participant list

The participant list is the list of all users in a meeting, identified by their usernames. Lists may vary from system to system and can be extended to contain additional data besides just usernames. Lists should be flexible and enable the filtering of any changes in information associated with users on the list (Koskelainen et al., 2002). In such cases, users are then able to select and modify the list according to its appearance and the level of detail they prefer. For example, some users may want to know the speaker or pay less attention to certain speakers and can therefore adjust an attention knob that will filter those speakers' words. In turn the speaker also gets feedback on the level of attention he has from attendees (Greenberg, 1989).

2.2.2 Floor control

Floor is the permission to access or manipulate a specific shared resource or a set of resources temporarily. Floor-control refers to the mechanism that enables applications or users to gain safe and mutually exclusive or non-exclusive input access to the shared object or resource (Koskelainen et al., 2006). The provision of mutually exclusive access to shared resources enables the smooth management of meetings, which results in clear meeting procedures being followed. In most meetings, the host has special privileges over users. In most cases, the user only has control over the information they would like to view on their computers. For example, some users may not care about the level of detail in a participant list and may prefer to just see notifications of when people enter or leave the meeting. Therefore they can control this feature if the participant list enables such (Petri Koskelainen et al., 2002).

There are many kinds of meeting control methodologies. One of the simplest is an "open-voice" conference where there is no control and anyone can freely join the meeting and speak at any time (Koskelainen et al., 2002). In a controlled meeting, the moderator can speak at any time whereas participants need to explicitly request the floor from the moderator(Greenberg, 1989). The simplest illustration of Floor control is the ability to control who has the right to speak. Basic control starts from the host when they create a meeting and are allocated rights to modify or terminate that meeting. More control involves granting or denying people permission to join that meeting, or to speak during the meeting. Information about permissions can be kept in a list associating each user to the privileges

allocated to them. In Chan's (2004) turn-taking protocol, a user could be in three states: either in control of the shared resources or application; waiting for control; or observing their collaborator's activities and actions. A user in control has privileges over the resource and could release it. Once released, a resource is passed on to the first user (users were queued) who had requested it, who then takes control over the shared resource.

A meeting can have a number of floors, each associated with certain resources and each having its own queue (for example, there could be a floor for the video stream, and another floor for screen sharing etc.). Each queue can either be managed automatically on a first-come first-served basis, round robin basis, higher priority basis or by a mediator who can manipulate queue entries. The problem with automatic queuing is the risk of running into starvation when a user holds the floor indefinitely (Koskelainen et al., 2002). A solution could be to use time limits to prevent indefinite blocking. The mediator approach is easy to implement, however problems arise if the mediator gets disconnected unexpectedly. Systems should have mechanisms that enable recovery from moderator failures.

Although floor control protocols generally consume less bandwidth than media streams, precaution has to be taken when dealing with larger meetings (Koskelainen et al., 2002). This is because in such cases the general conference information might be large and may encounter transmission problems when being sent to users with low bandwidth. Therefore, updates in information could be delivered in increments as opposed to sending whole descriptions.

Garcia-Luna-Aceves et al. (2005) mentioned two kinds of floor control; the receiver based floor control and sender based floor control. The receiver based approach is useful when there is a surplus of bandwidth available. In this case all the meeting participants continuously send both the audio and video streams regardless of who has the floor. This method however, saturates the end receiver as they receive many packets and have to filter them. The sender based floor control approach permits only the speaker to send their streams. This approach overcomes the problem of saturating the end receiver but runs the risk of saturating the network. How?

2.2.3 Hand raising

Electronic hand raising is the ability for one to signal when they want to speak. This is simple in face to face meetings as people can physically raise their hands or the speaker can easily pick up signs from audiences that want to speak, or the audiences can specifically sense when it was appropriate to interject and ask a question. Isaacs & Tang, (1994) stated that it was impossible to direct attention toward a specific person in a multi-way conference. Everyone sees you through the same camera, so if you are looking at one person's video image, it appears to everyone as if you are looking at all of them. They observed that people tend to use one another's names to address one another in such situations. To add onto this, the sharing of a single audio channel limited the ability of people to conduct side conversations, and pointing did not work either as it was difficult for the others to use spatial position to figure out who was being addressed. Pointing only worked when one wanted to focus attention on certain parts of their own environment. Therefore to avoid such issues, a good hand raising or signaling mechanism is needed in order to effectively use bandwidth and enable satisfactory meetings.

Malpani (1997) introduced a tool called the question board as a floor control mechanism that simplified and aided the asking of questions in an online meeting. It was stated that fewer questions were asked by remote participants as opposed to local audiences. Suggested reasons for this were the lack of floor control tools, or the weight of having to set up microphones or cameras beforehand, in order to enable the

proper transmission of their questions. The question board enabled users to signal when they had a question by entering a text message into the question board's interface as the meeting progressed. The interface enabled users to select whether the question should be treated as private, anonymous or public. They could also select the medium in which they wanted to ask their question, whether as text, audio, video or as a combination of different mediums. This would reflect on the moderator's panel, who could then identify the person that requested to speak and address the question by attracting the speakers attention. The moderator could then respond by either reading the question out aloud or by temporarily granting the floor to the user. Despite having the question board in meetings, it was found that some users were still reluctant to ask questions. The reason could have been that users felt that they did not have adequate feedback to detect when it was appropriate to ask a question. It is therefore important to stimulate environments that are favorable for asking questions when conduting video-conferencing. It would also help if the speakers openly invited questions from remote users.

Another version of handraising involves the use of interrupts. A user can force everyone to pay attention to them by "shouting". The user creates the text message and sends it. The message appears on the receiver's side in a separate window above other windows and it cannot be hidden, filtered or disabled (Greenberg, 1989).

Hand raising is not only limited to when one wants to speak, but can also be extended to convey other emotions such as "laugh" and "applause" during meetings. These are not necessarily requests for the floor, but are designed in order to track users' emotions and opinions as the meeting progresses. Such hand raising mechanisms involve the use of icons and colours to identify the different states of a user. The moderator can then quickly notice the active users and distinguish between those that have questions and those that do not (ie those just portraying emotions). For the sake of keeping other users aware of the activities taking place, a list of all users with their state representations is displayed (Causey et al., 2006). However, unlike face to face meetings where people can easily pick up signs from audiences or instructors, it is still difficult to provide such discernments in video-conferencing meetings.

2.3 Screen sharing

Screen sharing is the process of broadcasting the content of one's screen to others. This is useful when one wants to portray an idea, demonstrate a new product or show a website during a meeting. Normally the entire screen is shared. However, it is possible to share certain portions of the screen as opposed to the entire desktop, as some information on the desktop may be private and not intended for viewing by other people.

Most screen sharing applications do not work well with low bandwidth as the presenter's screen image has to be streamed across the network (Lieb & Benton, 2010). This often causes slow updates and freezing of the images on the receiver's side. As a result, screen sharing is not the best method to use in the case of wanting to share video or fast changing images. Screen sharing should not be used as a substitute for presentations in a meeting but rather to complement existing presentation features. This is because software solutions that support screen sharing alone do not efficiently meet all the user's needs (Lieb & Benton, 2010). For example, when an idea has not been fully understood, then one can share their screen to further demonstrate that idea. This does not mean that you can carry out an entire meeting presentation through screen sharing your own PowerPoint slides to the other users as you read along as this is bandwidth consuming.

Hansen (2006) implemented a screen sharing system whose abilities were based on a screen sharing condition. The remote device would send an "indication" of the screen sharing condition, the intended receiver would send its data and a screen sharing session between the remote and receiving device would

be established. Session keys were used between both devices in order to create and provide access to the session. Communication between devices took place using HTTP POST and GET commands over TCP/IP and file transfer was done using FTP (File Transfer Protocol).The system was designed such that it could also work over the public networks, as most of the current screen sharing products work over local area networks (LANs) and were not functional over the Internet.

Lieb (2010) also designed a remote screen sharing application for Web browsers only. This was in order to address limitations of existing commercial desktop sharing solutions such as WebEx™ and GoToMyPC™ which have time-consuming set-up processes. In addition, they require the installation of special software executables and plugins before one can actually use the system. An applet would run on the client side capturing screen images and pointer positions and sending them. The application could automatically opt between transmitting an entire screen image or sending only the parts of the screen that had changed. The feature also supported the sharing of portions of the screen. In order for the applet to detect the section of the screen one desired to share, identifying images were placed at the corners of the "display area". The applet would look for these images on the window and determine the position to capture. The HTTP Multi-part POST Web encoding standard was used in order to bypass firewalls and other security systems. These images were then encoded using standard encodings such as GIF, PNG and JPEG and transmitted to the server. The advantage of this approach was that viewers were able to carry out browser screen sharing without having to install additional software.

2.4 Chat and Video

Although this report focuses specifically on the participant list and screen sharing features of video conferencing systems, research on the other features was done with more emphasis on video in order to better understand the entire system. The features affecting video would most likely affect screen sharing too and therefore some background work on video would be helpful.

2.4.1 Chat

Chat is one of the least bandwidth-consuming features found in most video-conferencing systems. It enables communication via the transmission of short text messages. An experiment by Scholl et al. (2005) made use of text chat to complement the video (video-chat) as opposed to the common audio-video version of communication. The video would display an image of the current chatter (video follows chat). Their findings revealed that most of the users found the application useful. However it would be more beneficial to incorporate video, audio and text chat in designing efficient adaptive video-conferencing systems to operate in low bandwidths conditions, so that when one feature is unavailable due to the low bandwidth, another feature can be used to balance and continue communication.

2.4.2 Presentations

Real face to face meetings often involve using physical work-like materials such as documents, plans, notes, brainstorming tools, etc. These facilitate the creation of workstations that enable participants to record or express their ideas as the meeting progresses (Greenberg, 1989). Most video-conferencing systems support such features by enabling people to share their work, such as PowerPoint slide presentations. These can be initially uploaded at each participant site and viewed simultaneously as the speaker presents. Isaacs & Tang (1994) however, indicated that, in face-to- face encounters, participants were able to more tightly coordinate their utterances, which improved their ability to reach mutual

understanding faster than in video-conferencing. They stated that most video interactions do not allow participants to build on each other's work, and manipulate real-world objects, nor do they allow users to look over each other's shoulders to gain another perspective.

2.4.3. Audio

Audio is the most common form of everyday face to face communication. It also plays a crucial role in online meetings. Many factors such as packet loss, jitter, echo, choice of codec etc. affect the quality of the audio received (Goode, 2002). Most VoIP systems use RTP/UDP/IP for communication. Unlike TCP/IP which offers the reliable transmission of packets, UDP does not cater for packet loss or any delays encountered. Although the bandwidth generally required for voice is small, lower bandwidths may increase these problems, resulting in a variety of audio distortions at the receiver's side. Packet loss and delays cause skips in the audio resulting in broken conversations. Parts of the encoded audio may be deleted at low bitrates while coding high frequency content under low bandwidth situations (Jumisko-Pyykkö, 2006). Having a buffer on the receivers end helps to compensate for these delays.

2.4.4 Audio & Video

In most cases, it has been found that it was best to combine audio with video. In support of this, Isaacs et al. (1994) found that, compared with audio only, a video channel improved the ability to cooperate, express oneself and communicate more easily. They mentioned that the advantages of video depended critically on the nearly-instantaneous transmission of audio, even if it meant getting out of sync with the video image. On the other hand, when compared with face-to-face, it can be difficult in video interactions to control the floor, notice peripheral cues, easily point things out and manipulate real-world objects. They further suggested that, in order to fully enable rich interactions, video should be integrated with other distributed tools such as audio that enable natural collaborative behaviours within shared environments. Daly-Jones et al., (1998) also demonstrated through experimentation the significant advantages of video-conferencing over audio-only conferencing. They found that in the case of an audio-only channel, the absence of a visual channel resulted in less fluent conversations. VoIP was used to increase communications among users in an online meeting. It was suggested that the users experience would improve if they interacted with audiences through voice. It was found instead, that video provided a better sense of alertness among users (Causey et al., 2006). These findings suggest the provision of a visual channel in addition to audio, for better group interactions.

2.4.5 Video

The use of video in asynchronous meeting tools provides the users with a richer sense of presence and helps in the coordination of communication. It also facilitates emotional expression. Scholl et al., (2005) explain that delivering high quality video to larger groups remains a technical challenge since the available bandwidth has to be shared among users. The video needs to be transmitted to users simultaneously in order to aid meeting participation; however this puts constrains on the available bandwidth (Cohen & Wang, 2009). Thus a larger group has less bandwidth for each person's video stream, imposing severe limitations on the quality and leading to a high required level of compression.

Most video compression techniques, however, do not efficiently support low bit rates as they compress and transmit each and every entire frame. Different bandwidth reduction techniques for video streaming

exist. Examples include MPEG-4 and H.26. MPEG-4 achieves high compression rates by sending only the face model parameters across. However, problems arise in reconstructing the face model as the reconstructed models fail to appear natural or match the original video. H.26 is an automatic and robust waveform-based coding technique but isn't suited for low bandwidth as it does not make use of face models. The two techniques were combined in an effort to obtain a better compression for face video streaming, however, the resultant system could not work at very low bit rates (at 8Kbps)(Cohen et al., 2009). They further designed a low frame rate video compression system that enabled users to communicate over extremely low bandwidth. Good quality faces were automatically selected at the encoder side, compressed and transferred. An "image-morphing" technique was used to generate the normal video frame rate at the decoder side. It was found through experimentation that the system was better than other traditional low bit rate video codecs. In addition to Microsoft's MPEG-4 codec for compression, three other low bandwidth video compression standards exist. These are the discrete cosine transform (DCT), the feature-outline and the model-animation, which deliver usable video at less than 10kbps, 10kbps and 1kbps respectively (Chen, 2002). The MPEG-4, compared with other previous standards, had the ability to achieve a bit rate saving of more than 50% with the same quality J. Kim and B.Kim (2011).

Bandwidth can also be reduced by changing the compression parameters or lowering the frame rate, which also enables computational savings. However, reduction in the frame rate in some cases is problematic. Scholl et al., (2005) state that when video is used only to provide a sense of presence, for example to identify basic emotions in a video chat, one frame every five seconds may be acceptable; however if complex emotions are to be portrayed, then 0.2fps will not suffice. An example is the Portholes project by Bly and Dourish (1992). They demonstrated that a frame rate of one update every five minutes could provide alertness in a work setting but may not be adequate for remote classrooms (Chen, 2002). His experiments showed that lowering the frame rate from 25 to 15 and 5 fps did not decrease a person's understanding of the content of the video and suggested that 5 fps may be the minimum required frame rate. However experiments have shown that video could still be useful at 1 fps. So clearly the choice of frame rate depends on the type of application to be designed.

Monk & Watts, (1995) conducted an experiment to test the effect of the size of the video image on users. They had thirty two members of the general public work remotely from one another in pairs on some simple joint tasks. All the pairs had high quality audio links and were able to see one another's faces through an on-screen video image. For half the pairs this image was small (40 x 65 mm) and for the other half it was large (103 x 140mm). The conversations were analyzed and it was observed that the smaller video image resulted in formal and less fluent verbal interaction than the bigger image.

Chen, (2002) did a gesture detection video-conferencing experiment using three different frame-rates: full-motion, gesture sensitive, and low-update. In gesture sensitive, the video image was updated on detection of a hand being raised. His data revealed that conveying postures alone (low update) was insufficient for small group discussions due to difficulties with floor control. However conveying gestures in addition to postures was a viable option if limited bandwidth would otherwise prevent using videoconferencing at all. Another study, by Takao (1999), examined the effects of: face to face meetings (FF); switching video (SV), which showed only the current speaker; and mixing video (MV), which showed each group member simultaneously. His results revealed that MV yielded better group decision quality than FF and that SV and MV showed no difference

2.5 Discussion

It has been shown that some of the features used in video-conferencing systems are generally sensitive to bandwidth fluctuations and unstable connections. Various different techniques can be employed and used to improve meeting effectiveness in low bandwidth situations. For example, one could opt to send incremental updates in a very large meeting as opposed to sending entire packages at once. In addition, an approach that allows only the speaker in a meeting to send their streams could help reduce the number of packages transferred over a network. Specific factors such as delays, jitter, echo, packet loss, choice of codec, frame rates, bitrates etc. have an impact on the overall production of audio and video and therefore have to be analyzed and carefully selected to suit the user's preferences and ensure pleasant use of the software tool. Furthermore most Web conferencing applications, in general, cannot provide very good quality to their end users and support a limited number of participants per meeting (Lu et al., 2010).

Chapter 3

Design and implementation

This section starts by giving an overview of the design and description of the experimental system, the functionality it provides and the features it supports. It then gives details of the implementation of the software.

3.1 High-Level Overview of System: Front End

This part of the system involved designing the participant list and screen sharing components of the meeting tool. Additional features include a floor control mechanism and a Group Decision Support Tool (GDST) such as polling. The system offers the essential and basic features that are expected of most videoconferencing systems while utilizing minimal bandwidth. These features are adequate to give an acceptable user experience when the module is tested on its own, before integrating it with the other modules developed by the other team members.

Users are able to perform basic functions such as logging into and out of meetings, changing meeting rooms, creating new meetings, raising their hands or expressing their emotions. These are tackled as general meeting options that all participants can use. The participant list contains all the names of users in a meeting, arranged in the order of which they entered the meeting, with the first at the top. In addition, details of the host (Chair) and current presenter should be made visible to all users. The system supports multiple meetings where each meeting has its own unique list of participants. By clicking on an icon on the hand raising menu, the icon should appear in the participant list next to the user's name, representative of that user's current emotion or status. The list rearranges itself when users raise their hands, with those who have raised their hands appearing at the top of the list in the order of first raised at the top. In addition to this, there is a feature responsible for displaying information on current meeting statistics. These statistics correspond to the number of people in the meeting with their hands raised, the number of people that have agreed or disagreed and those that have stepped away. This is highlighted by the red box in the diagram below.

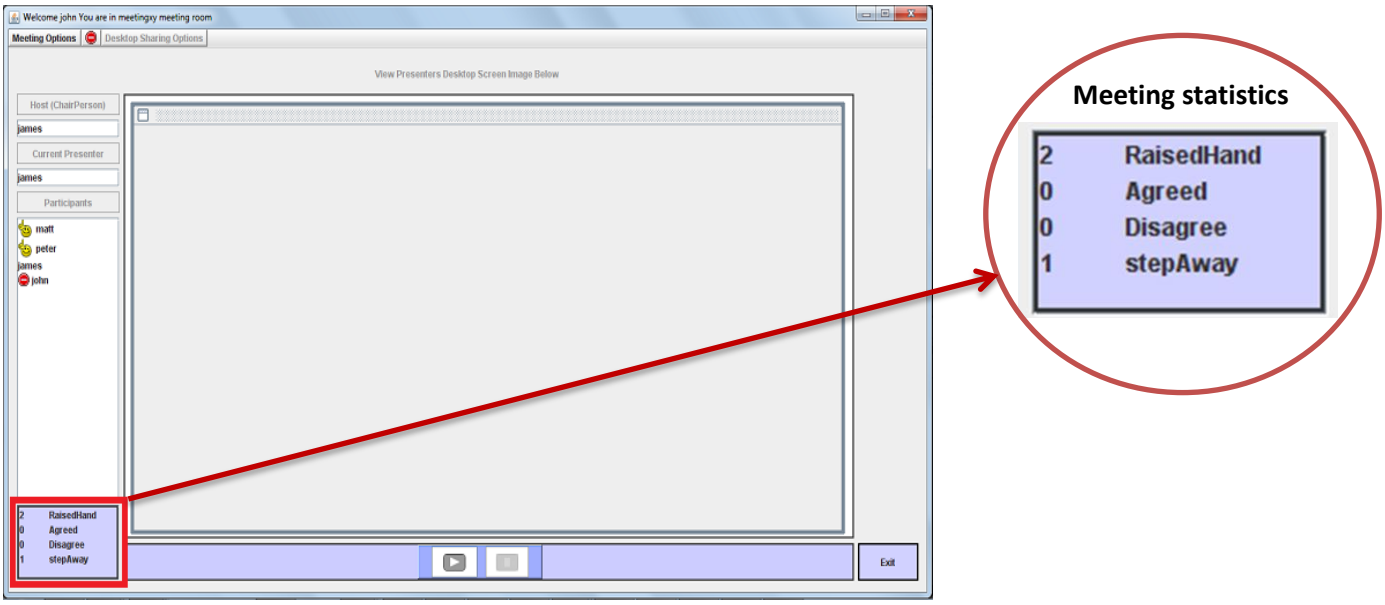


Figure 2: Meeting statistics displayed at the bottom of the window

The floor control mechanism controls who has access to certain resources at a given time. Users in a meeting can either be holding the floor, waiting for the floor or just observing meeting activities. The host of a meeting generally has most of the control and is able to temporarily pass control around to different participants. The user is able to request permission to access the floor by raising their hands using the hand raising menu previously mentioned. This automatically puts them in the list of users waiting for control. Once a participant receives control they become the current presenter. Presenters hold the floor at a given time and are the only ones who are able to share their desktop screens and create new polls. The host is the only one who can change the presenter, or change the host at any given time. By changing the host, total meeting control is permanently passed away from the current host to the new host. Meetings only have one host and one presenter at a time. The system maintains control and keeps track of resources unique to a meeting when simultaneous meetings occur. More screenshots of the system are shown when describing the implementation of the individual features of the meeting system.

The screen sharing feature is limited to the current presenter who has the ability to share their screen with all participants. Once a presenter decides to share their screen, participants in that meeting could have the option to either view that screen or not. The presenter also has the ability to create polls, where they send a question to all participants and they vote. Meeting Poll results are recorded so that they are always accessible to meeting participants. Lastly, users are able to successfully exit the application whenever desired. A summary of the list of features and their properties is shown in the diagram below:

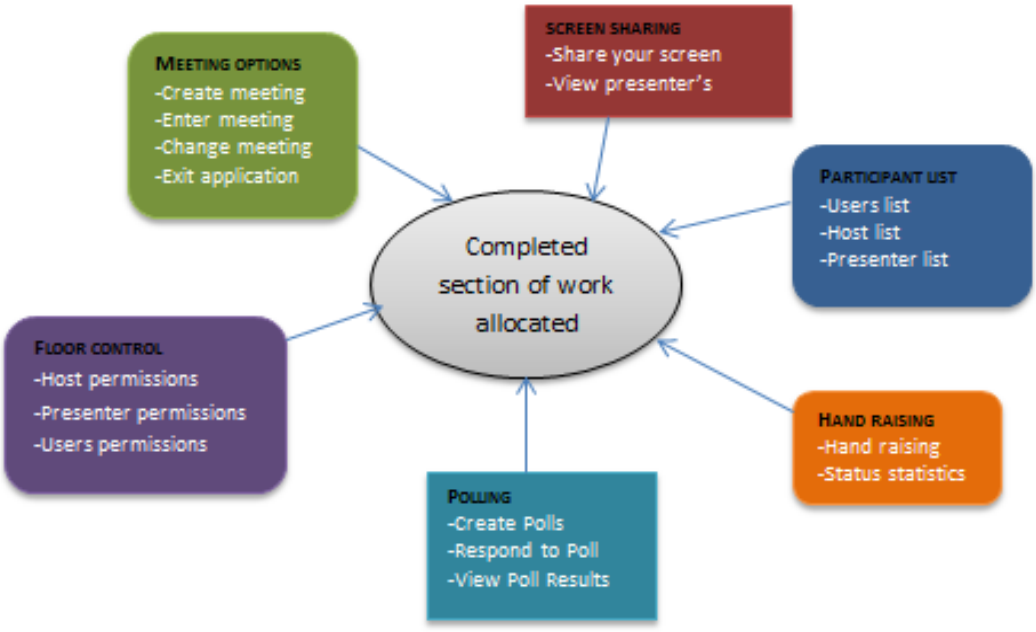


Figure 3: Breakdown of the desired meeting features

3.2 Low-Level Implementation: Back End

In order to be able to test and get results to answer the research questions, the participant list, floor control, hand raising mechanism, polling and desktop sharing features of the videoconferencing systems had to be developed. The main challenge was to optimize the entire system for low bandwidth, while providing usability, presence, decent floor control, good user interaction and enabling the interface to provide an acceptable end-user experience. The system was implemented in such a way that the client application would have to be running on each user's computer.

3.2.1 Technology and tools

The system was implemented using client-server architecture. Java Web-based technology was used to develop the system. Java is a platform-independent, open source programming language that supports multithreading and has many APIs that programmers can easily access. The Java Swing library was used to provide the graphical user interface to the system. The Hyper Text Transfer Protocol (HTTP) was used for communication between the Java servlets and HTTP client using TCP/IP at the Transport layer. Apache Tomcat was used as the Web server for the software.

Another important library that was used was the ImageMagick library for image processing and manipulation, specifically for the desktop sharing application. This is an open source software suit that enables image processing and manipulation such as creating, editing, composing and converting bitmap images.

The project was done in three iterations: the first one was to test whether the technology needed to implement this system actually existed and worked; the second tested usability and functionality to prepare for the final prototype. The third iteration was an optimization of the second, and involved implementing techniques that would reduce the amount of bandwidth needed by the system. Testing the systems would determine whether the implemented bandwidth control mechanisms aided the performance of the system while also producing an acceptable and satisfactory end-user experience. The participant list, hand raising mechanism, flow control and polling were given priority over the desktop sharing application as transmission only involved simple string objects and consumed less bandwidth.

The following sections talk about the design of each of the features mentioned above starting from the general meeting options to the screen sharing feature. The layout is shown below.

- The HTTP protocol
- Implementation of the Participants List
- Implementation of the Meeting Options
- Implementation of the Handraising Mechanism
- Implementation of the Floor Control Mechanism
- Implementation of Polling
- Implementation of Desktop sharing

3.2.2 The HTTP Protocol

The HTTP Protocol is a stateless and connectionless protocol based on a request-response pattern that takes place over TCP/IP at the transport layer (JCEA Part 1: Protocols, 2002). Therefore the server does not retain information or state of each consecutive transaction during multiple requests. Each request is viewed as a new and different independent transaction without knowledge of previous requests. As a result, a user's progress is not tracked from transaction to transaction. An illustration of the client server architecture over HTTP is shown below.

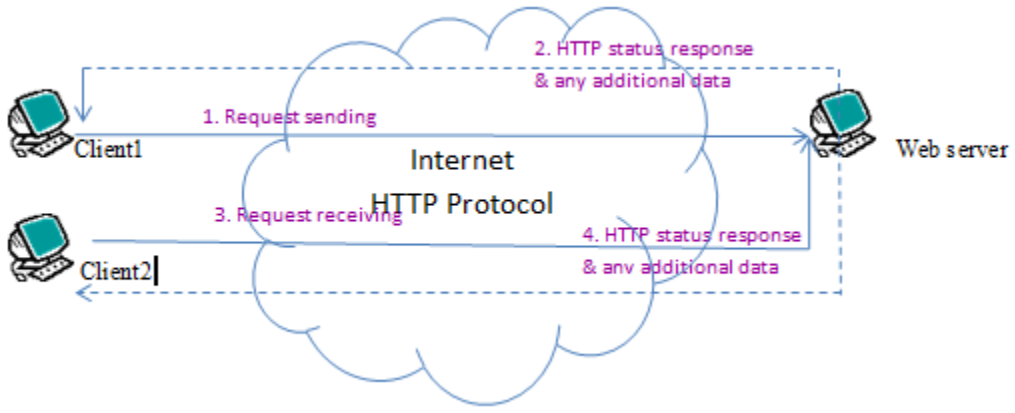


Figure 4: client server architecture

The HTTP protocol is based on a request- response approach; therefore a response is only generated and sent based on a given request (known as pull technology). The transaction is always initiated by the client sending a request to the server. HTTP does not have a technique to push data from the server to the client on its own. Therefore several supplementary Java libraries were used. These are the Commons Libraries

(fileupload, io, logging, httpclient, codec), which helped in multipart file upload, downloads and keeping long-lived connections. A long-lived connection is obtained by not terminating a connection at the server after a response has been received from a client. The connection is kept open so that the server could continue sending data when events occurr without the need to be triggered by a new request. This would help reduce the amount of data being transmitted over the network through redundant requests for the same resource and reduce latency.

Data sent to the server has to be encoded first using the Java URL encoder class. This was done as a precaution as the data to be transmitted might contain special characters such as "$ & < > ? ; # : = , " ' ~ + %" and " ".These characters might not be accepted in valid URLs as they have special meanings and may be altered during the transfer (Introduction to URL encoding, n.d) . A few examples of the encoding and decoding are shown below. The space character has two encodings and it can either be encoded as **%20** or **+** as shown in the last example.

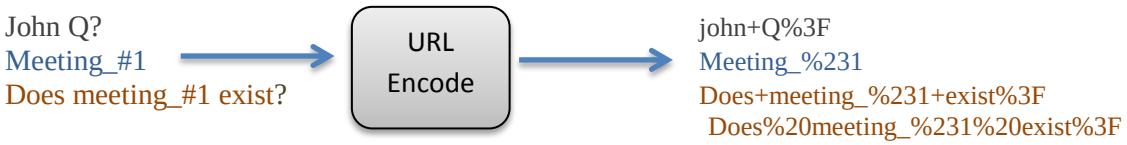


Figure 5: URL Encodings examples for different strings

Two kinds of methods called GET and POST are used for data transmission in which Key-Value pairs are sent across to the server. An attribute is associated with a value entered by a user to generate the key (attribute) value pair. For example, if the user enters john when asked for the username, the key value pair will be "username = john". Many key value pairs may be attached to a URL, however, each key value pair is separated by the "&" character. The GET method is HTTP's default method which appends the parameters (key-value pair) as a query string at the end of a URL (HTTP Protocol, 2008). An example is shown below.

HTTP GET:
http://localhost:8080/CSERVERLOGIN/Aserver?Username=john&Meetingname=cscmeeting

The POST method attaches the query string along with the requested object, so that it does not get appended to the URL and is not visible. The parameters are transferred as a stream. This is useful in the case of sending encrypted username and passwords across the internet and in the case of sending file data across as some file contents are too large to be appended to the end of a URL. The application uses either the GET or POST methods for data transfer depending on the type of data to be transmitted.

One of the challenges faced when using this protocol was to figure out how to implement the server such that it would behave in a manner where transactions and activities could be associated with specific users, that is, to make it appear as if it were keeping track and records of each user's previous transactions. A solution involved appending metadata to the end of a URL. This metadata had to be unique to each user and it had to be sent in each and every consecutive request made to the server. The metadata used for identifying users was their usernames and the name of the meeting they were participating in. As a result, no two users on the entire system could have the same username.

The client-server diagram below illustrates the basics of how the overall system was implemented.

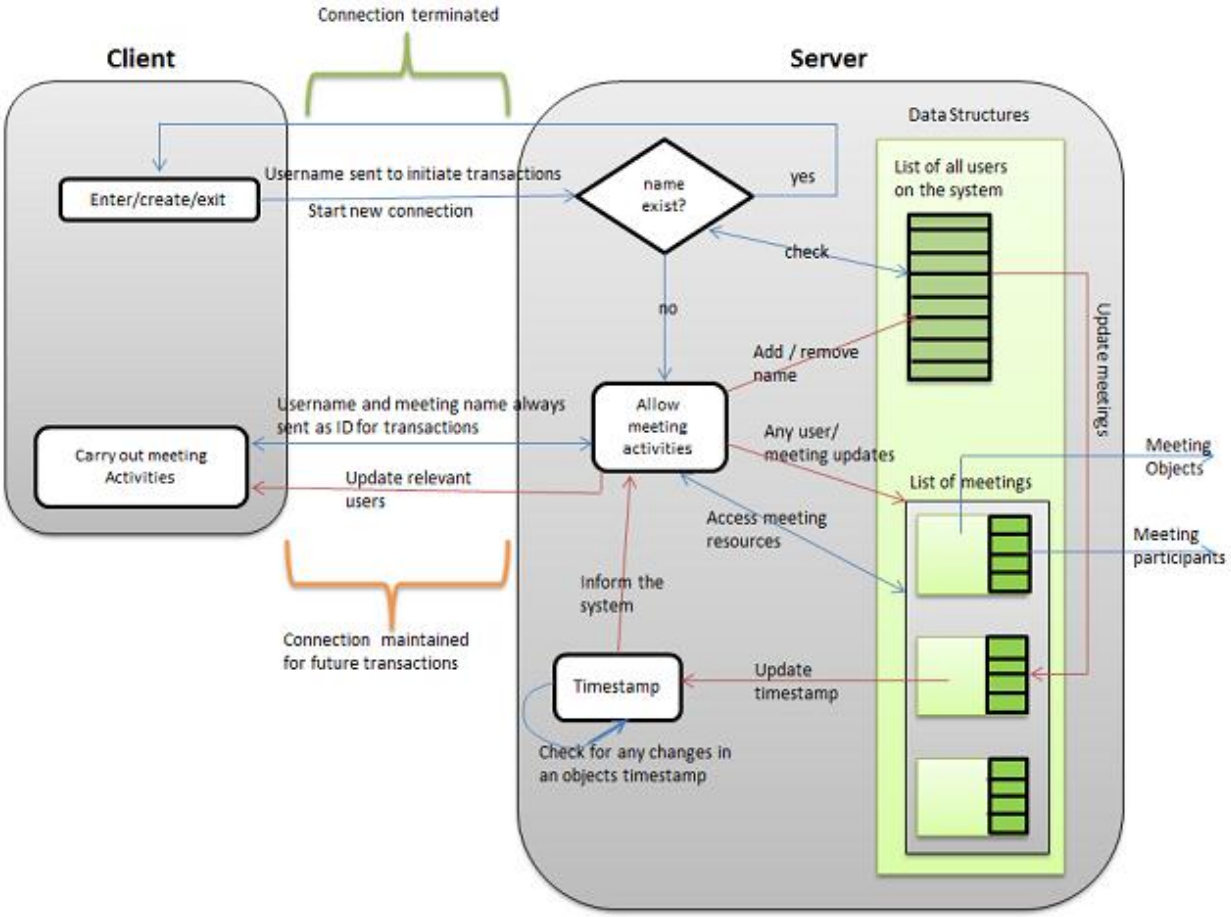


Figure 6: Overview of the flow of data in system

The server uses list data structures to store information that it can use to identify users. A list of all users on the system and a list of meeting objects are kept. A timer for each user and meeting object is also kept. Any new update or transaction that occurs can be detected by comparing the latest and newest update time. This update is propagated to users specific to a meeting and then their own individual times are updated to the time of the new transaction. The implementation of the participant list is described below.

3.2.3 Participant List

The participant list is important as it exposes all the users that are in a meeting. Implementing the participant list involved the use of a "many to many" approach, where multiple users send updates to the server and multiple responses are sent to the different clients from the server.

3.2.3.1 List of Users

This part of the system was designed such that duplicate usernames are not allowed. The server keeps track of all the users on the system regardless of which meeting they are attending. A user logs onto the system and the server responds with a go ahead if there is no similar username being used among the current users. Once authenticated, they can opt to either enter a meeting room or create a new one. The

first option displays a list of all the available meetings on the system with details of the meeting name, the host (chair) of that meeting, and the number of people currently participating in that meeting; the second automatically generates the new meeting and opens a new window with the new user as one of the participants in the meeting room. Individuals who create a meeting automatically become the host and the current speaker and appear as the available attendee on the participant list. The host would initially have control of the meeting and be able to run the meeting. As users join that meeting, the list of participants gets updated with the new users. Under normal circumstances, users would appear on the list in the order which they entered. The snapshot of the participant list is shown below:

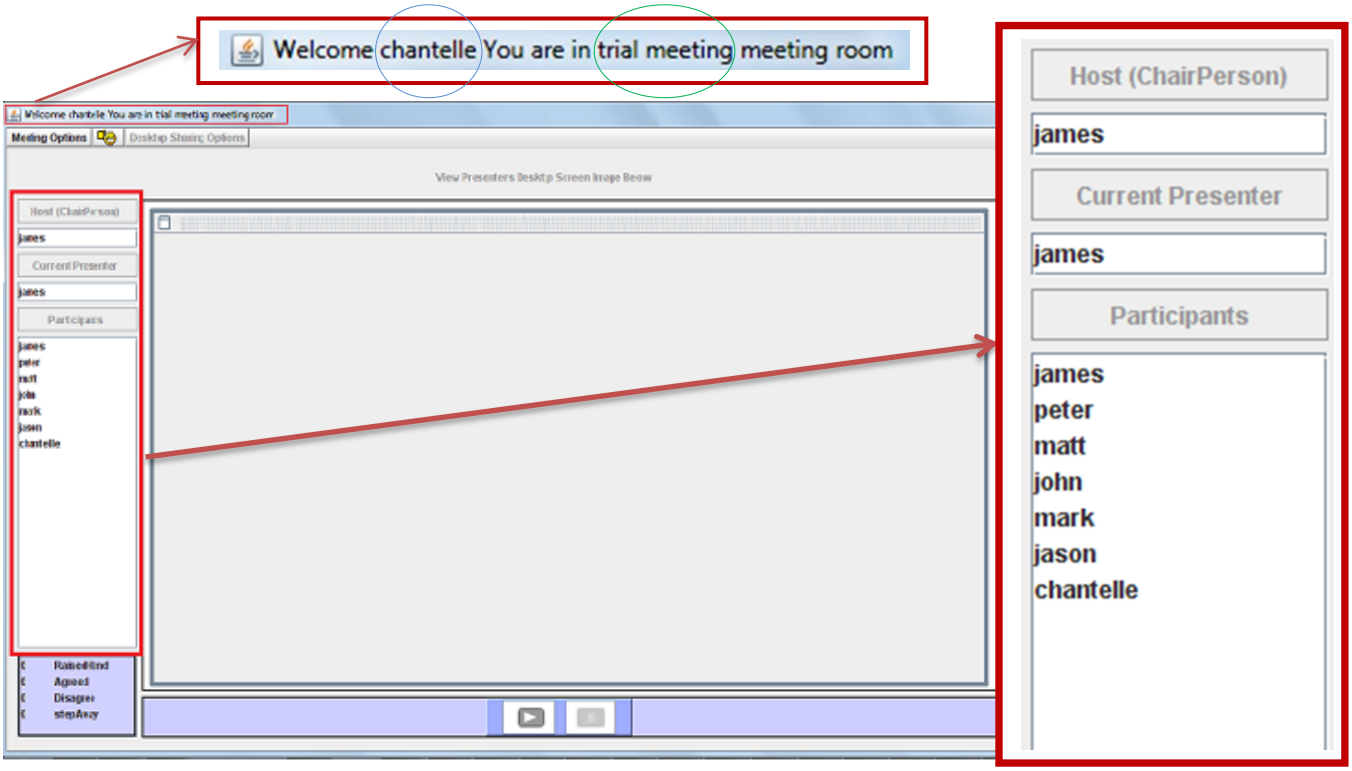


Figure 7: Participant list

The participant list is shown magnified on the left of the diagram. The two circles indicate who the user is (blue) and what meeting they are in (green). In this case the user is chantelle and they are in a meeting called "trail meeting". They appear last on the participant list. This indicates that they were last to join the meeting.

Implementing this part involved maintaining four user lists, two at the server side and one at the client side. The client side list contains a list of users corresponding to the meeting in which the client is in. If a request is sent to create a new meeting, a new list containing the available meetings from the server side is generated on the client side and displayed to the user. The server has a list of all users on the system and another list of all meeting objects taking place on the system. Each meeting object has a name, list of user objects, list of meeting state objects, hosts name, current presenter's name, a meeting timestamp and a meetingpoll timestamp. Each user object has a name, a meeting name, a user timestamp, a userpoll time stamp and a current status. Each state object has a name, count, and a list of state users. This is shown in Figure 8 below.

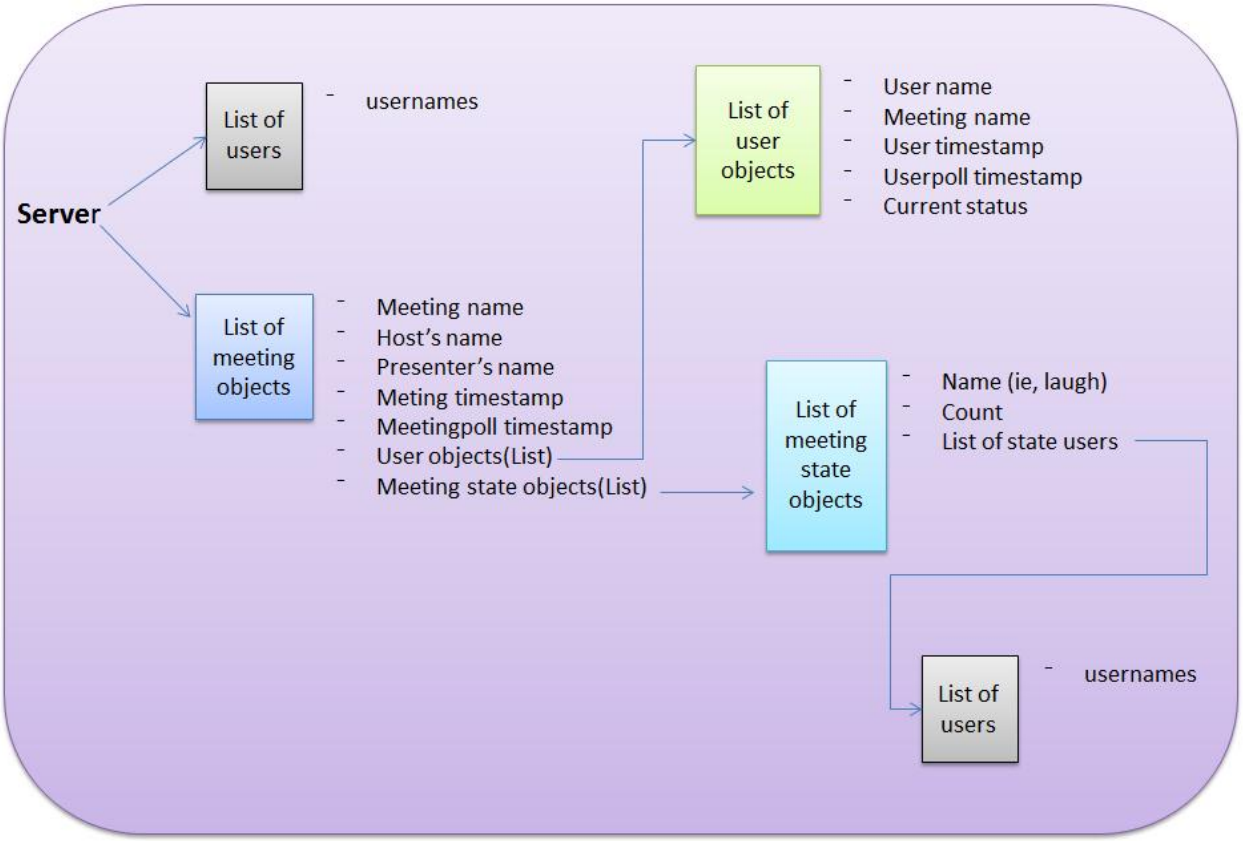


Figure 8: Data structures in the server

To keep track of each user, the client always has to send its URL encoded identification details, namely the username and meeting name to the server in each request as the server does not keep records of any previous transactions. Initially the user sends the username at the beginning in a request; the server checks this user against a list of system users to avoid duplicate usernames. If no similar name is found, then it authenticates the person and they get added onto the list of system users. Next the user either creates a new meeting or enters an existing one.

3.2.3.4 Updating the participant lists

From this stage onwards, every interaction with the server requires that the username and meeting name be sent as parameters in each request. In order for the system to be able to keep real-time information about all the current participants in a meeting, the list of meetings and users on the server side as has mentioned, keep a timestamp of each change. Each meeting has a timestamp responsible for keeping track of event changes in the meeting. In addition, each user also has a timestamp that corresponds to the time the last copy of an update was obtained from the meeting. A long-lived connection has to be maintained with each user so that the server can push data to the clients each time the meeting participant list changes. IP Addresses were not used as multiple users can be accessing the server from what appears to be the same IP address and these can be confusing to the system and treated as one IP address. This may be the case when multiple local IP addresses share one public IP address to access external sites. Each time the meeting participant list changes (either by a user entering or exiting the meeting), the meeting timestamp is updated to that time of change. If a user's timestamp differs by being less than the meeting timestamp, then that user's participant list is updated by sending the new list. The user's timestamp is

then updated to match the meeting timestamp. As long as these two timestamps are equal, no events for that client are triggered and therefore nothing is sent across to the client. Initially a polling approach was used where the client continuously sent requests to the server to check for changes in the participant list. However this did not prove beneficial as there was performance overhead and it took up unnecessary bandwidth. In addition it also proved difficult in actually detecting accurate changes in the participant lists. The method described above, on the other hand, was event triggered and data was only transmitted upon an event occurring, thus saving bandwidth.

3.2.3 Meeting Options

3.2.3.2 Create a new meeting

To create a new meeting, the user enters the meeting name they would like to create and these details together with the initial username entered at the beginning are propagated to the server. The server checks the request and upon seeing that it is a create meeting request, it checks to see if the current username is hosting any meetings at the moment as a user can host only one meeting on the system at a time. If the check is clear then the meeting name is checked against similar meeting names; however if the user is already hosting a meeting, a notification message is sent notifying them that they will not be able to create a new meeting as they are already hosting one. They are then forwarded to a list of all the available meetings on the system. If the meeting name already exists, then a notification is sent back requesting the user to enter another meeting name as the one previously entered is being already used. If the request passes the above two checks then a new meeting is registered on the system and the user is automatically entered into the system. The diagram below illustrates this.

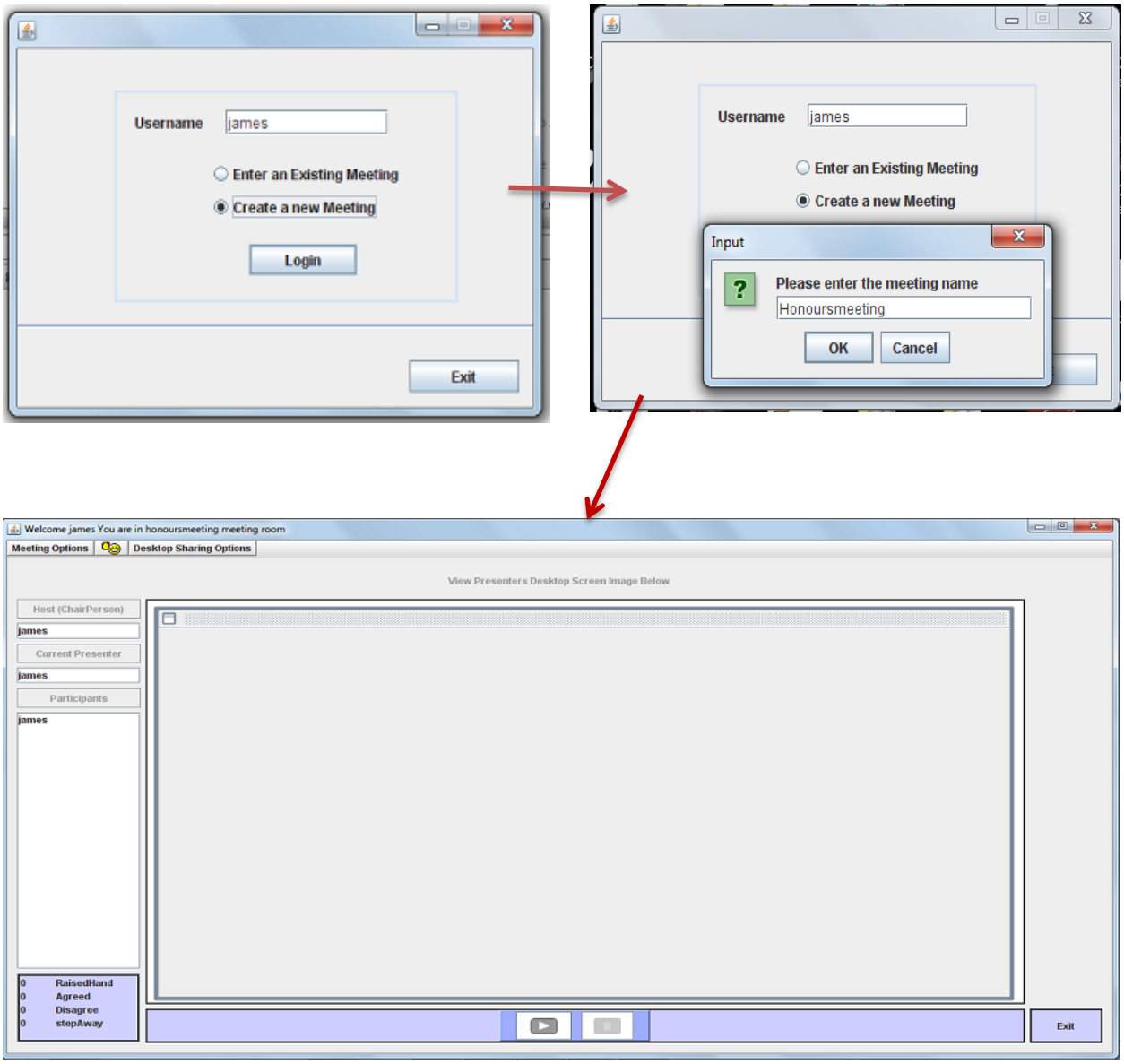


Figure 9: Creating a new meeting

As mentioned, the one who created the meeting automatically fills the host and current speaker roles, therefore they appear in the host list, presenter list and the participant list. The host has full control of the meeting and can change the speaker as they like. This will be discussed further in the floor control section. Figure 10 shows screen shots of the processing errors due to duplicate usernames, or meetingnames.

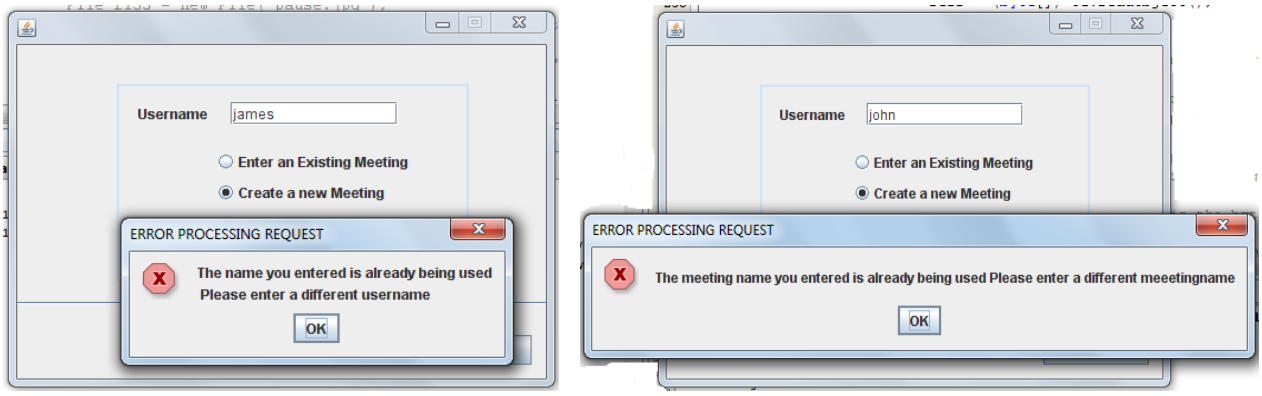


Figure 10: Errors when creating a meeting

3.2.3.3 Enter a meeting or changing a meeting room

To enter or change the current meeting room, the server responds with a list of available meetings once a request to enter a meeting is received from the client. This list is displayed on the client side with details of the meeting such as the name, the host, the current speaker and number of participants in the meeting room. This is illustrated in Figure 12. From this list the user can now click on the desired meeting and the system enters them into it. A window opens in which the meeting room is displayed together with the list of participants. The user is automatically added to the end of the list of users in the meeting, and their name is propagated to all users in that meeting.

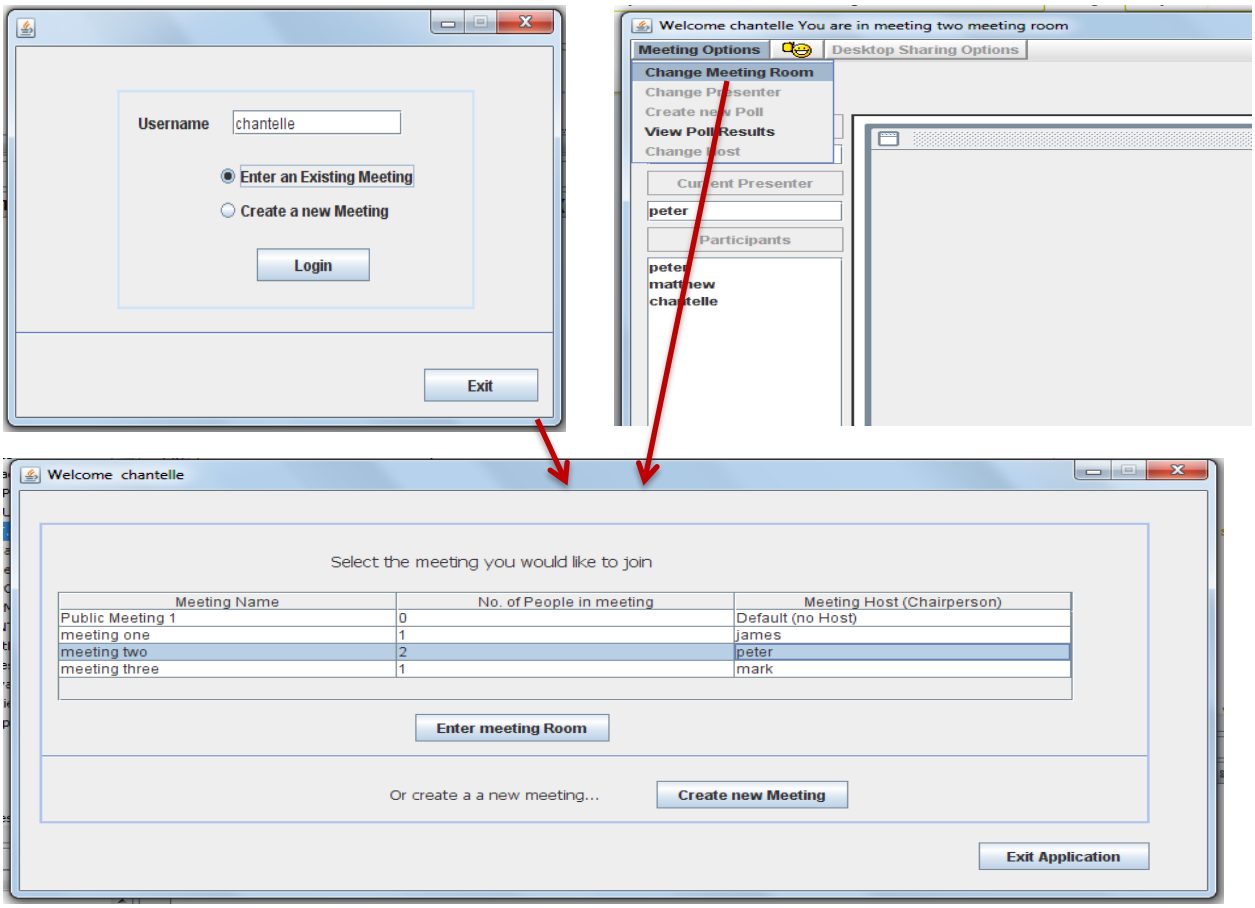


Figure 11:Snapshots of entering or changing meeting rooms

3.2.4 Hand Raising and Status Statistics updates

Icons are used in the hand raising mechanism in order to increase the sense of presence. Users can indicate their emotions during the meeting process by clicking on the relevant icon in the hand raising menu situated on the left of the meeting room window. This would result in the icon appearing next to their name in the participant list, which everyone in that meeting can view. In implementing this section, each user sends an update of their status to the server once an icon is clicked. The server then updates the individual user's status in the user list. This in turn updates the meeting list timestamp as it indicates a change in the meeting object. Threads of users with the older timestamps connected with long lived connections receive this event notification and then retrieve and propagate this new list with the updated statuses to the relevant users. The list on the client side then updates and reflects the new change by displaying the icon by the user's username in the list. The following expressions were used: Raise hand, drop hand, step away, returned, laughing, speak louder, agree, disagree and clear statuses. A screenshot of the handraising system is shown below.

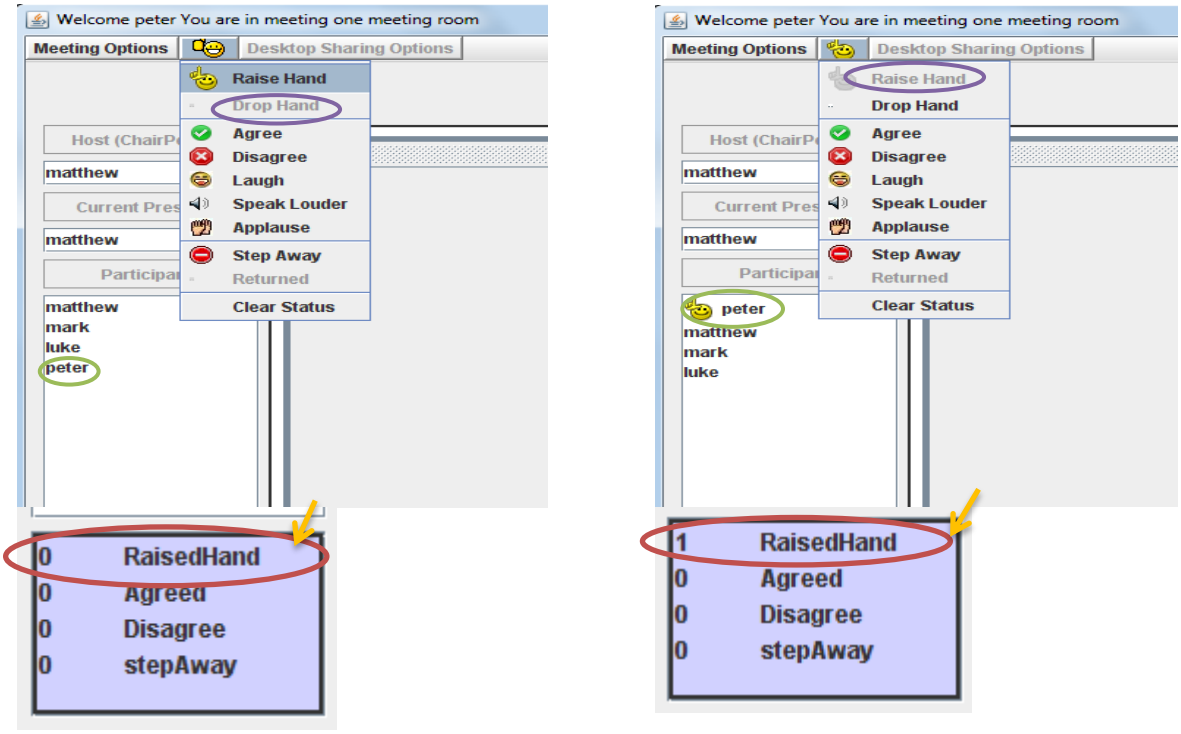


Figure 12: Screenshots of handraising

When one raises their hands, the participant list rearranges itself with those that have raised their hands appearing at the top of the list in order of who raised their hand first. The option to drop their hands is enabled so that can opt to drop their hands at any time by clicking on the drop hand menu item. This is accomplished again using timestamps from the client side that record the exact time a hand is raised. This time is sent to the server and the user list is updated and sent to all the clients based on the timestamp event trigger of the meeting object. Once on the client side, the list received from the server is then sorted by comparing timestamps of hands raised, and ordered according to the first that was raised. The other users appear after the list of hands raised in order of first-come first-appear or to be displayed in line. At the bottom of the window is a status box displaying the number of people in a meeting at a given point that are agreeing, disagreeing, or raising their hands in addition to those who have stepped away. This was implemented in a similar way to the participant list by sending the user's name with metadata, meeting name and the state to the server, which would in turn updates its meeting status list and then propagates these changes to the other clients using the timestamps. This system was helpful in that it enabled one to keep track of the number of hands raised and in what order to give floor rights fairly. At all instances in the meeting, a user is able to view and keep track of all meeting changes in the meeting room from the view of the application meeting room window.

3.2.5 Floor control

Floor is the permission to access or manipulate a specific shared resource or a set of resources temporarily. Floor-control refers to the mechanism that enables applications or users to gain safe and mutually exclusive or non-exclusive input access to the shared object or resource. In this application the resources that need to be controlled are those currently holding the floor (current speaker), being able to change the current speaker like handing a microphone or passing a token around. This is controlled by the

host. There can only be one host and one presenter at a specific time per meeting. The host is able to change the speaker by giving them the floor. This was implemented by having a jComboBox of all the usernames from the participant list appear when the option to change presenter or host was selected. The host could then select any user in the jComboBox and automatically make them the new presenter or host. The presenter is then the only one who has the ability to share their screens and to create new meeting polls for participants to vote when necessary. This enables the meeting to be more controlled and reduces the amount of traffic such as everyone's desktop image being sent to the server continuously. In this case only the presenter's image is sent to the server and the server propagates this image to other clients upon request. Therefore it operates on a one to many basis. In addition, certain controls are specific only to the host. The host can also choose to release control of the meeting by changing the host to someone else in the case where they probably cannot host it anymore or have to leave the meeting early.

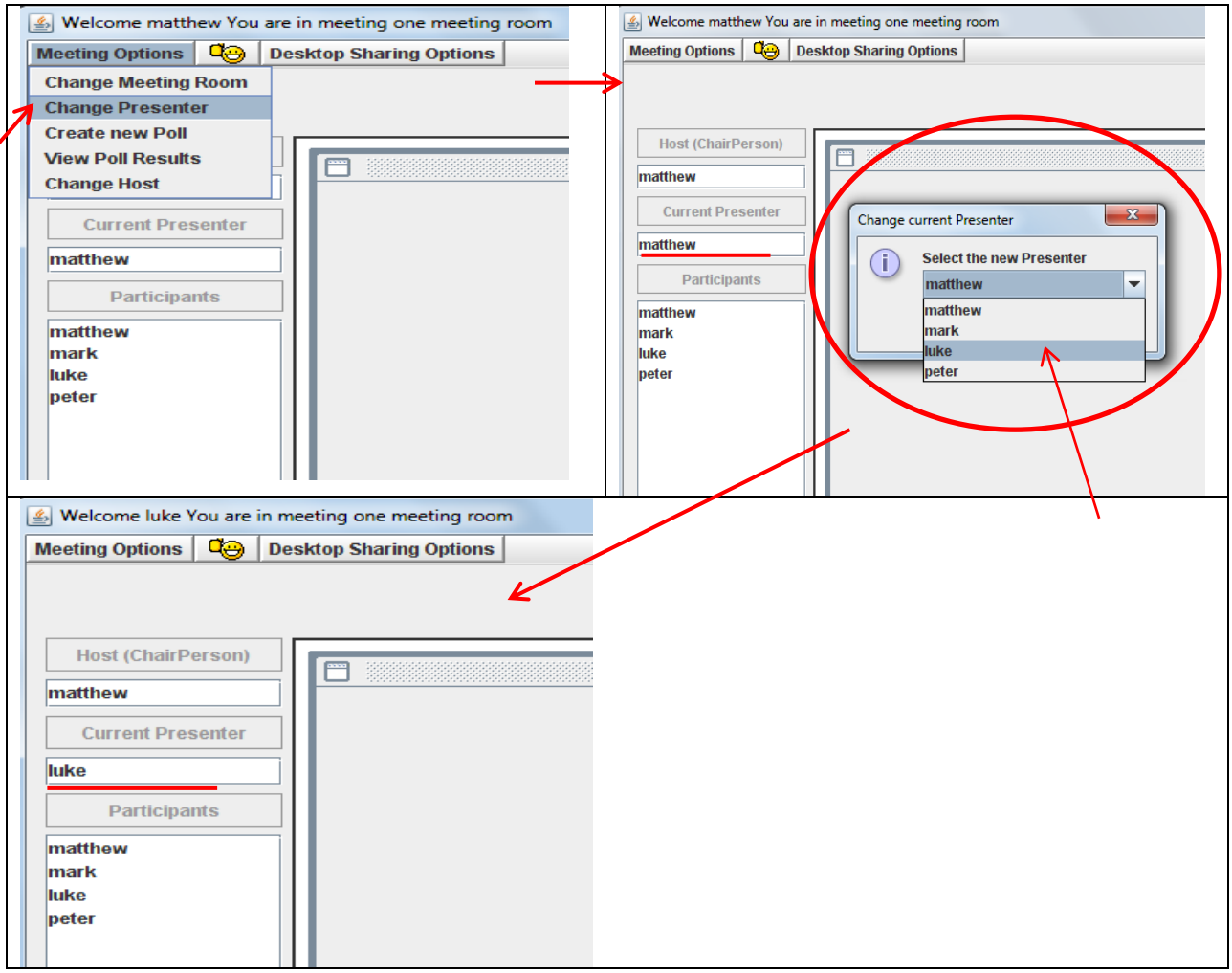
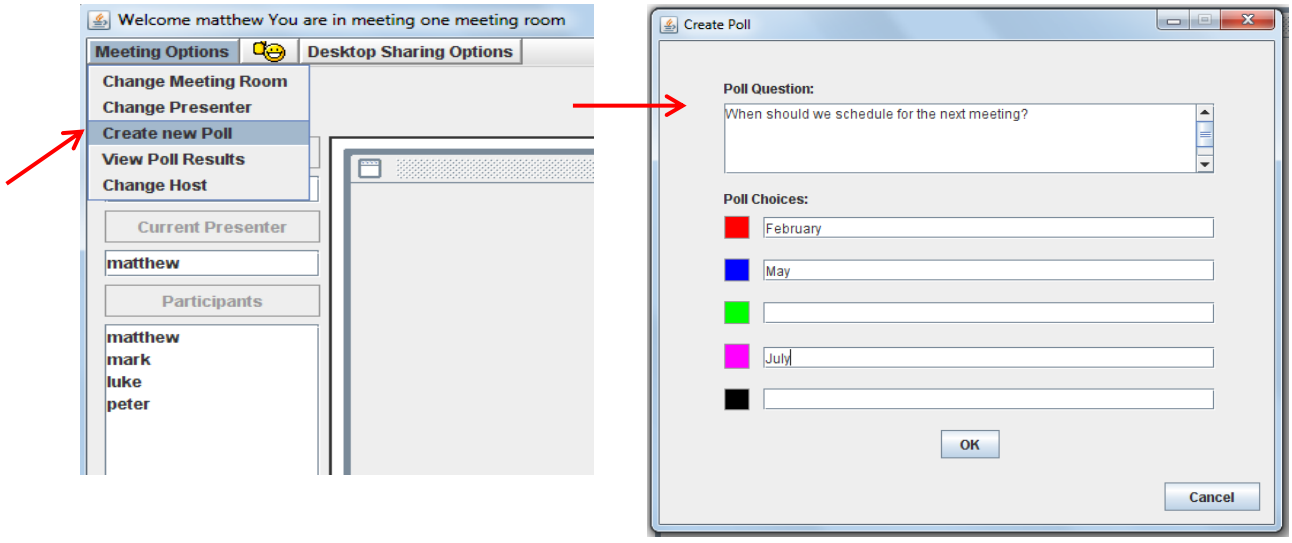


Figure 13: Changing the presenter

Any user who creates a meeting is set as the host and presenter of that meeting on the server side. This is done by adding them to the different lists under that meeting. When a user logs onto that meeting, they receive a list of all the users in that meeting. However, in order to identify the host or the presenter from that list, the host or participant name is appended at the server with a unique string to identify them. Before the list of users is displayed on the client screen, this list is searched through for those identifying strings and the user identified with that string is added onto the host or presenter display. Then all the users are added onto the participant list. During the searching process, the client also constantly checks to see if the username found with the host or presenter identifying string matches theirs. If it does, the display window immediately responds by activating all the features that should be accessible by the host or presenter respectively. By doing so, this user gains temporary control over the system, and special privileges to perform actions such as to create polls or share their screen which they would have otherwise been unable to perform as a normal participant.

3.2.6 Polling

Polling is a group decision support tool and it enables users to anonymously vote in meetings and therefore quicken the decision making process. More people are comfortable expressing their true opinions on things if they know that their decisions are anonymous. Any user on the system could view all the polls that had been conducted in a meeting and their results. However, creating a poll was limited to the presenter only. If a presenter selected the option of creating a poll, a window would open requesting for the poll question and its alternative answers. This data would then be encoded and transmitted to the server with username and meeting name. At the server side, a list of polls is kept. Each poll has a name, the meeting it belonged to, list of poll options, and a list of voters for that poll. Once a poll was received, it would be added to the polls list. The current poll made would be recorded as a string under that user's username, in their servlet context with data identifying them as the meeting presenters. In addition, each meeting object in the meeting list at the server had a meeting pollstamp. This pollstamp would be updated to the current time when the new poll is made. The server would initially check if a user belonged to a meeting and then continuously check that meeting's poll stamp against the user's pollstamps to see if they were different. If the meeting pollstamp was greater than the user's pollstamps, the current poll question would be extracted from the meeting presenter's servlet context and forwarded to that user. This was made possible by maintaining a long lived connection with all users at the server side in order to push data to them anytime a change on the system was detected. Screenshots of the polling feature are shown below.



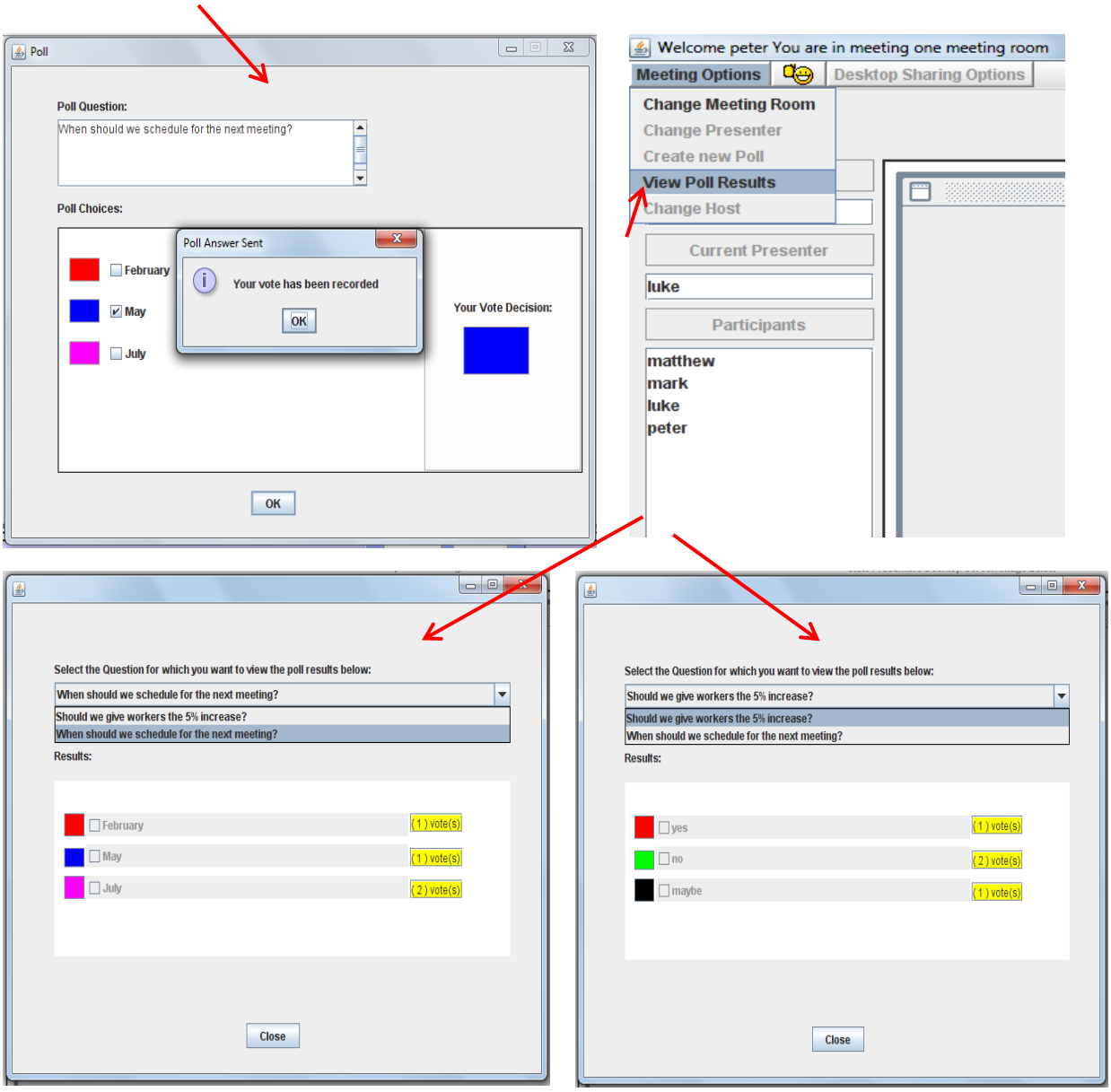


Figure 14: Screenshots of the polling feature

Users who received polls had to vote. Voting was anonymous and votes could not be associated with specific users when viewed at the client side. Once an option on the poll has been selected, this is sent to the server. To avoid duplicate votes from the same user on the same poll, each poll object keeps a list of voters. If a vote is received, this list is checked to see if the user had previously voted. If the user is not found in the list, the vote count for the poll option which they selected is incremented and the user added to the list of poll voters. This way the system maintained the number of votes received for each poll option and ensured that votes did not exceed the number of participants in a meeting.

For any user that requested to view the results of all the polls conducted in the meeting so far, the server would respond by iterating through the list of meeting polls mentioned above. Each poll's meeting name would be compared to that user's meeting name. If a match was found, the poll question and results would be transmitted to the client. These would be stored in another list on the client side, and this list would be added onto a jComboBox. The user can then select form the jComboBox whichever poll results

they want to view. The result of the poll selected is displayed in a window and users can iterate back and forth through the various polls and their results.

3.2.7 Screen sharing

Presenters could opt to share their screens while presenting in order to illustrate a point or express an idea by enabling remote users to see what is happening on their screens. The screen sharing feature is the most bandwidth-consuming so a design that would enable the feature to work with minimal bandwidth had to be planned and implemented. As mentioned, screen sharing was limited to the current presenter who was the only one who could decide to share their screen or not during a meeting. This is a "one to many" approach, which would aid in minimizing the number of packets transferred over the network. In this case only one client (the presenter) would send their desktop images to the server.

In order to reduce the amount of data transferred during screen sharing, a differencing approach was used. This idea was based on the fact that, more often than not, consecutive screen images are quite similar and therefore instead of sending an entire image packet across to the server, a difference could be computed and only the parts that had changed in the image could be transmitted. The ImageMagick library mentioned previously was used to achieve this effect. An initial image would be sent to the server, and from then onwards only the differences were sent. Figure 4 below shows the flow of data in the system for the screen sharing application.

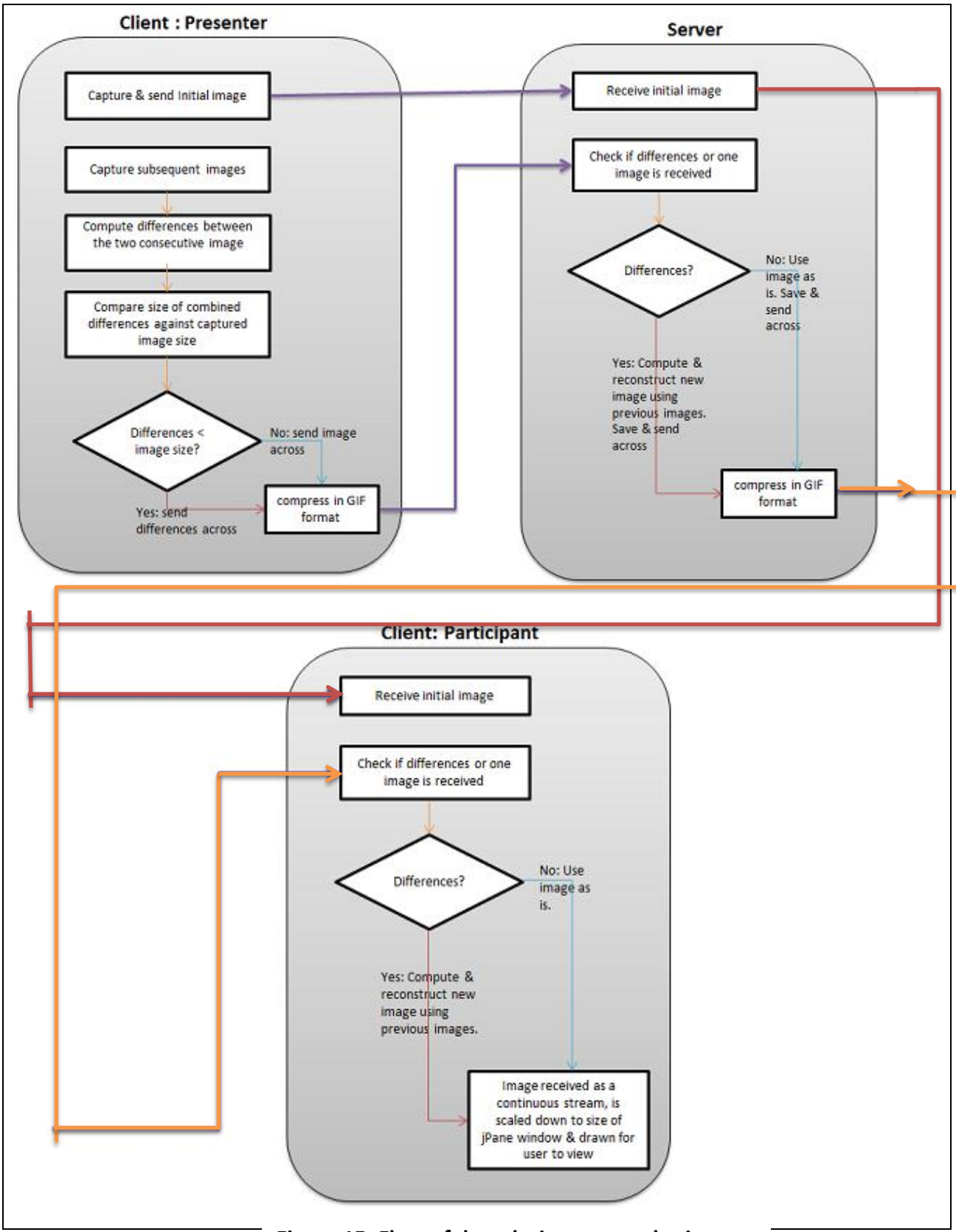
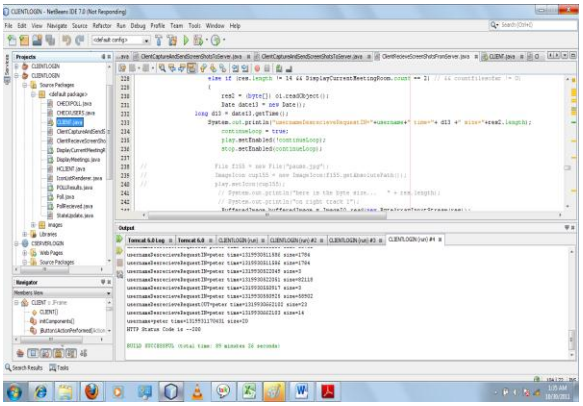


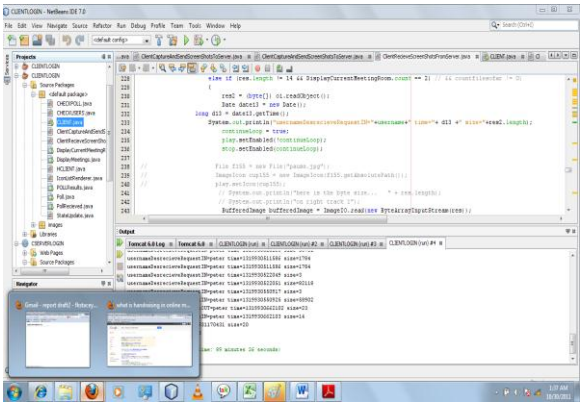
Figure 15: Flow of data during screen sharing

At the server, the images are

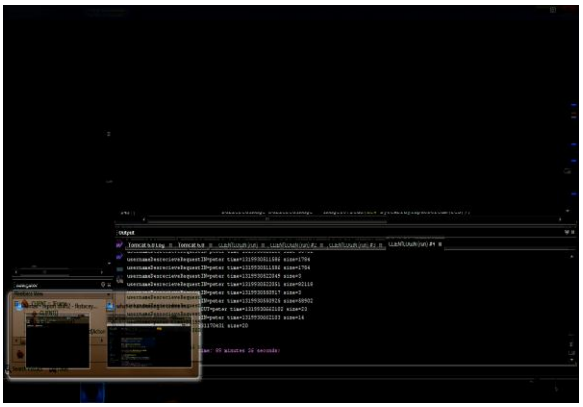
The images are reconstructed at the server in order to keep all images received by meeting “latecomers” in sync with the current differences being sent across by the presenter. For example, if an initial image is sent once, the second consecutive set of image differences received by the server from the same client no longer correspond to the initial image, therefore if another user is to log on and receive this very first initial image from the server, that client will no longer be able to reconstruct the original image. This is because the differences being now sent correspond to an undated version of that image, and only that update version can produce a perfect reconstruction of the original image. The subtraction of one image from another may occasionally result in negative numbers which, if not dealt with, hinder a perfect final reconstruction of the desired image. Instead of creating one image difference, two image differences were obtained by subtracting consecutive images both ways. The following screenshots illustrate this.



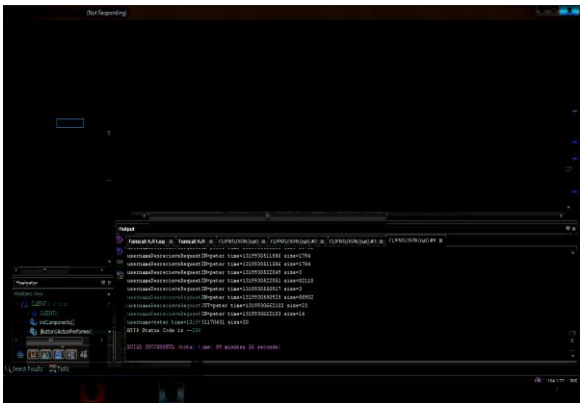
Screen image A



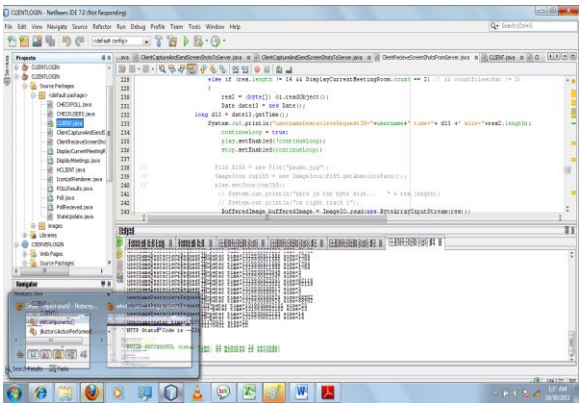
Screen image B

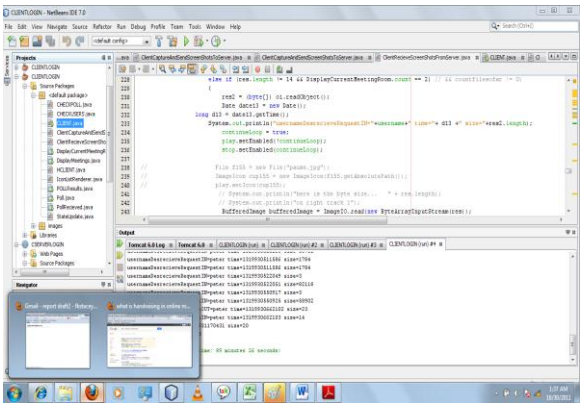


Screen image (A-B)



Screen image (B-A)



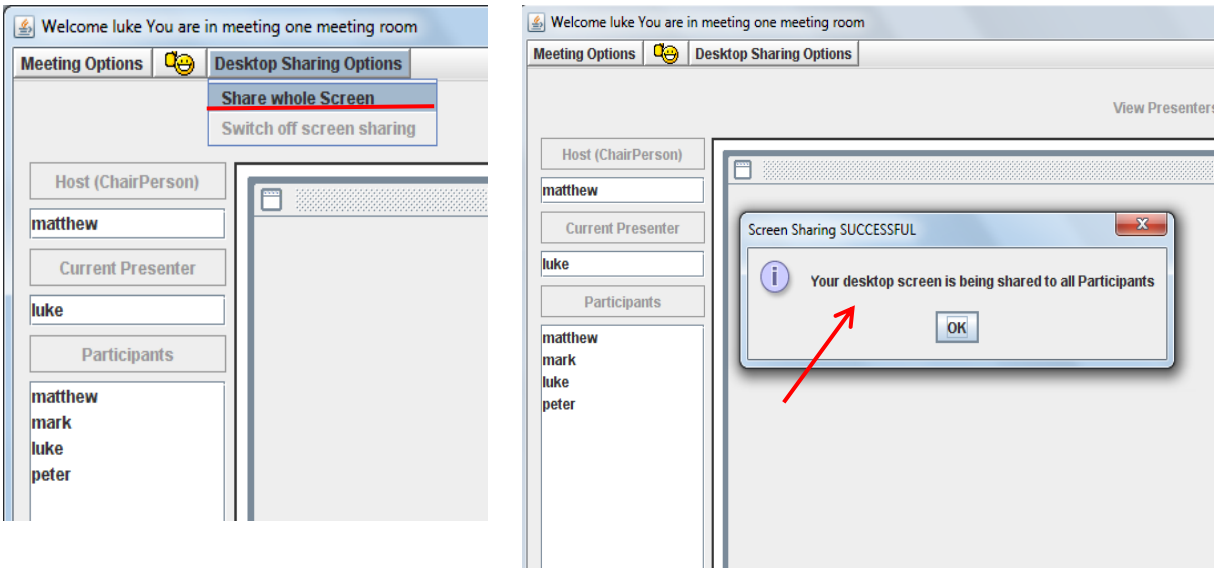


Screen image (A-(A-B)) Screen image (A-(A-B)) + (B-A) = image B

Figure 16: Computing image differences using ImageMagick

The two resultant image differences appearing as black in the above diagrams are much smaller in size then the entire image (in this case B). Only image A is sent across and image B is produced through the image reconstructions.

It was found that this approach also preserved the quality of the image and therefore the image did not degrade with time. The size of the two images obtained from the subtractions were added and compared to the size of the current image B in the above example. If their size was found to be smaller than the current image, the differences were sent to the server, otherwise the whole image was sent. In addition to this, the images were further compressed using the GIF image compression before sending across. The decision to use the GIF compression came about as a result of comparisons between the PNG, JPEG and GIF compressions. It was found that on average GIF provided a better compression than JPEG and PNG. For example, a desktop image of dimensions 768 x 1366 compressed with JPEG resulted in 100588 bytes, PNG compression produced 117570 bytes whereas the compression by GIF compression resulted in 55224 bytes. Once this image was received and constructed on the client side, it was then scaled down to the size of the user's jPanel window and painted there.The approach also raised the possibility of clients receiving the same pair of difference images more than once. In this case, performing the reconstruction would distort the current mage they had as the differences might have probably been used in the reconstruction of that same image. Therefore reapplying the same differences to that image again could result in bad quality images that do not match the original ones and further affect the ability to get other future proper screen images out of the reconstructions. In order to control flow and maintain the quality of images on client sides, the server used timestamps for both the image file and each user. If the user received an updated version of the differences, their timestamps would be changed to match the image timestamp. As long as these were equal or the clients time stamp was greater than the image timestamps, then no images were sent to the client. Screenshots of the screen sharing application is shown below.



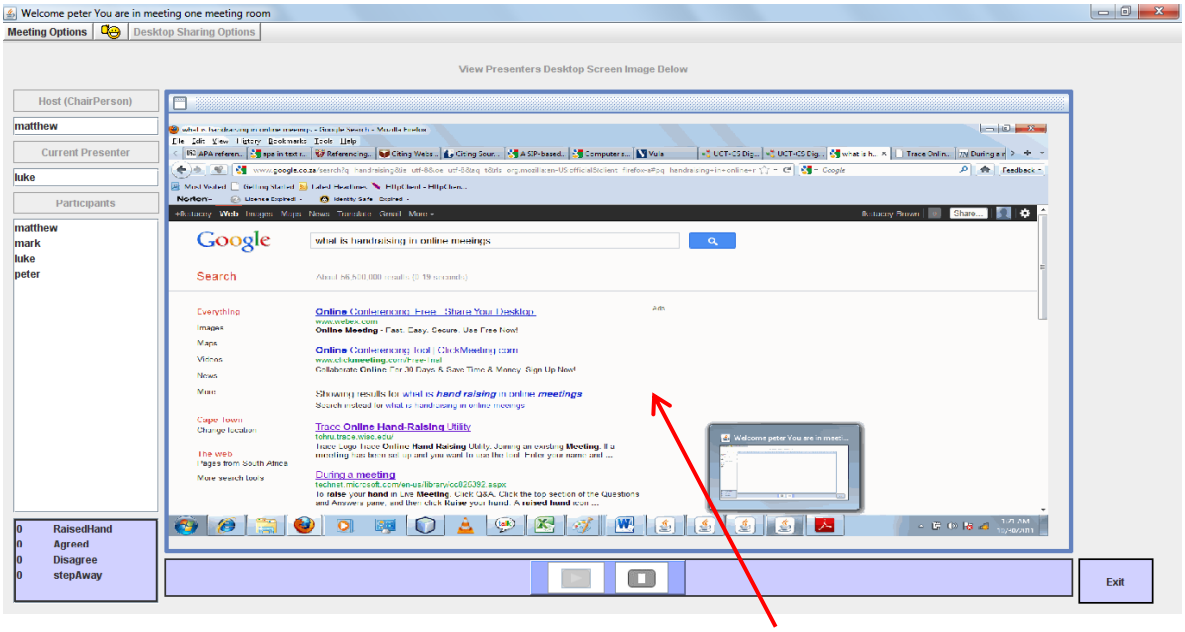


Figure 17: Viewing the presenters desktop

3.2.8 Exit application

To exit the application, the user's name would be deleted from the list of users on the system at the server side and the server terminates its connection to the user. In addition, the meeting would be searched and their names removed from the meeting objects participant list. The meeting timestamp would be updated, notifying all users of this change. As a result users would get updated lists without that users name on it.

3.3 Iterations

The software was built using an iterative approach. Three iterations were performed, each involving a design stage, an implementation stage and an evaluation stage. The first iteration was to ensure that the concept was feasible and that the technology and tools required to build the system were available. The second iteration produced an intermediary system which more or less looked like the final system. The third iteration was an improvement and optimization of the second iteration and produced the final prototype.

3.3.1 Iteration 1

The main aspect that had to be tested was ensuring that a screen image could be captured, transported and displayed at the other client's end. The participant list was not a major concern at this point as it would involve the transfer of simple strings across the network. The prototype produced had just the basic features of transmitting screen images between the client and server. The system was demoed in front of the supervisor and second reader. Suggestions on how to further implement the system was made. An image of the screen application obtained from running the code is shown below in Figure 5:

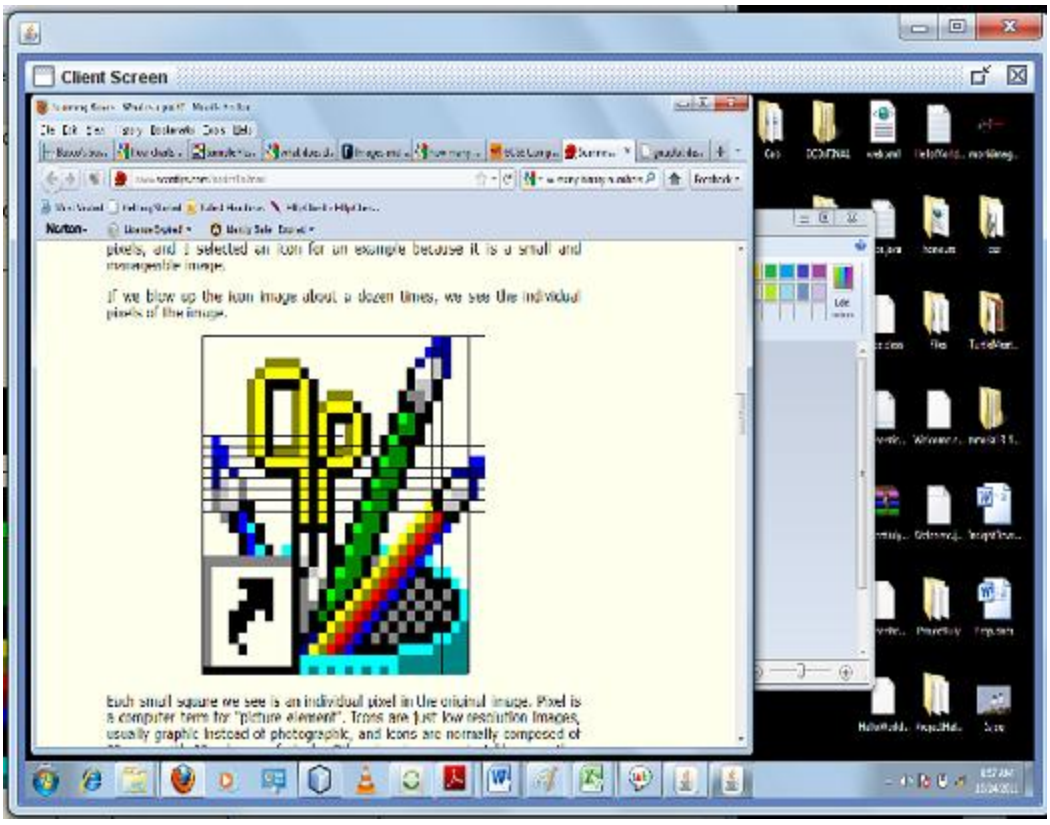


Figure 18: Initial screen image as viewed from the receiver side

3.3.2 Iteration 2

Iteration 2 saw the implementation of the participant list, floor control, polling, screen sharing and hand raising mechanisms. This involved understanding the HTTP protocol and managing all data transportations despite its limitations. However, the screen sharing feature had not yet been optimized for bandwidth by performing image differences. Work at this stage had begun on the implementation of the image subtractions; however, this was later left for the third iteration. This was due to time constraints and the need to produce a user testable prototype as soon as possible. Therefore more focus was put on adding features onto the system and refining those features.

The prototype was tested for usability using eight people, two groups of two and one group of four. Four students were from the computer science department and the other four from the engineering departments at UCT. Initially it was intended to be tested by a group of two, three and four different users; however, one user requested to be excused. Suggestions from the first and second group were implemented as they tested the prototype earlier (Two users and four users). The last group tested rather late and we were only able to implement a subset of their suggestions as time did not permit further work on the look of the application. Additional suggestions are mentioned in future work. A few of the changes done in this section are illustrated in the images below.

The first image shows a view of the very first GUI window made for the system. All clients were run on the same machine and as a result the desktop image created would appear reflected back onto the screen, this is normal behavior if the screen snapshot is being taken from the same machine on which the

receiving client is running on. The screen sharing application operates normally when users run it on two different machines. The first screen shot shows that the hand raising and all meeting data were available for the user to see at a go. However, users suggested this was a bit distracting and it would help if a menu was used instead that would hide these features and only reveal them when required.

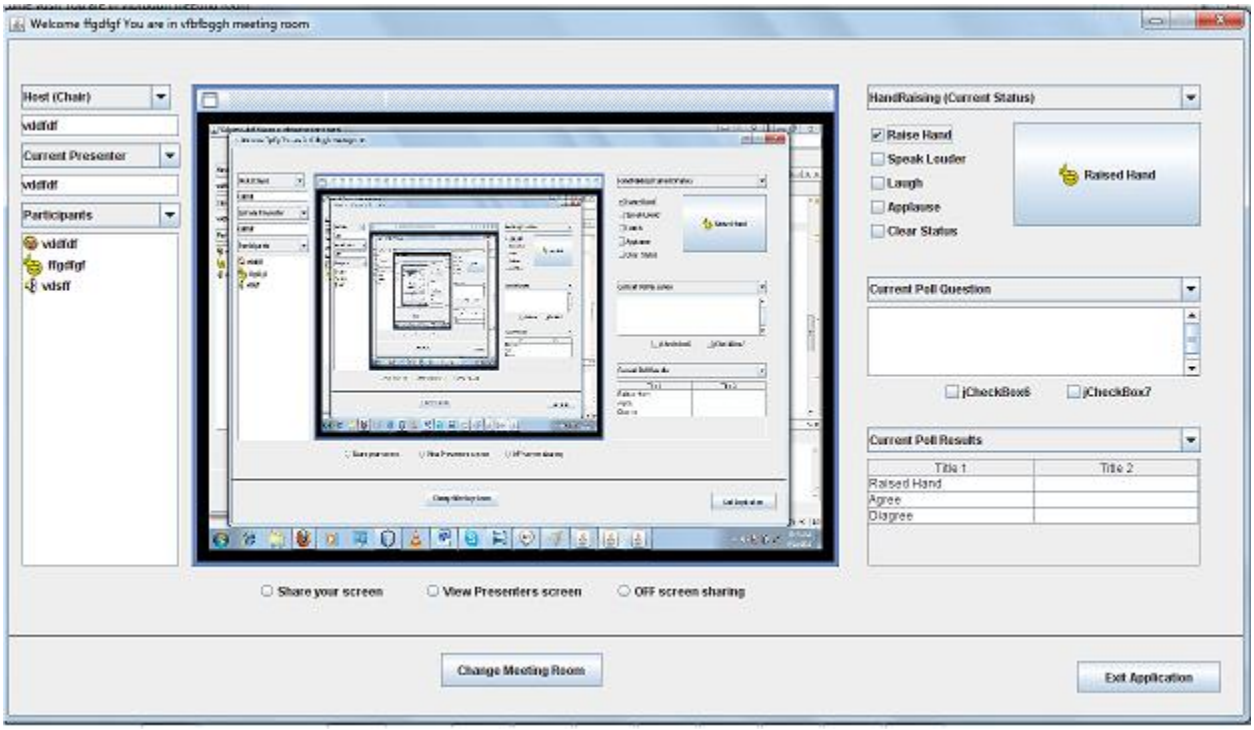


Figure 19: What the integrated system initially looked like

This brought about the second prototype shown below. Each feature was added to a menu at the top of the window. The icons in the hand raising menu bar would switch from enabled to disabled depending on the icon was selected. Users who raised their hands were arranged in order of who raised them first. A statistic of the number of activities occurring in the meeting was added at the bottom of the participant list.

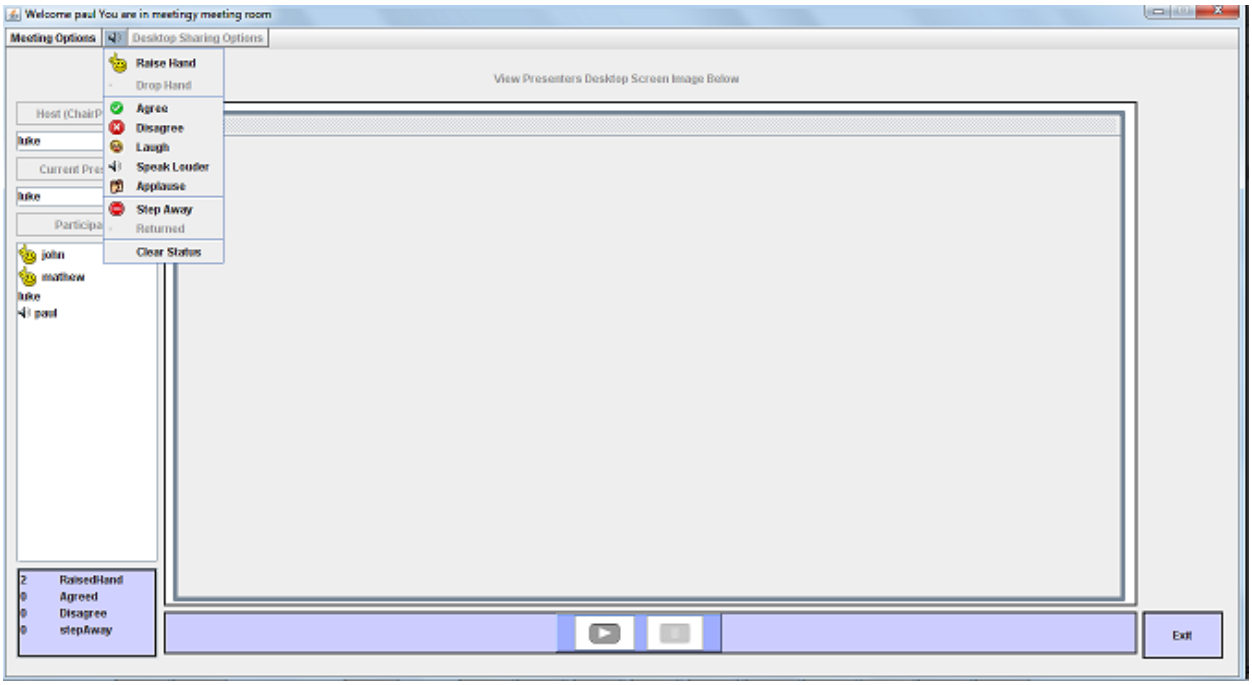


Figure 20: Improvement of previous version with the features shifted onto the above menu

This image shows the polling feature. Polling question were set to be more than ten characters long in order to ensure proper understandable questions were being asked. If a user enter less characteristic or did not choose the options to the poll question, a message box would pop up notifying them. Users suggested that it was better to have the notification text inside the text field of the poll question so that user is aware of this restriction right from the beginning.

In addition to the above, some users requested that the hand raising menu be put back on the side as it was easier to access from there. They felt it was a bit cumbersome for them to always go to the menu bar, scroll down to select the option they wanted and then click on it. They preferred the jCheckbox approach where they clicked once on the item and it displays their emotions. In addition, some said that the current location of the hand raising menu (menu bar at top of window) was not obvious to someone who had never used a video-conferencing system before, that this menu contained icons that could be used to express themselves during a meeting and therefore would miss out on the opportunity of using the hand raising menu at all.

3.3.3 Iteration 3

Iteration three was meant to be an improvement of iteration two However, due to time constrains, not all the suggested improvements were implemented. In addition, as this was not intended to be a totally complete product, the final iteration was more focused on optimizing the code to work with low bandwidth as this was the main purpose of the project in the first place. So more effort and work was put into optimizing the screen sharing feature and trying to add bandwidth management and control structures into the system. User testing was aimed at determining responsiveness and the user's experience with the system. The final prototype looked very similar to image 2. Nothing much changed on the front end as most of the work done in this section was more focused on the back end of the program.

Chapter 4

Experimentation and Evaluation

4.1 Introduction

This chapter assesses the system in terms of responsiveness and the user's experience interacting with the system. Additional tests are conducted in order to determine the bandwidth requirements of the individual components of the system and eventually the system as a whole. This section starts with the user testing, followed by five different experiments that were conducted to evaluate the bandwidth requirements of the system. The outline of the chapter is shown below:

- User Testing
- Performance Evaluations
  - Experiment 1: Bandwidth required for desktop sharing only
  - Experiment 2: Bandwidth required without desktop sharing
  - Experiment 3: Bandwidth required for a normal meeting
  - Experiment 4: Bandwidth required in an extremely active meeting (Stress Test)
  - Experiment 5: Bandwidth required without image differencing

All the experiments conducted in the Evaluations section had three clients and one server. Each experiment lasted for three minutes. In order to minimize the experimental error and increase the accurateness of experiments, each experiment was repeated four times. Data was obtained through logging the time and number of bytes transferred within the program using println statements as the program executed. This data was stored in a log file that was later transferred to a spreadsheet in order to produce the corresponding graphs. The following section is on user testing, followed by the performance evaluations.

4.2 User Testing

In order to evaluate the responsiveness and end user experience of the system, user testing was carried out. The program variables that resulted in an acceptable user experience and acceptable responsiveness could be determined. These variables could then also be set and used when evaluating the overall bandwidth required by the system. The program variables specifically referred to here are the ones responsible for the refresh rate of data sent from the client to the server and from the server back to the client.

Two groups of six individuals participated in the experiments, each experiment lasting for forty five minutes. The experiments are described in detail below.

4.2.1 Methodology

An update rate of one image per second for the desktop sharing application and meeting processes was used. Therefore images and new notifications would be updated every second. This would enable users to get responses from the system in near-real time. An update rate of two screen images every second produces an almost continuous stream of desktop images. This would be able to capture even the slightest changes in the desktop and make it visible to users. However, dealing with low bandwidth, an update of one image per second or one event update each second was assumed to be adequate to give an acceptable

experience with the software. However, to test this theory, users had to test the software and give feedback on its responsiveness.

The tests carried out were simple and straight forward as only the responsiveness and the user's experience with the technology was what was required. The experiments were designed such that the users were guided along and given tasks to perform.

For the first group of six users, initially one user would log on, interact with the system and observe the responsiveness of each feature on the system. A second user would log on when instructed to; the two would then interact with the system and check for responsiveness. Users would progressively log on and rate the system responsiveness and their experience. Once six users were reached and had interacted simultaneously with the system in one meeting with one another, they were instructed to gradually log out in a first in first out manner and those that remained on the system would observe any changes as interactions continued.

After the above was complete, the ability of the system to support multiple meetings was tested with the same group of six users. Here they were instructed to log onto two different meetings and carry out interactions to see if meetings would interfere with each other. Questionnaires were provided at the beginning before testing and users were asked to fill in their feedback in the provided questionnaires as the experiment occurred. They were also asked to indicate the number of people that were already logged onto the meeting before they did. A sample of the consent forms and questionnaires are shown in Appendix A and B respectively.

The second experiment on the second group of six users was done in reverse chronological order to the first. Here, initially users were instructed to all log onto the system at once and enter into one meeting room. They then interacted with the system and observed its responsiveness. Then, one by one, users were instructed to log out of the meeting and those who remained tested for any changes in the system responsiveness. Once one person was left in the meeting room, users were asked to log back in gradually again, starting from the first one who had been instructed to leave the meeting. This was done until all the six users were back again onto the system and any noticeable changes were recorded. Again the two meetings scenario was conducted to test the system support for multiple meetings.

This experimental approach was chosen in order to reduce bias and balance out the experiment. Users were able to experience different arrangements in the time and number of people available when they logged in. Their feedback on their experience with the system was recorded in the user questionnaires.

4.2.2 Experiment set-up

The experiment was set up and conducted in the honours labs on level three of the computer science building. Seven computers were used; six had the client program installed on them and one had the server program running on it. The server ran on a Windows machine while the clients were all installed on Linux machines. This was because the Linux machines had the preinstalled libraries that supported the software to be tested whereas the Windows machine didn't and everything had to be installed. In addition, getting administration rights to install programs on all the Windows machines would be a bit time-consuming.

An experimenter was always with the users to give instructions and tasks to perform and complete. In addition to observing what was happening, the experimenter was also responsible for answering any questions that users had to ask. The labs were generally quiet as most people were not in the labs.

4.2.3 Participants

Participants were selected randomly from students in the Computer Science Department at the University of Cape Town. A total of twelve users were tested. Five of these were females ranging from the age of Twenty two to twenty four years and in various years of study. Three of these had never used a video-conferencing system before. The rest were males ranging from twenty two to thirty years. Only one of them had never used a video-conferencing system before. Users varied from the Faculty of Science to the Faculty of Engineering, all were computer literate and in their 1st to 4th year of study for their various degrees.

4.2.4 Results

The following points provide a summary of the feedback results that were obtained from the user testing. The following results were obtained:

- Most of the users were happy with the system and found it very useful. Eleven of them said they would use the system if it were to be deployed.
- User's loved the handraisng feature and suggested the addition of more icons to represent additional emotions.
- On a scale of one to five in terms of responsiveness, most of the users found the system responsive enough and they felt they could participate easily and felt they were part of the meeting.
- Only one user indicated they noticed a small change in the performance as more users were added.
- Six users felt they would prefer fast clear quality images when the bandwidth is good and no image when the bandwidth is bad. Five users felt getting a consistent rate of image regardless of bandwidth was much better. One suggested they would love an adaptive combination of the two between fast and clear and constant and poor when bandwidth reduces.
- When asked how to improve the system, four users suggested they would like to see a more improved and faster screen image quality. Another suggested a more interactive environment and being able to add personal work details such as contacts, location, degree etc. One user suggested adding a Help database for the system when combined with the other modules.
- Compared to another system, one user suggested that this system had more features than the previous system they use.

4.2.5 Discussion of results

The feedback indicates that the system could be very useful; however more work is needed especially on the desktop sharing application. In addition, making the system more interactive by letting users add features such as their contacts and work addresses could proof useful. The responsiveness of the system is great and it provides an acceptable end- user experience.

During user testing, a lot of the users advised on improving the system by getting faster good quality screen images. The differencing images used in the reconstruction of the initial desktop sent by one client to the server were analyzed. Multiple reconstructed GIF images were compared and it was found that the image quality from the reconstructions did not degrade. Therefore this was excused at the reason for the poor image quality. Two other reasons were looked at. For starters, users were stationed at different computers that had different screen dimensions (wide screens and narrow screens). An image transferred from a client using a narrow screen computer had to be elongated to fit on a client using a wider screen computer. This could be one contributor to the problem. In addition, this image then had to be redrawn on

the jPanel window for that user. The jPanel window did not have the same dimensions as a normal user's desktop dimensions and was smaller. The reconstructed desktop image had to be redrawn to fit the jPanel dimensions. This could be another reason for the course of slight image blurriness.

Another issue looked into was speed. The images were set to update every second. However, images displayed on the clients seemed to take longer than the second to update. Therefore consecutive actions would appear as freezing images. To account for this delay, an analysis was done on the image difference processing to see if it was a possible cause for the bottleneck in the system. The time from when a desktop image was taken on the client to when it was processed and ready to send was recorded. These times were recorded for five minutes of desktop transactions and the average difference computation time was calculated from the values obtained. On the server, the time taken for the reconstruction of the differences to form the new image was calculated, as images on the server had to be updated before sending differences to other clients.

It was found that the Java Robot class took a negligible average of 110.5556 milliseconds to capture the screen image. The screen sharing was left to run for five minutes. After five minutes, it was found that the client took an average of 3674.893 milliseconds to compute the two image differences. The server took an average of 4538.615 milliseconds to compute the reconstructed image. These figures were obtained from logging data from the server for the five minutes and computing the average of all the data collected. The above figures indicate that an image was sent every three to four seconds to the server as compared to the one second set on the system. In addition the image would need a total time for sending the screen image from client1 to the server and the server processing time and client2's processing time, assuming client2 takes the same amount of time to compute the reconstructed image as the server. The calculation is shown below:

- Time to compute image at client1 = 3674.893 ms
  after 3674.893 milliseconds the image differences can be sent to the server
- As explained in the implementation chapter, the image is reconstructed at the server so that late users can receive this initial image before receiving differences.
  Time to reconstruct image at server = 4538.615 ms
- Time to reconstruct image at client2 (similar to that of server) = 4538.615 ms
  Therefore the time it takes for an image to travel from client1 to client2, ignoring network latency, in this case is:
  $3674.893 + 4538.615 + 4538.615 = 12752.123\ milliseconds$

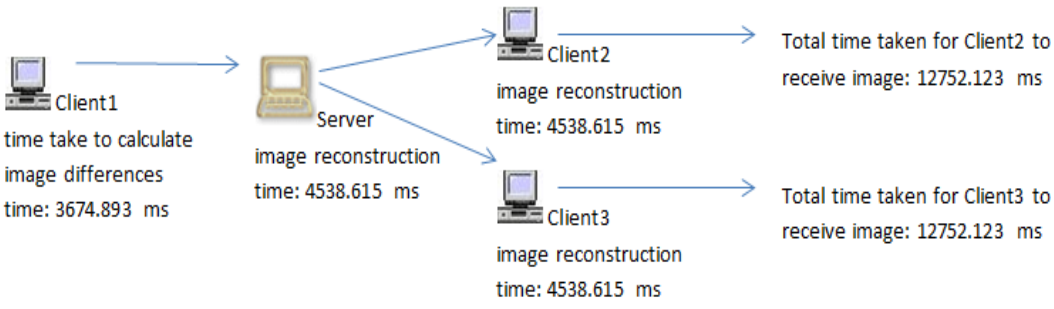


Figure 21: Time delays in image transmission

The results exclude the time needed for network transfer in the network. This is just the time taken in computations from client1, the server and client2 just before it is redrawn and displayed on client2's

jPanel window. The time taken to redraw the images on the jPanel is negligible as was proved when the system was tested without the image differencing. Therefore one can assume that the processing time could be the reason for the delays in the image reception. In addition this could account for why the users did not notice much change in the responsiveness of the system especially the screen sharing, since updates took time. However, this cannot be assumed as the cause for the blurred image quality as the image quality was maintained during all the image processing.

These results indicate the different opinions of users on the responsiveness of the software and their experiences using it. However, the results of the testing are not irrefutable as the number of tests carried out could have been increased. In addition, users in different locations could have been found and requested to carry out the meetings and test the system in real life. Lastly, had time permitted, a low bandwidth environment could have been simulated for the selected users to test the system on.

4.3 Performance Evaluations

A series of additional tests were conducted to determine the overall amount of bandwidth needed by the system. This section evaluates the results of these subsequent tests and is divided as follows:

- The bandwidth required by the desktop sharing application alone is evaluated.
- The bandwidth required by the handraisng feature is evaluated.
- The bandwidth required by the system as a whole during a normal meeting is evaluated.
- A stress test on the whole system is conducted and the results evaluated.
- The bandwidth required by the desktop sharing application alone without image differencing is evaluated and compared with the results of experiment1.

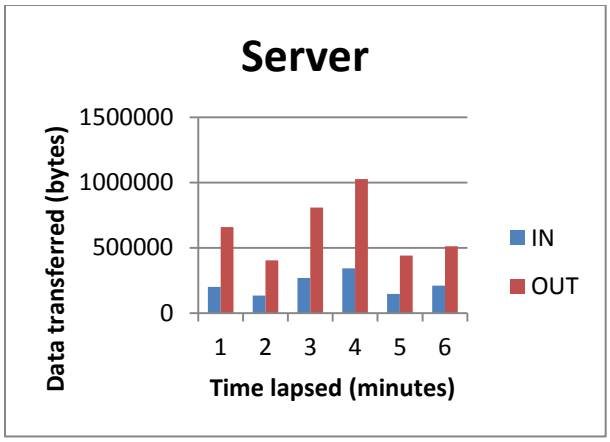
4.3.1 Experiment 1: Bandwidth required for desktop sharing only

4.3.1.1 Purpose

The aim of this experiment was to determine the amount of bandwidth consumed by the desktop sharing feature alone. This would determine how much bandwidth is needed at the server side and how much bandwidth is needed for a user to be able to share their screens.

4.3.1.2 Procedure

A series of four meetings were run for five minutes with three users. One user (presenter) shared their screens as the two others viewed this image on their screens. In order to get feedback on what exactly the others were seeing, the presenter also received back a stream of their screen images from the server. The window displaying these images at the presenter's side was minimized and kept at the corner of the screen for occasional feedback and in order not to interfere with other activities. Therefore, the presenter's image was being sent to the server and the server was sending data back to the three users. The first experiment had the presenter demonstrate software to the other clients for three minutes. The second involved typing text on a document, the third involved sharing a video to the other clients and the last involved finding and demonstrating an online website. The time and number of bytes transferred by each user and the server was recorded onto a log file. The average for the four tests was computed, and these were then transferred onto a spreadsheet and graphs obtained from the results. Before the four meetings were conducted, a previous test had been completed. The results are shown below. These demonstrate that for every packet sent to the server, the server has to send out that packet to three different users, as seen in the graph below.



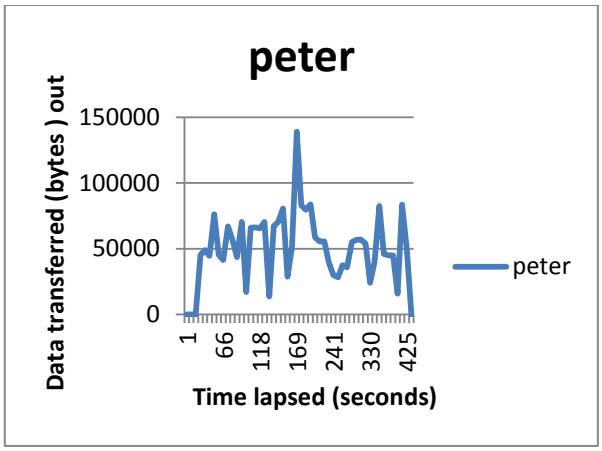
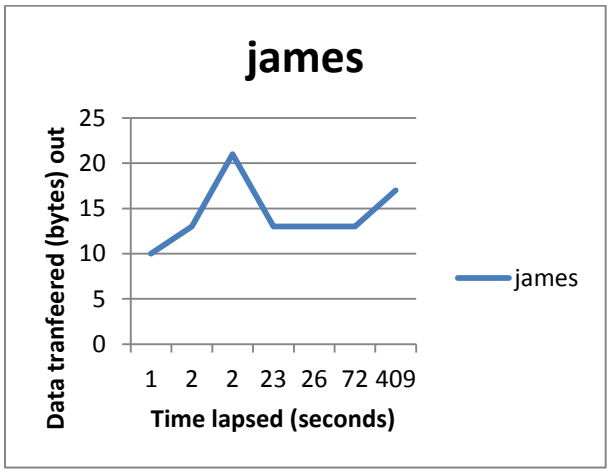
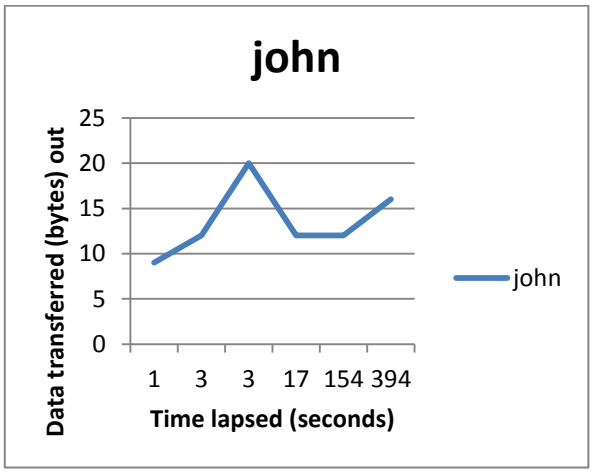


Figure 22: Data transfer for the server and Peter during screen sharing

The screen sharing application is the most bandwidth consuming feature in the system. It can be observed from the graphs above that Peter's outgoing bandwidth is relatively larger compared to the other users (James and Johns graphs are below) because he has to send his desktop image every second. On the other hand the server has to send these image bytes to the other two users including Peter, so the bandwidth almost triples on the server side. An interesting thing to note are the peaks on Peters graph which correspond to sending an entire image instead of the differences across to the server. This occurs in the case where the combined computed differences are larger than the image itself. This happens when two very different images are subtracted from each other. The dips in the curve correspond to small image differences especially when the users screen isn't changing much. James and John (graphs below) have similar usage patterns. This is because they are both viewing the desktop and only need to send one request to the server for images, after this all they do is receive screen images and therefore do not need much outgoing bandwidth.





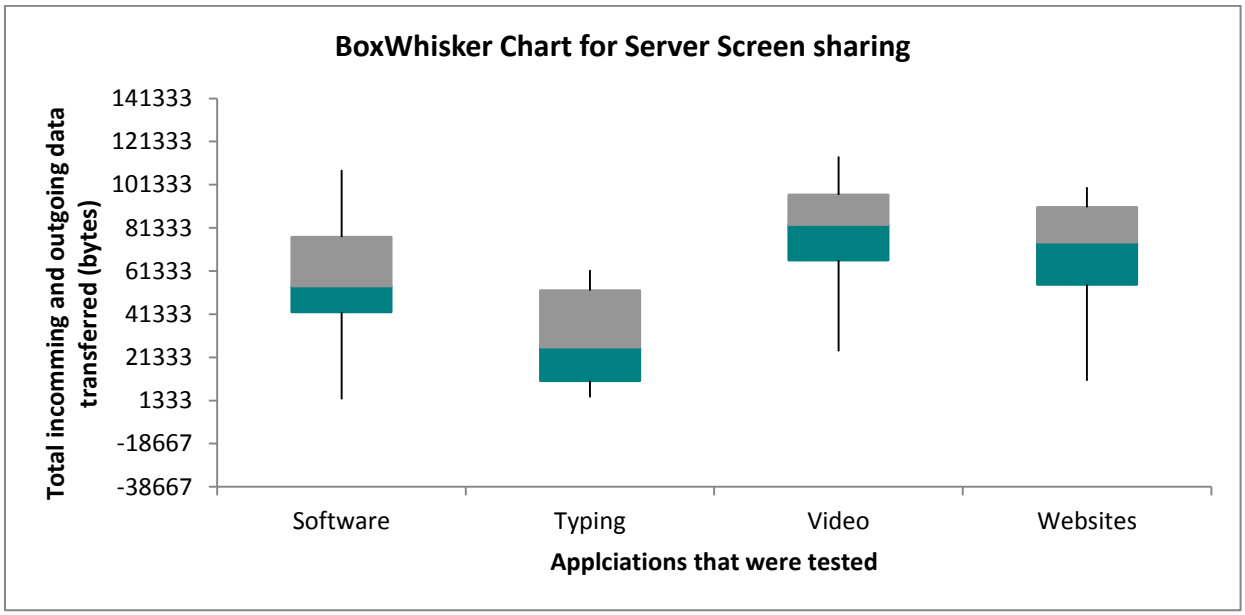
Total server Side Bytes sent in 5 minutes	
James John Peter	2515047 2279614 57
Total server Bytes (Outgoing)	4794718

Individual client Side Bytes sent in 5 minutes		
Peter	Total Bytes (Outgoing)	2516423
James	Total Bytes (Outgoing)	100
John	Total Bytes (Outgoing)	81

Figure 23: Graphs of data transfer for James and John during screen sharing

4.3.1.2 Results

The following box and whisker diagram shows the total incoming data received at the server side for the different applications within the five minutes of testing. From the diagram, no outliers were indicated and of spread of data seemed more or less even. For the software sharing application and typing, most of the data was located above the median, whereas the spread of data in screen sharing and websites were nearly equally distributed. Typing seems to consume the least bandwidth, followed by software, websites and finally video.



The data was combined in order to obtain the average bandwidth used by screen sharing for all the applications combined. The histogram below shows the spread of data and provides additional information into its distribution.

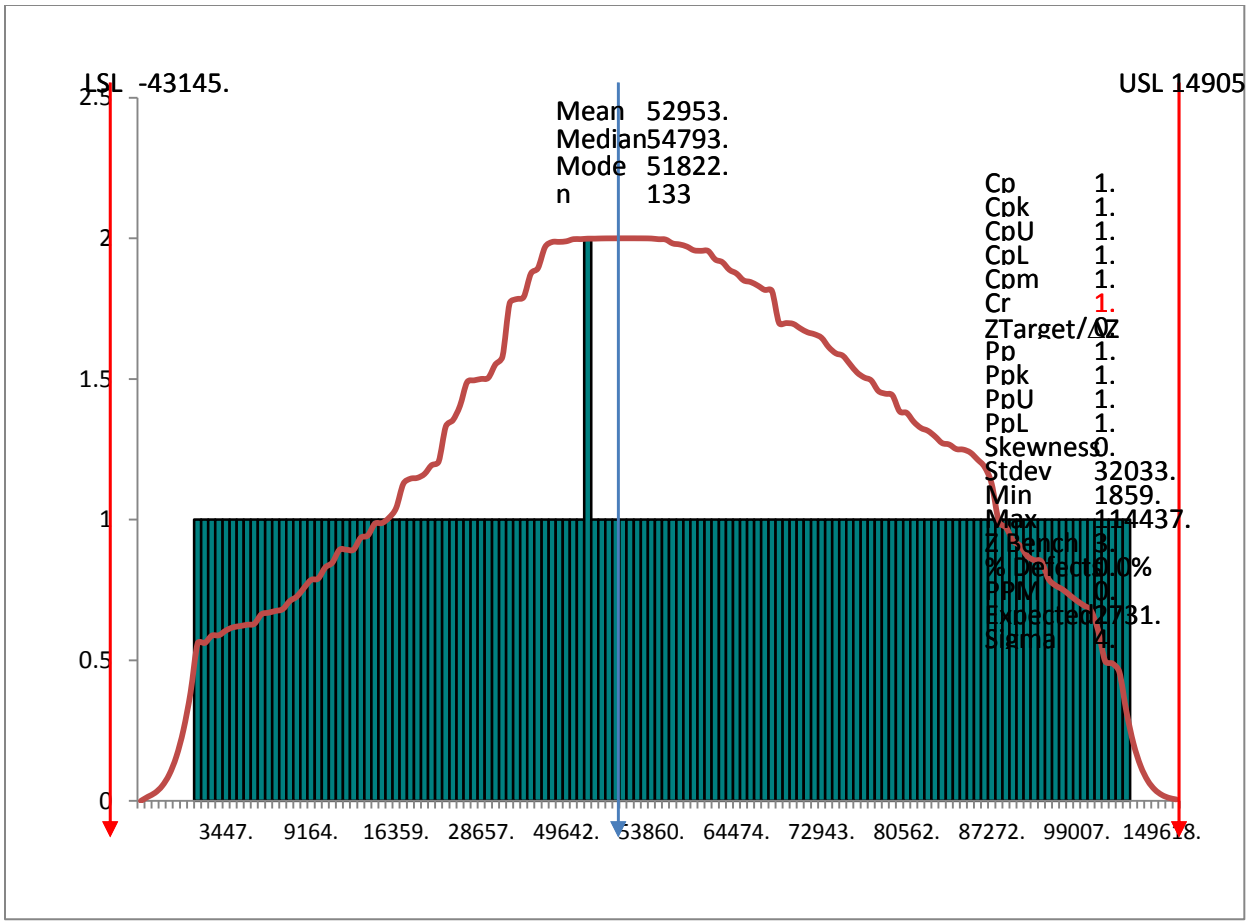


Figure 24: Distribution of data from the 5 screen sharing experiments combined

The average amount of data consumed by the screen sharing application for five minutes in a meeting with three users was found to be 52953 bytes with a standard deviation of 32033 bytes. The curve shows that the data is evenly distributed and not skewed. The largest value obtained being 114437 bytes (most likely a whole image) and the lowest 1859.

4.3.1.3 Discussion

The average data obtained for screen sharing was found to be 52953 bytes for the five seconds. As mentioned, the system sends data or desktop image updates every second. This means that the bandwidth required for transactions on the system is approximately 52953 bytes every second. However, this result includes both the incoming and outgoing data combined on the server side assuming that for every incoming package the server has to send transmit this data three times to the three users.

4.3.2 Experiment 2: Bandwidth required for handraisng

4.3.2.1 Purpose

The purpose of this experiment was to determine the amount of bandwidth consumed by the application with the desktop sharing feature turned off. This would determine how much bandwidth is consumed by these features.

4.3.2.2 Procedure

The meeting durations and number of tests were reduced in this second experiment due to time constraints. However three experiments with three users each lasting for three minutes were conducted testing the various sections of meeting procedures. The first one had all the users randomly use the hand raising menu only. The second involved polling and the presenter randomly sent polls for users for votes. The last one tested involved testing the floor control mechanism. Here users just exchanged roles (host or presenter) between them as the meeting progressed. Again data was logged into a file, transported to a spreadsheet and graphs plotted.

4.3.2.3 Results

The following box and Whisker graph was obtained.

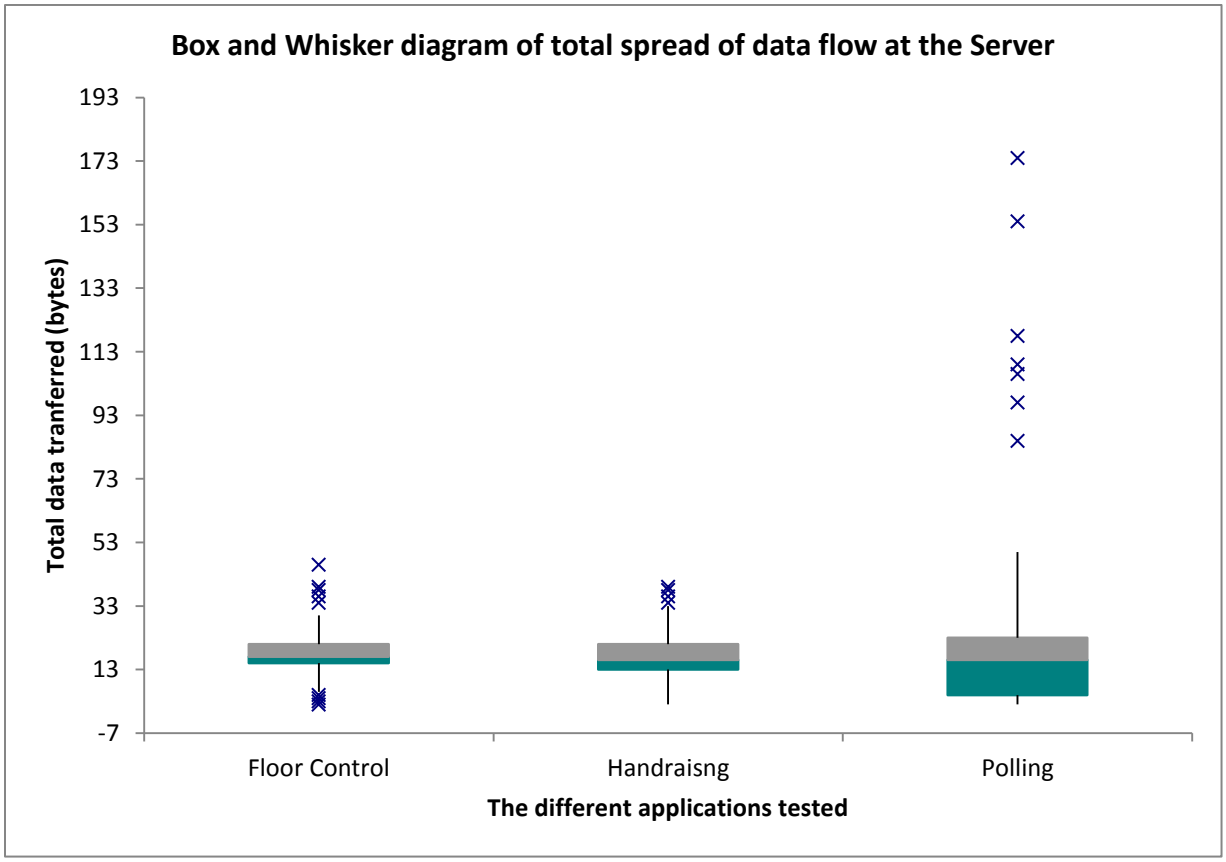
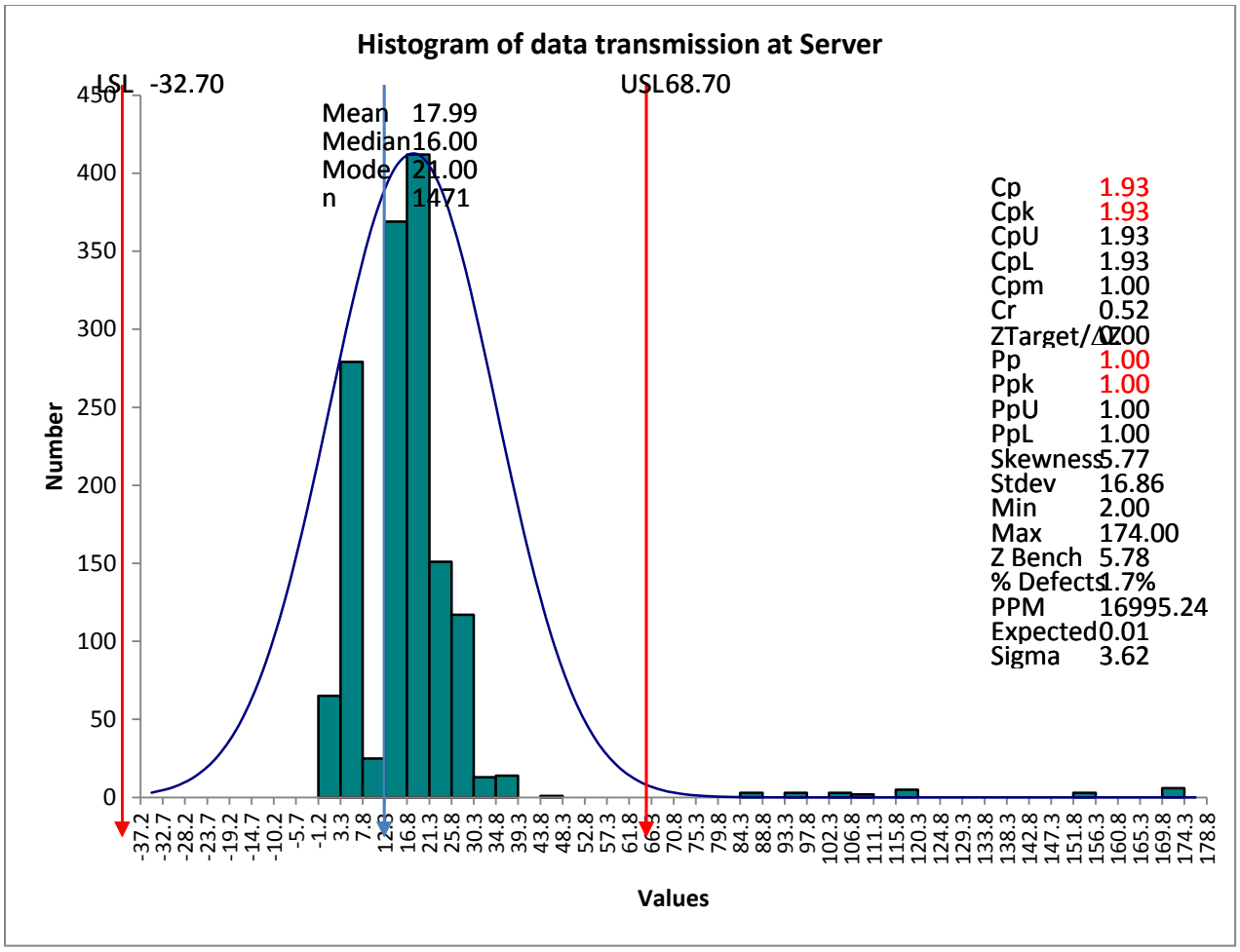


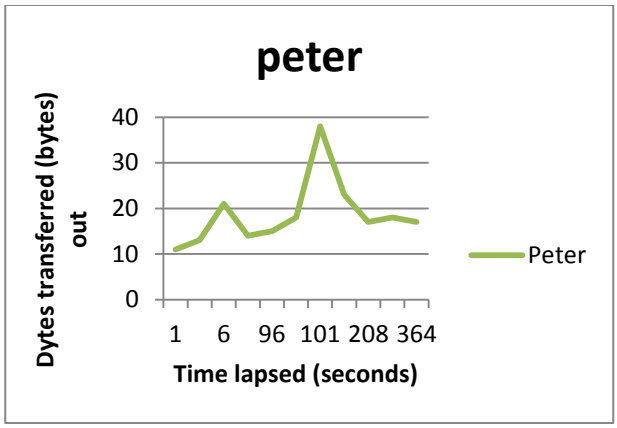
Figure 25: Total spread of dataflow at server

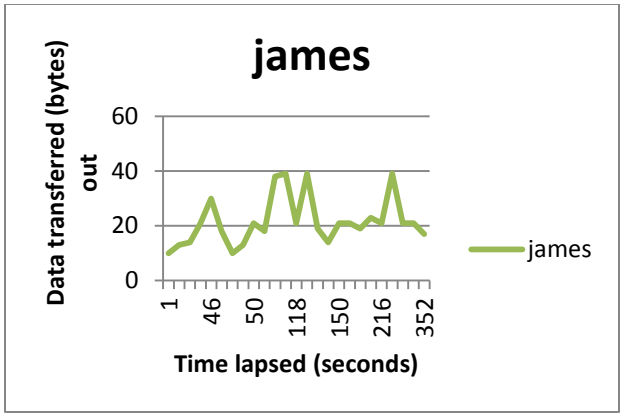
Compared to the first set of graphs from experiment 1, one can clearly notice the decrease in the bytes transmitted by looking at the range of the y-axis. All the different applications had the occasional outlier's; however, these were very small values. The distribution of data in the floor control and handraising application seems to be equally distributed. Polling has more data slightly towards the lower quartile. The data was then combined and a histogram plotted in order to obtain the overall average value of the total bytes transferred. The histogram is shown below:

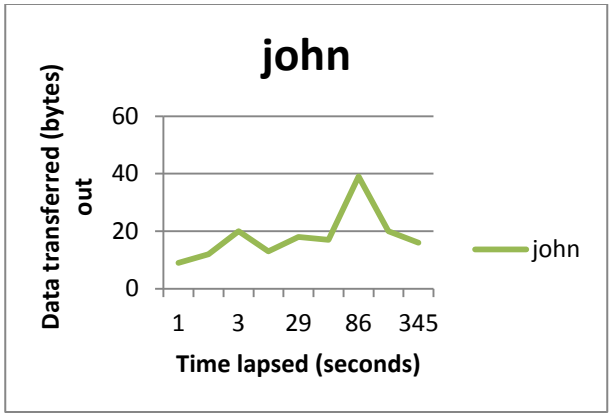


The histogram shows that data is slightly skewed to the left with a skewness value of 5.77. The minimum is 2 and the maximum value obtained was 174. This is negligible compared to the screen sharing case where the minimum was 1859 and the maximum 114437. In addition the standard deviation is 3.62 with a mean of 17.99 as opposed to 32033 and 52953 for the screen sharing case. Therefore the bandwidth required by the server to entertain a meeting of three users for 3 minutes without sharing their desktops is an average of 17.99. Assuming data is sent every second then this will be 17.99 bytes per second. The graphs below show the flow of data at the server and individual client end during screen sharing. As can be seen, all the users have similar bandwidth usages as compared to screen sharing and the server sends data to all the users in a similar pattern.

Graphs of Bytes against Time (seconds)







Total server Side Bytes sent in 5 minutes	
Peter James John	3551 3629 3514
Total server Bytes (Outgoing)	10694

Individual client Side Bytes sent in 5 minutes		
Peter	Total Bytes (Outgoing)	205
James	Total Bytes (Outgoing)	541
John	Total Bytes (Outgoing)	164

Figure 26: Bandwidth usage without screen sharing

4.3.2.4 Discussion

The histograms obtained showed that the overall bandwidth usage has drastically reduced as compared to the screen sharing only. Within the 3 minutes of usage, each application consumed less than 174 bytes. An average bandwidth of 17 bytes per second is needed for the server to support a meeting of three users for 3 minutes. The small bandwidth values enable these features to be given more priority over the screen sharing application, especially in low bandwidth conditions.

4.3.3 Experiment 3: Bandwidth required for a normal meeting

4.3.3.1 Purpose

The aim of this experiment was to determine the amount of bandwidth consumed in a normal meeting setting by both the client and the server.

4.3.3.2 Procedure

A meeting was run for three minutes using three users. All test spent approximately one minute sharing video. The following two minutes were spent switching presenters in test one (floor control,), handraising in test two and polling in test three. The following Chart was tabulated from the results.

4.3.3.3 Results

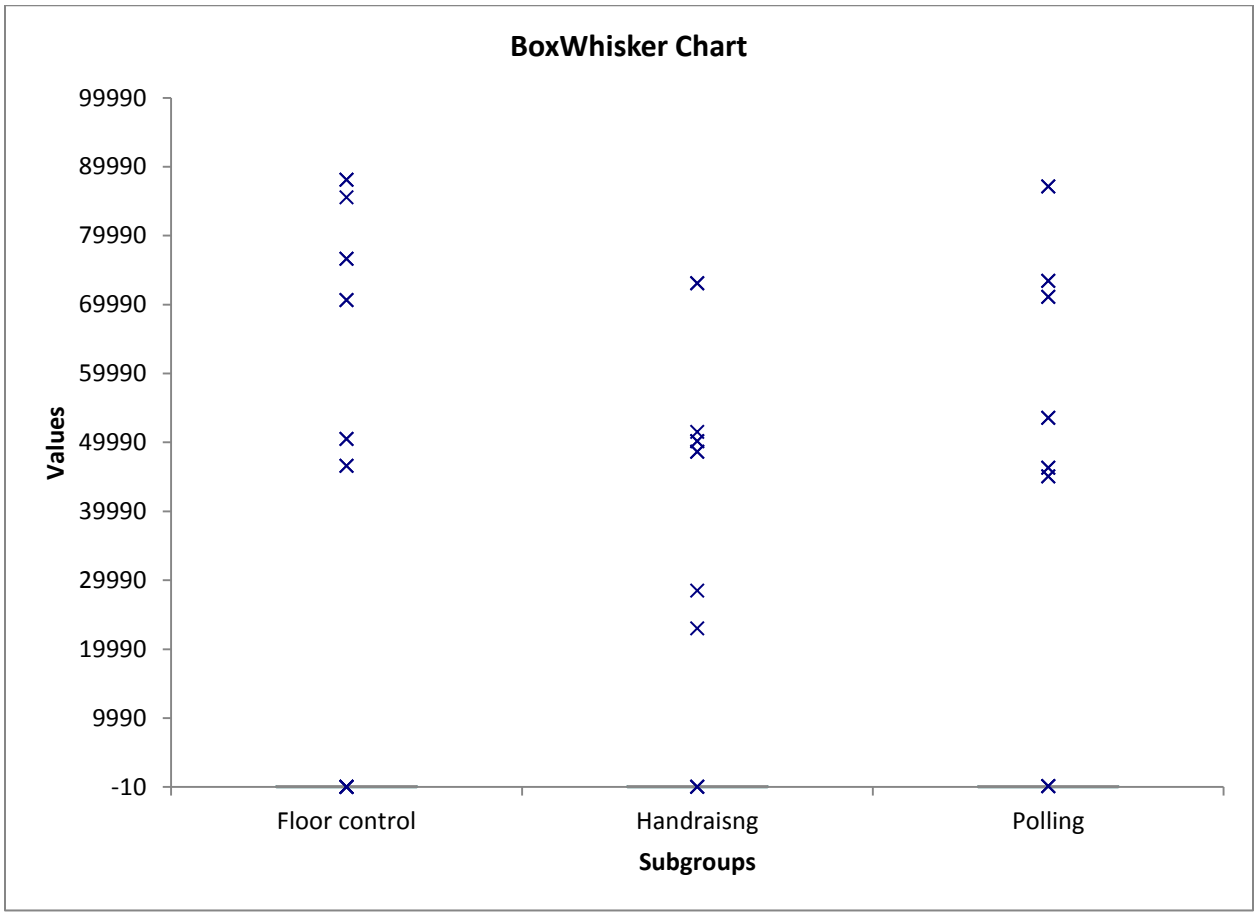


Figure 27: Screen sharing tested with various applications

The graph generally shows that most activity involved the transmission of smaller byte. The outliers correspond to the time when the desktop images were being transmitted as they have larger values. More insight into the distribution is given by the histogram below.

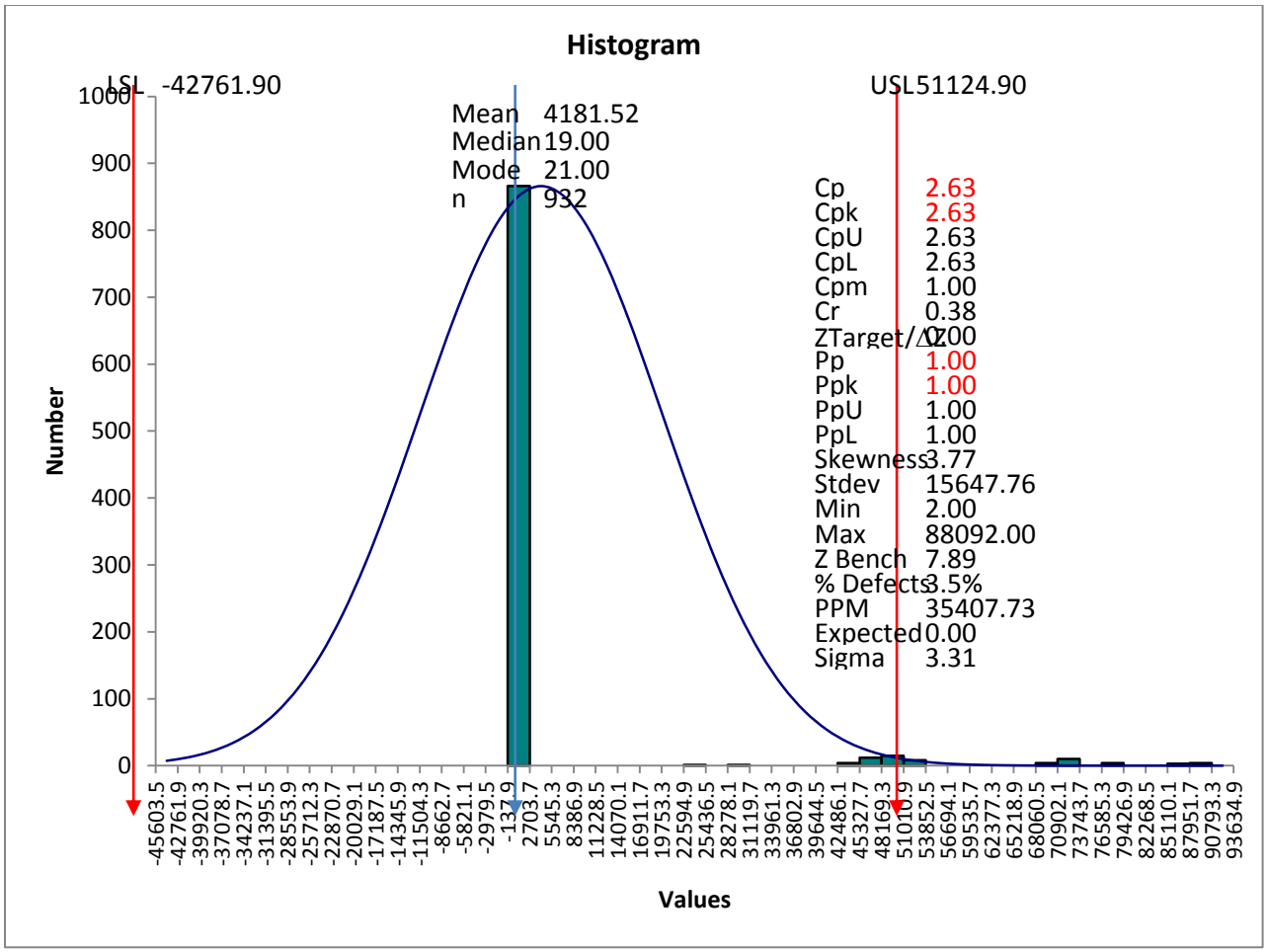
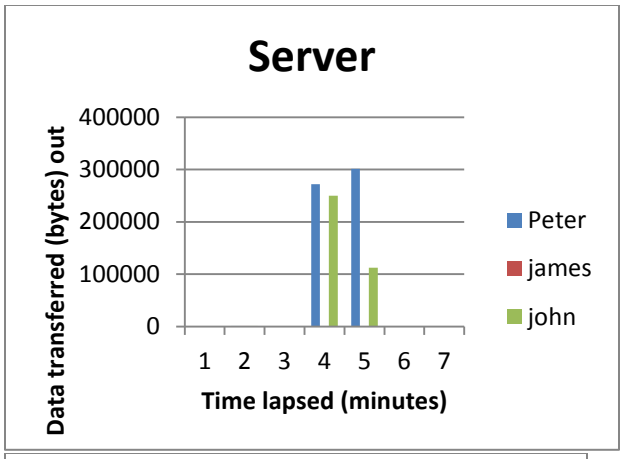
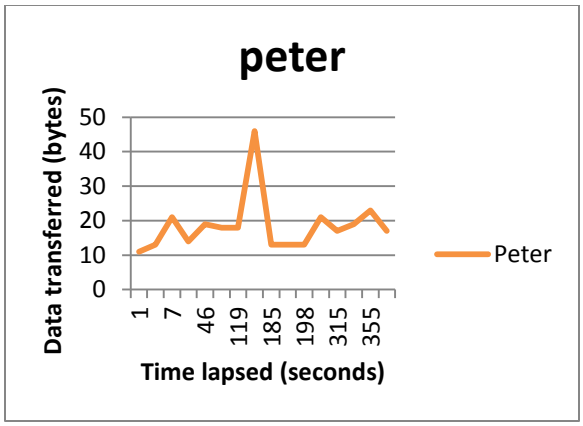
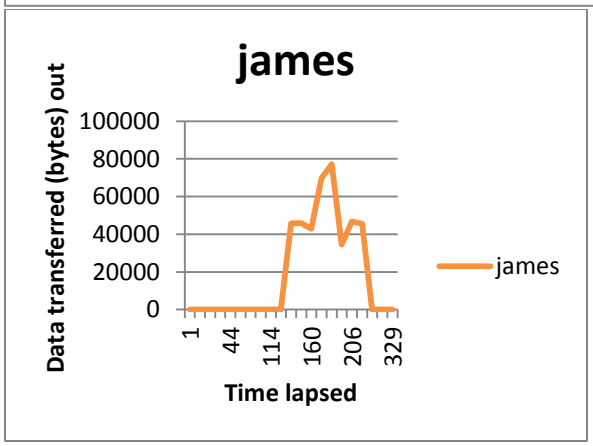


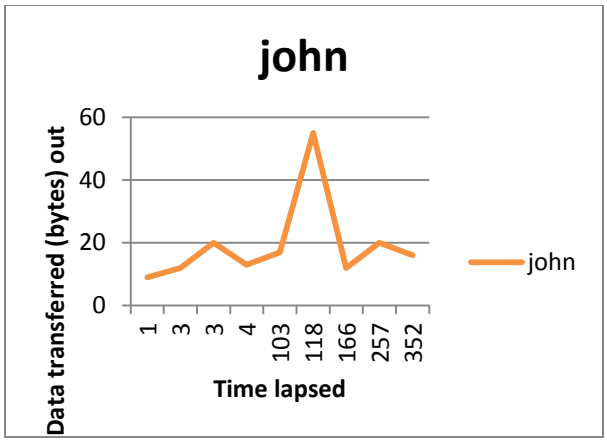
Figure 28: Screen sharing applications combined

The median obtained is 4181 bytes with a maximum of 88092 and a minimum of 2 bytes. The standard deviation is 15647 bytes indicating that data varies between small Strings and image files. This graph implies that the bandwidth required for this application was 4181 bytes per second. The following graphs of the number of bytes transferred against the transfer time were also plotted. The first graph corresponds to the server, the second three are clients. The bandwidth required by each client is shown in the Individual client side Bandwidth table and the servers total bandwidth is shown in the server Side Bandwidth's table. These are graphs to just demonstrate the flow of data at both the client and server ends.









Total server Side Bytes sent in 5 minutes	
Peter James John	575575 2517188 2643862
Total server Bytes (Outgoing)	5736625

Individual client Side Bytes sent in 5 minutes		
Peter	Total Bytes (Outgoing)	296
James	Total Bytes (Outgoing)	408784
John	Total Bytes (Outgoing)	174

Figure 29: Bandwidth required by normal meeting

4.3.3.3 Discussion

From the graphs and table, one can immediately recognize that it was probably James that was sharing his screen since he consumed the most bandwidth of the three users. It can be noted also from the server side that there could have been a period of inactivity from the users. As indicated by the times with very little or close to zero data transferred on the server.

4.3.4 Experiment 4: Bandwidth required in an extremely active meeting (Stress Test)

4.3.4.1 Purpose

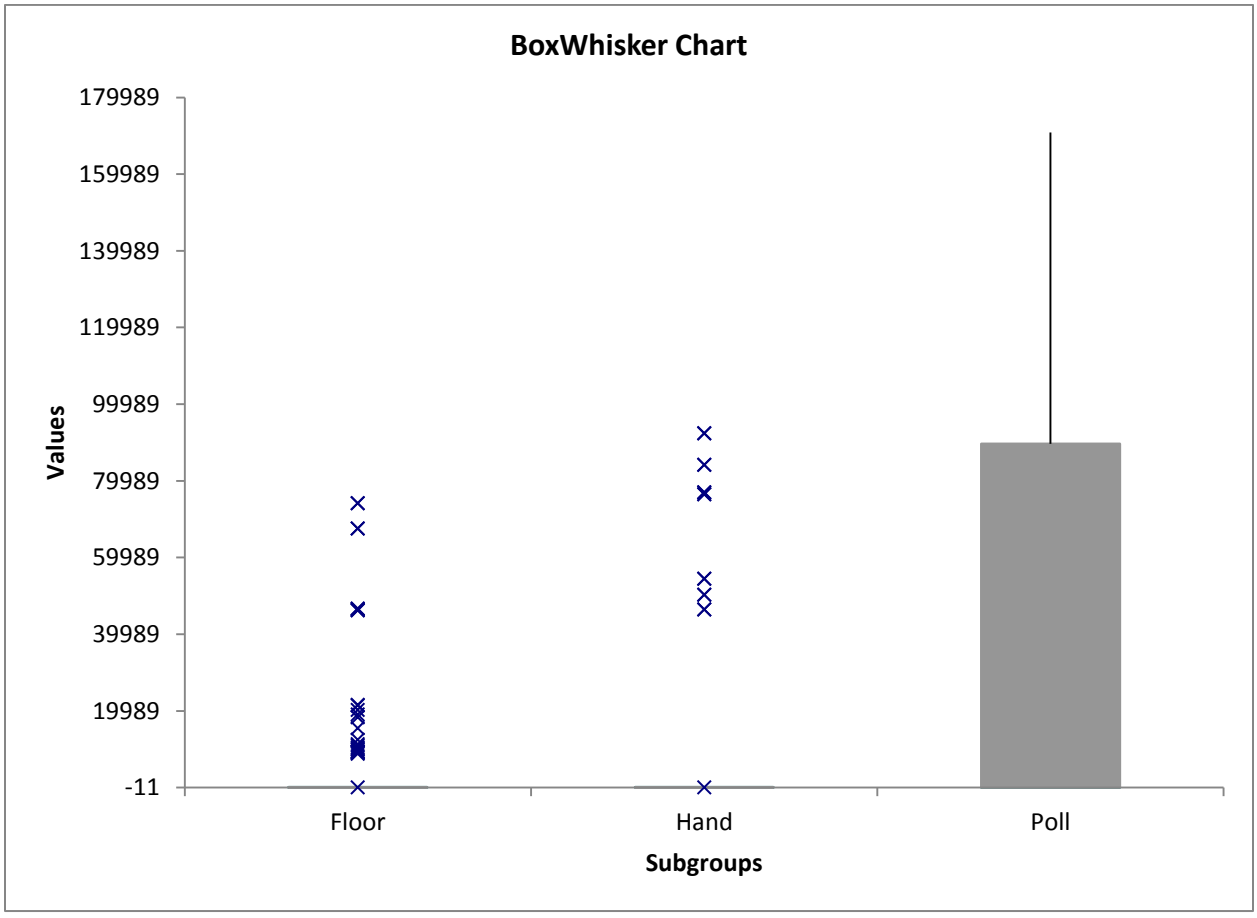
The purpose of this experiment was to determine the amount of bandwidth consumed in an extremely active meeting where all users were vigorously participating and engaging. This was a form of stress test that would help determine the maximum bandwidth required by the system.

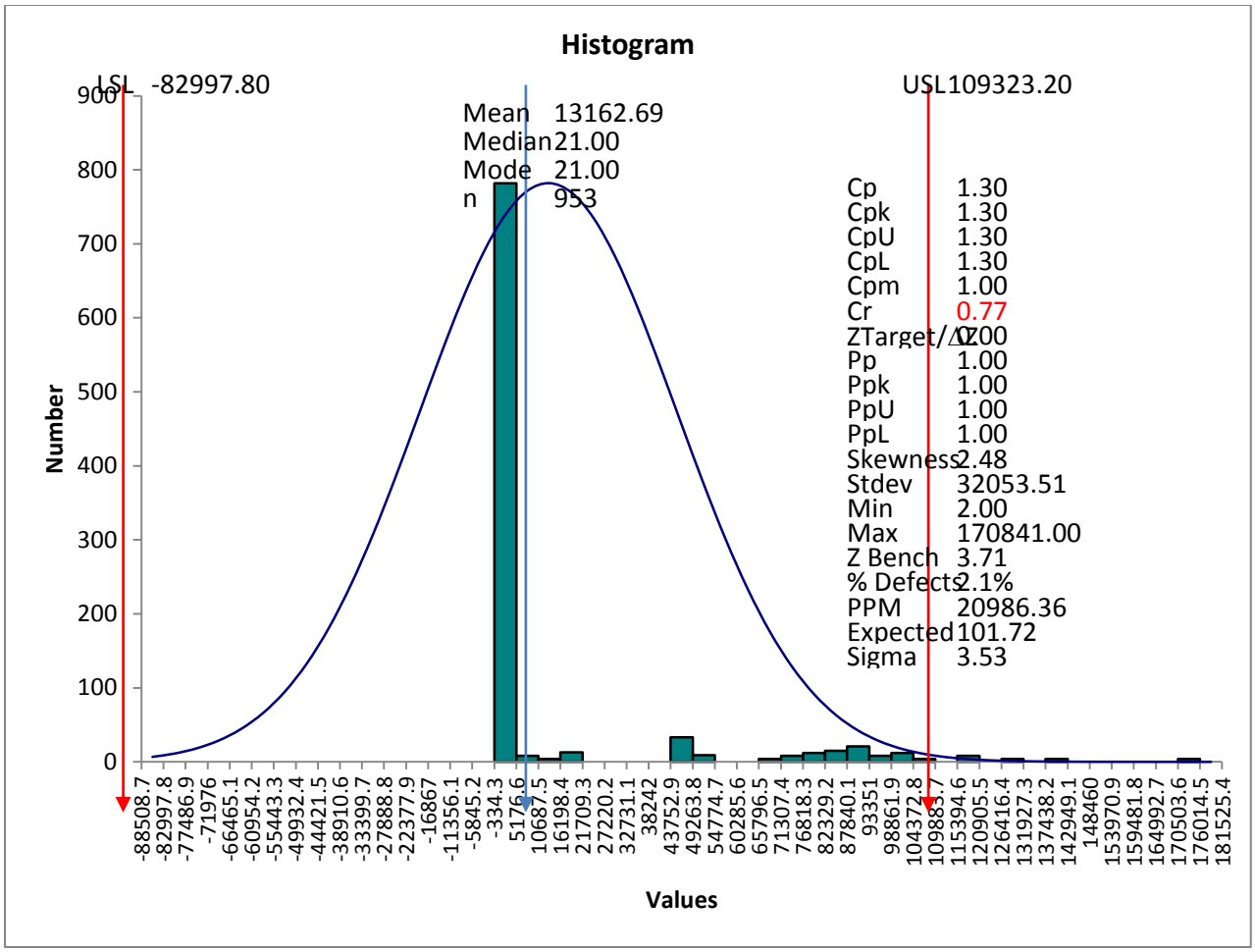
4.3.4.2 Procedure

The meeting was run for three minutes with three users aggressively participating. This meant that the desktop feature was left on with a video playing on the side of the screen. All users had the screen viewing on and continued to carry out other activities. The first test did vigorous floor control , the second vigorous handraising and the third did vigorous polling.

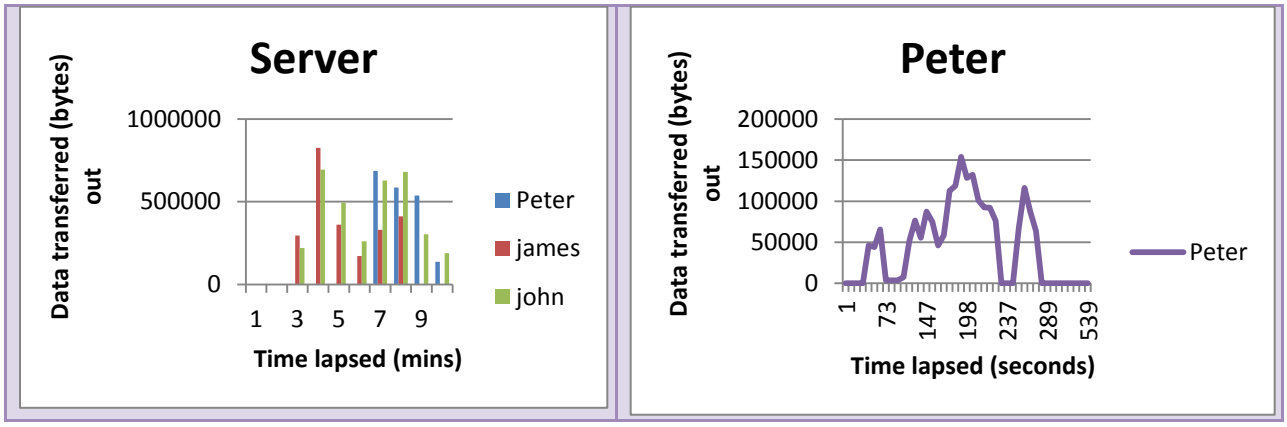
4.3.4.3 Results

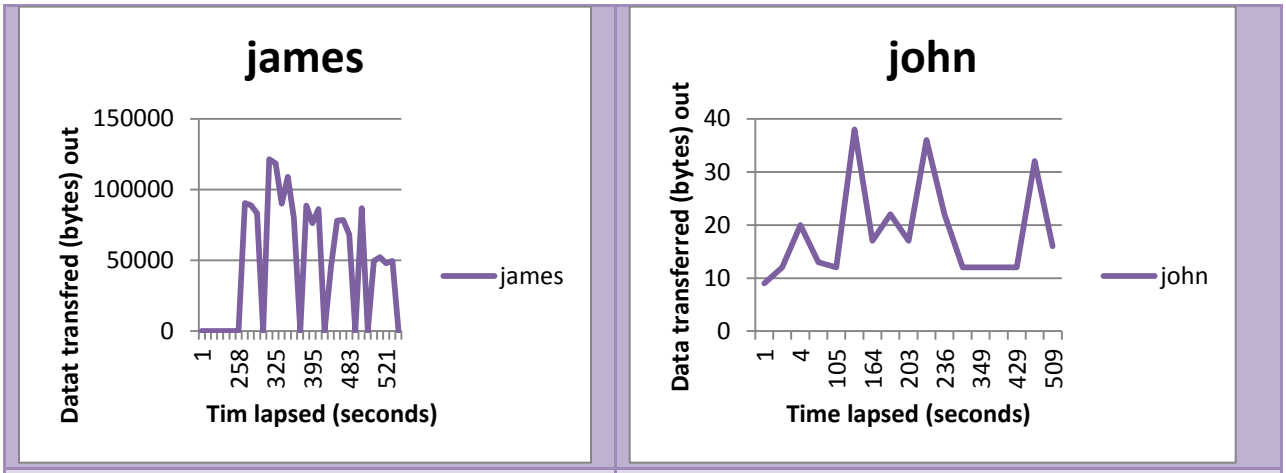
The following graphs were obtained.





Graphs of Bytes against Time (seconds)





Total server Side Bytes sent in 5 minutes	
Peter	1945669
James	2397262
John	3467153
Total server Bytes (Outgoing)	7810084

Individual client Side Bytes sent in 5 minutes		
Peter	Total Bytes (Outgoing)	1965645
James	Total Bytes (Outgoing)	1589578
John	Total Bytes (Outgoing)	314

Figure 30: Stress test for extremely active meeting

4.3.4.4 Discussion

The graphs obtained above potray that users in this meeting were very active. From the client Side Bandwidths, one can immediately see that Peter and James had excessive bandwidth usage. This could mean that they were presenters at one point in the meeting at which they participated in sharing their screens while others viewed. This can also be seen at the server side as more bandwidth was used to send data to John who never presented and therefore never shared their screen. John therefore was sent approximately (1589578 + 1965645) bytes of data. As previously explained the peaks in the graphs correspond to sending entire image packages instead of sending only the differences. James received only Peter's packages and Peter received only James packages as shown in the reduction of packages on the server side sent to them. The bandwidth consumed by each user is shown in the client Side Bytes table above. The following equations calculate the bandwidth needed per second by the two users that did desktop sharing.

$$\left(\frac{1965645}{5x60}\right) = 6552.15\ bytes\ per\ second\ or\ 20KB\ every\ 3\ seconds\ consumed\ by\ Peter$$

$$\left(\frac{1589578}{5x60}\right) = 5298.6\ bytes\ per\ second\ or\ 16\ KB\ every\ 3\ seconds\ consumed\ by\ James$$

The total bandwidth required on the server side is calculated below:

$$\left(\frac{7810084}{5x60}\right) = 26033.61\ bytes\ per\ second\ (26KB)$$

Therefore in an extremely active meeting with three users, the server needs approximately 26KB. This is an extra 7KB more than that needed for a normal meeting.

4.3.5 Experiment 5: Bandwidth required without image differencing

4.3.5.1 Purpose

The purpose of this experiment was to determine the amount of bandwidth that the system would have consumed had there been no computation of the image differences. In this case a screen image was taken and immediately sent to all clients without any additional processing or image manipulations.

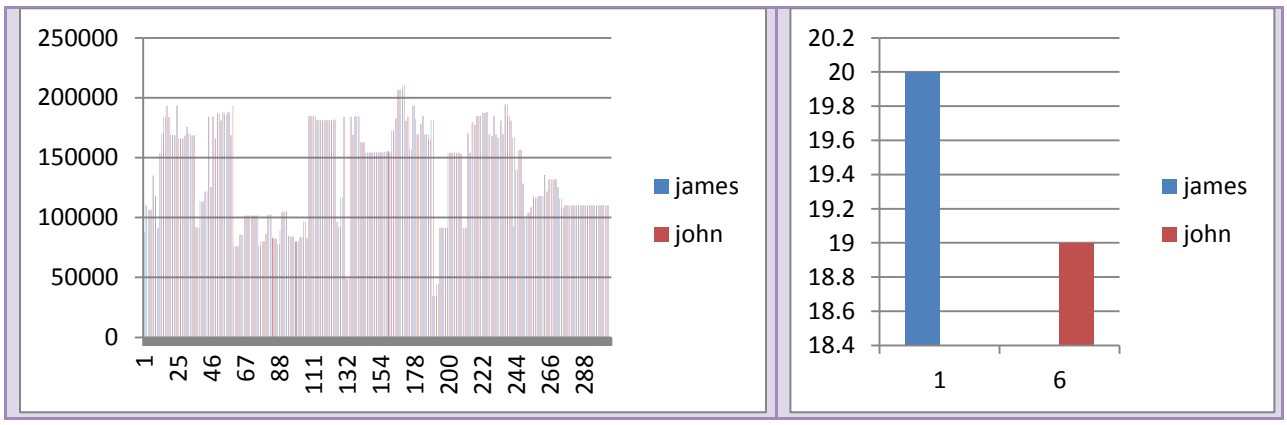
4.3.5.2 Procedure

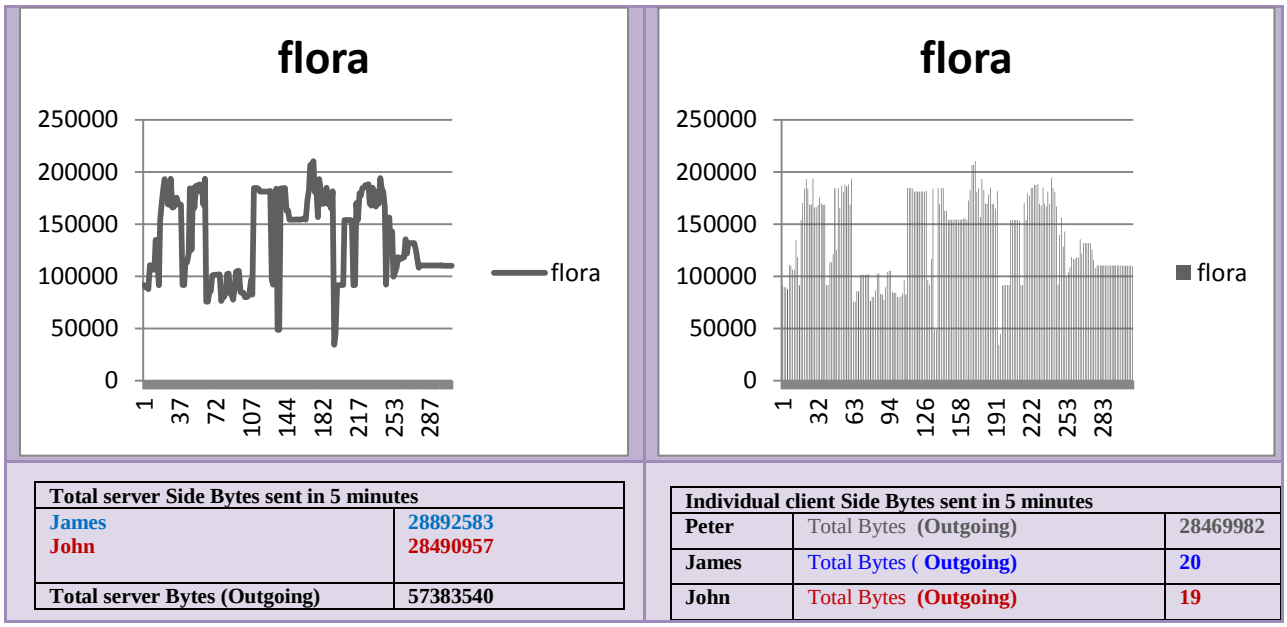
A previous version of the same software with all the functionality before image difference was implemented into the system was used. The meeting was run for five minutes with three users participating. The focus was on the desktop sharing feature, so only the desktop application was used. Users viewed the desktop of the presenter and a log file of all the desktop transactions was generated. This last test used previous data that had been tested once on the screen sharing with and without image difference. Due to time, multiple tests were not conducted. The following is a comparison obtained from the previous results.

4.3.5.3 Results

The following graphs of the number of bytes transferred against the time transferred were plotted. The first graph corresponds to the server, the second three are clients. The bandwidth required by each client is shown in the Individual client side Bandwidth table and the servers total bandwidth is shown in the server Side Bandwidth's table.

Graphs of Bytes against Time (seconds)





Total server Side Bytes sent in 5 minutes	
James	28892583
John	28490957
Total server Bytes (Outgoing)	57383540

Individual client Side Bytes sent in 5 minutes		
Peter	Total Bytes (Outgoing)	28469982
James	Total Bytes (Outgoing)	20
John	Total Bytes (Outgoing)	19

Figure 31: Bandwidth consumed by screen sharing without the use of image differencing

4.3.5.4 Discussion

At first glimpse of the graphs shows a tremendous increase in activity on the server side. The server is sending packages received from Peter to James and John. The graph for Peter shows that most of the packages sent are over 100KB as compared to Experiment1 with the computation of differences where most of the packages were below 70KB. As before negligible bandwidth is required by James and John to receive screen images. Most of the bandwidth is seen on Peter who sends packages to the server and the server which in turn sends these packages to both James and John.

The following are calculations for Peter's bandwidth requirements.

$$\left(\frac{28469982}{5x60}\right) = 94899.94\ bytes\ per\ second$$

As previously mentioned, it takes approximately 110.5556 milliseconds to capture an image, therefore there are no processing delays to consider and an image is transmitted every second. Therefore in three seconds, Peter consumes a total of:

$$94899.94x3 = 284699.82\ bytes\ per\ 3\ second$$

This is about 285KB every three seconds as compared to 25KB every three seconds obtained with using image difference. If the image difference was faster and managed to transfer an image every second for the sake of the user's experience and system responsiveness, it would result in 75KB (25KBx3) of data sent every three seconds which is still far lesser than the 285KB. This means that by implementing a differencing strategy into the system, the total bytes transferred by Peter was reduced to 26.3% provided the system was able to send an image every second.

$$\left(\frac{75}{285}\right)x10 = 26.3\%$$

The following calculations are to determine the server's bandwidth requirements.

$$\left(\frac{57383540}{5x60}\right) = 191278.47\ bytes\ per\ second$$

Again this is multiplied by 3 to see how much data is distributed in 3 seconds.

$$191278.47x3 = 573835.4\ bytes\ per\ 3\ seconds$$

The server therefore needs to transmit 573KB every three seconds. In the desktop sharing experiment with image differencing, 48KB of data were sent every three seconds. Assuming that the image processing occurred much faster and images were sent every second then approximately 144KB (48KBx3) of data was sent every second.

$$\left(\frac{144}{573}\right)x100 = 25.13\%$$

This means that implementing the image differencing approach to the system reduced the total amount of data transferred at the server side to 25.13% provided that the system was able to send an image every second.

4.4 Summary

The responsiveness of the system and the users experience with the system has been evaluated. The system was found to be responsive enough and to provide an acceptable user experience at a refresh rate of one image per second for the desktop sharing application and one data update per second for the rest of the application. However, concerns to improve the desktop refresh rate were raised. It was then found that the system might have a potential bottleneck in computing differences for images and as a result cause the delay in updates.

The desktop sharing application was found to be the most bandwidth consuming; however, implementing the image differencing approach reduced bandwidth usage by approximately 75%. Despite the slight delay in displaying the current image, this approach is recommended as it drastically reduces the amount of bandwidth transmitted in the system. Therefore, in response to the two research question "*is it possible to build a desktop sharing application that works with low bandwidth conditions?"* the answer is yes and "*is it possible to manage meeting procedures effectively despite varying internet conditions?*" the answer is more reserah and testing is needed before sound conclusions can be made..

Chapter 5

Future Work and Conclusion

5.1 Future Work

The system would greatly benefit from more improvement. The hand raising menu could also be put on the outside an made visible and easily accessible by users. To increase user friendliness more event notifications could be added. For the screen sharing feature, an option to either share the entire desktop or portions of the desktop could be added. In addition, users could choose who they permit to view their screens. It would also be interesting to add buffering onto the screen sharing such that users could rewind an playback the screen to a certain extent. Additional bandwidth control based on latency or the speed of delivery of a package or update could be implemented. In this case, when the latency is large, the rate of sending or receiving screen packages is reduced thus dropping the number and size of data packages transferred. In order to produce clear desktop images, the desktop window should be a separate component from the main user's window, so that this window can be enlarged or zoomed to see mage features clearly.

Two approaches could also be used to tackle the bottleneck problem described in the evaluation stage. One could use pipelining to produce parallelism when calculating differences. In other words, while one thread is executing the image differences, another thread also starts executing differences for the next iteration. Another method could be that of sending a Key frame image every predefined interval in order to avoid the need of reconstruction the image at the server side and consuming more time. Lastly one could try use a faster differencing algorithm or try coding the project in C++ as java is perceived to be significantly slower and more memory consuming than C or C++.
Increase number of users when testing n see performance.

A help database could be added to the handraising menu and an option to search for a specific meeting room could be added. As suggested by a user, more details of the users in the meeting such as contact numbers, work locations etc could be added. The host could be given the right to choose whether to let uninvited users into their meeting or not, as currently anyone can enter any meeting room. An option to choose whether to answer a poll or not could be added, and the number of polls already conducted in a meeting could be added to the statuses section under the participants list. Polls option results could be calculated as percentages obtained from the option votes over the total number of people that have currently responded and voted to that poll. More expressions of emotions could be added on the hand raising menus (ie speak faster, slower, softer etc). It would also be great to be able to actually simulate a low bandwidth environment and test the application on it to confirm the results of the evaluations.

5.2 Conclusion

The project was aimed to produce a system that would work well under low bandwidth conditions. Researched showed that various methods had been used to tackle this problem. For this report, it was shown in the experiments that screen sharing consumes more bandwidth than handraisng and

therefore it is advisable to put higher priority on the handraisng mechanism. The system was designed using an iterative approach. Due to time constraints not all the evaluations could be completed properly and things were rather done in a rush. However research and development should be done in this system to improve the drawbacks experienced.

Appendix A

University of Cape Town
Department of Computer Science

Participant Consent Form

Researcher's name: _____

The research conducted in this experiment is for educational purposes only. Participation is voluntary and one can choose to withdraw from the experiment at any time or refuse to answer any questions which they feel uncomfortable. Data will be collected anonymously such that individual names are not associated with responses. You will not be paid or compensated for your participation. There are no risks involved in this study.

You may stop at any time before, during or after the experiment to ask the researcher questions or raise your concerns. You can also contact me via email at fkstacey@gmail.com if any other concerns arise.

By signing this consent form, you indicate that you have read the above information, asked questions, understood the procedure and agree to participate in the study.

Participant's Name: _____

Participant's Signature: _____

Date of Consent: _____

Thank you for your participation…

Appendix B

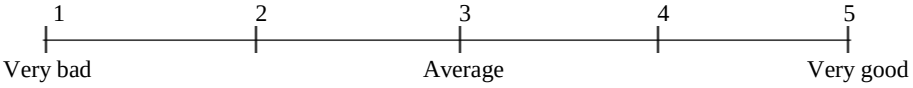
Experiment Questionnaire

Age: _____
Gender: _____
Degree: _____
Year of Study: _____

Please circle or tick your responses where appropriate

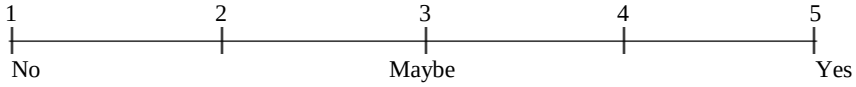
Have you ever used a videoconferencing system before? ☐ No ☐ Yes

If yes Please rate your experience with the technology:

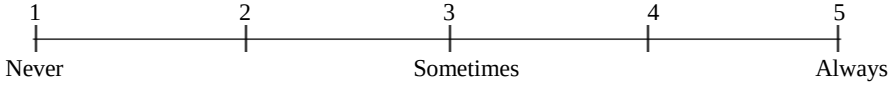


How many people were already in the meeting when you joined the meeting?
1st Round_____
2nd Round_____

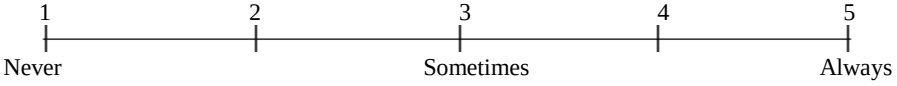
Was the system responsive enough?



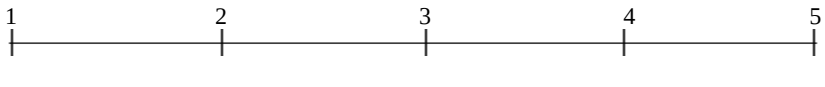
Did you feel that you were part of the meeting?



Could you participate as easily as others?



Was the hand raising mechanism efficient for conveying your emotions?



Not at all efficient Not sure Very efficient

If you were a host in at any point in the meeting:
Did you feel you had enough control over the meeting procedures?

☐ No ☐ Maybe ☐ Yes

If you were a Presenter at any point in the meeting:

Did you feel you had adequate control in the meetings?

☐ No ☐ Maybe ☐ Yes

Was it possible to keep track of what was going on in the meeting (i.e. meeting procedures)?

1 2 3 4 5

Not at all efficient Not sure Very efficient

Please rate the polling feature.

1 2 3 4 5

Very bad Average Very good

Please rate the screen sharing feature.

1 2 3 4 5

Very bad Average Very good

What would you prefer during screen sharing?

a. To receive fast, clear, good qualities images when the bandwidth is good and then have no image displayed when the bandwidth reduces.
b. To receive a slower but constant rate of clear, good quality images regardless of bandwidth fluctuations.

Please rate the sense of presence in meetings.

1 2 3 4 5

Very strong Average Very weak

Please rate your overall experience with the system:

1 2 3 4 5

Very bad Average Very good (satisfactory)

Would you use this system if it were to be deployed?

No ☐ ☐ Maybe ☐ Yes

How did the increase in the number of people affect the responsiveness of the system?

_____

How did the decrease in the number of people affect the responsiveness of the system?

_____

How does this system compare to the previous system you mentioned at the beginning?

_____

How can this system be further improved?

Feedback and comments:

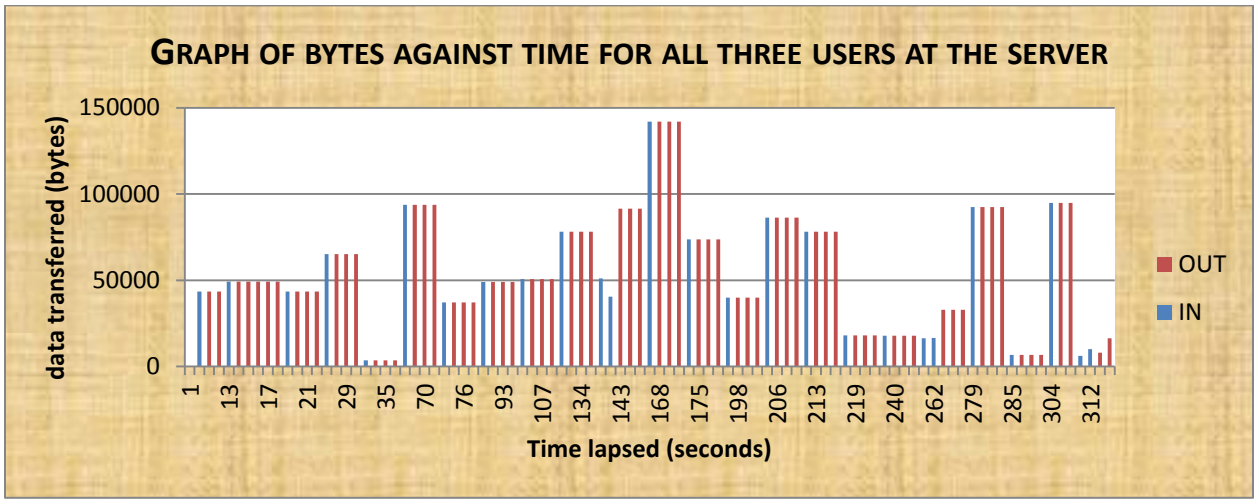
Thank you for your participation…

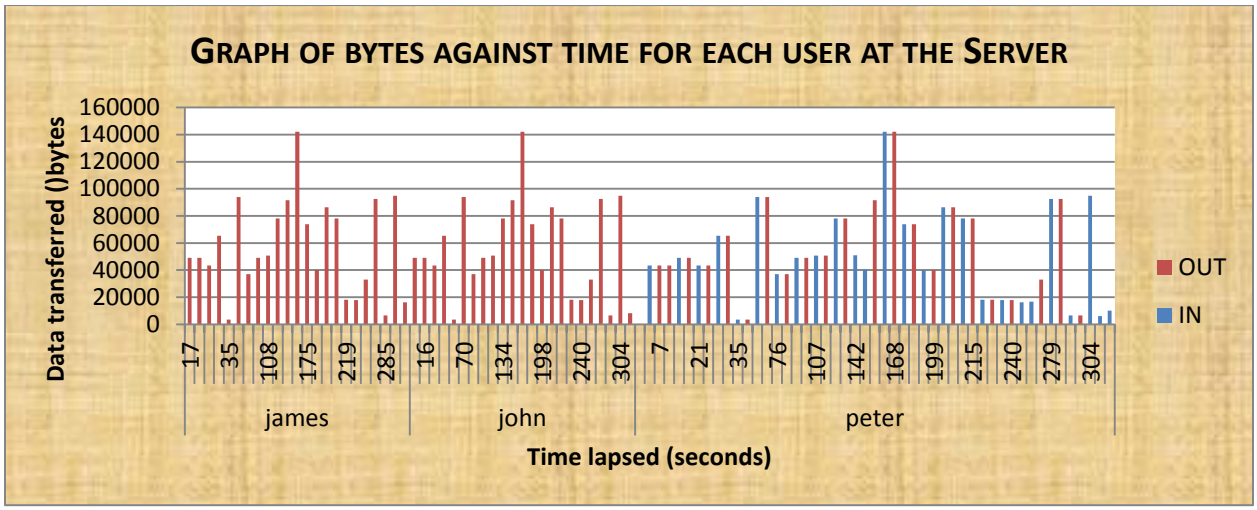
References

1. Austin, T., Drakos, N., & Mann, J. (2006). Web conferencing amplifies dysfunctional meeting practices. *Gartner Research Report*, (March). Retrieved from http://data.vitusbering.dk/vbi/isi/Gartner-web_conferencing_amplifies_d_138101.pdf
2. Causey, R., Birnholtz, J., & Baecker, R. (2006). Increasing Awareness of Remote Audiences in Webcasts. *Proc. CSCW2006* (pp. 59–60). Citeseer. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.7036&rep=rep1&type=pdf
3. Chan, A. (2004). *Designing Haptic Icons to Support an Urgency-Based Turn-Taking Protocol by Master of Science The University of British Columbia*. *October*.
4. Chen, M. (2002). Achieving effective floor control with a low-bandwidth gesture-sensitive videoconferencing system. *Proceedings of the tenth ACM international conference on Multimedia - MULTIMEDIA '02*, 476. New York, New York, USA: ACM Press. doi:10.1145/641108.641109
5. Cohen, M., & Wang, J. (2009, September 1). System and method for very low frame rate video streaming for face-to-face video-conferencing. *US Patent 7,583,287*. Google Patents. Retrieved from http://www.google.com/patents?hl=en&lr=&vid=USPAT7583287&id=bmTKAAAAEBAJ&oi=fnd&dq=system+and+method+for+very+low+frame+rate+video+streaming+for+face-to-face+video+conferencing&printsec=abstract
6. Daly-Jones O, Andew Monk, L. W. (1998). Some advantages of video-conferencing over high-quality audio conferencing: fluency and awareness of attentional focus. *International Journal of Human-Computer Studies*, *49*(1), 21-58. doi:10.1006/ijhc.1998.0195
7. Emanuel, H., Niederman, F., & Shapiro, S. (1995). Online services as distributed meeting support software. *Proceedings of the 1995 ACM SIGCPR conference on Supporting teams, groups, and learning inside and outside the IS function reinventing IS* (pp. 213–222). ACM. Retrieved from http://portal.acm.org/citation.cfm?id=212601
8. Garcia-Luna-Aceves, J. J., Mantey, P. E., & Potireddy, S. N. (2005). Floor Control Alternatives for Distributed Videoconferencing over IP Networks. *2005 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, 1-10. Ieee. doi:10.1109/COLCOM.2005.1651241
9. Goode, B. (2002). Voice over internet protocol (VoIP). *Proceedings of the IEEE*, *90*(9), 1495–1517. IEEE. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1041060
10. Greenberg, S. (1989). Computer support for real time collaborative work. *Proceedings of the Conference on Numerical*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.9112&rep=rep1&type=pdf
11. Hansen, J. (2006). Screen sharing. *US Patent 7,046,134*. Retrieved from http://www.google.com/patents?hl=en&lr=&vid=USPAT7046134&id=3-R3AAAAEBAJ&oi=fnd&dq=screen+sharing&printsec=abstract
12. Isaacs, E. A., & Tang, J. C. (1994). What video can and cannot do for collaboration: a case study. *Multimedia Systems*, *2*(2), 63–73. Springer. Retrieved from http://www.springerlink.com/index/k6u5j47555154187.pdf
13. Jumisko-Pyykkö, S. (2006). "I would like to see the subtitles and the face or at least hear the voice": Effects of picture ratio and audio–video bitrate ratio on perception of quality in mobile television. *Multimedia Tools and Applications*, *36*(1-2), 167-184. doi:10.1007/s11042-006-0080-9
14. Koskelainen, P., Ott, J., & Schulzrinne, H. (2006). *X. Wu," Requirements for Floor Control Protocols*. Retrieved from

http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Requirements+for+Floor+Control+Protocols#0
15. Koskelainen, Petri, Schulzrinne, H., & Wu, X. (2002). A SIP-based conference control framework. *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video, May* (pp. 12–14). Citeseer. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.98.1857&rep=rep1&type=pdf
16. Lieb, A., & Benton, J. (2010). METHOD AND SYSTEM FOR BROWSER-BASED SCREEN SHARING. *US Patent App. 12/953,054*. Retrieved from http://www.google.com/patents?hl=en&lr=&vid=USPATAPP12953054&id=bn_mAQAAEBAJ&oi=fnd&dq=method+and+system+for+browser+based-screen+sharing&printsec=abstract
17. Lu, Y., Zhao, Y., Kuipers, F., & Van Mieghem, P. (2010). Measurement study of multi-party video-conferencing. *NETWORKING 2010* (pp. 96–108). Springer. Retrieved from http://www.springerlink.com/index/8X27W207513N2312.pdf
18. Malpani, R. (1997). Floor control for large-scale MBone seminars. *Proceedings of the fifth ACM international*. Retrieved from http://portal.acm.org/citation.cfm?id=266356
19. Monk, A. F., & Watts, L. (1995). A poor quality video link affects speech but not gaze. *Conference companion on Human factors in computing systems* (pp. 274–275). ACM. Retrieved from http://portal.acm.org/citation.cfm?id=223671
20. Post, W. M., Huis in 't Veld, M. a a, & Boogaard, S. a a. (2007). Evaluating meeting support tools. *Personal and Ubiquitous Computing*, *12*(3), 223-235. doi:10.1007/s00779-007-0148-1
21. Reidsma, D., Akker, R., Rienks, R., Poppe, R., Nijholt, A., Heylen, D., & Zwiers, J. (2007). Virtual meeting rooms: from observation to simulation. *Ai & Society*, *22*(2), 133-144. doi:10.1007/s00146-007-0129-y
22. Scholl, J., Parnes, P., McCarthy, J. D., & Sasse, A. (2005). Designing a large-scale video chat application. *Proceedings of the 13th annual ACM international conference on Multimedia - MULTIMEDIA '05* (p. 71). New York, New York, USA: ACM Press. doi:10.1145/1101149.1101160
23. Takao, S. (1999). The effects of narrow-band width multipoint videoconferencing on group decision making and turn distribution. *ACM SIGSOFT Software Engineering Notes*, *24*(2), 109-116. doi:10.1145/295666.295678
24. Introduction to url encoding.(n.d). Retrieved October 24, 2011, from http://www.permadi.com/ tutorial/urlEncoding/
25. HTTP Protocol. (2008). Retrieved October 24, 2011, from http://www.roseindia.net/jsp/jspsession/ HttpProtocol.shtml

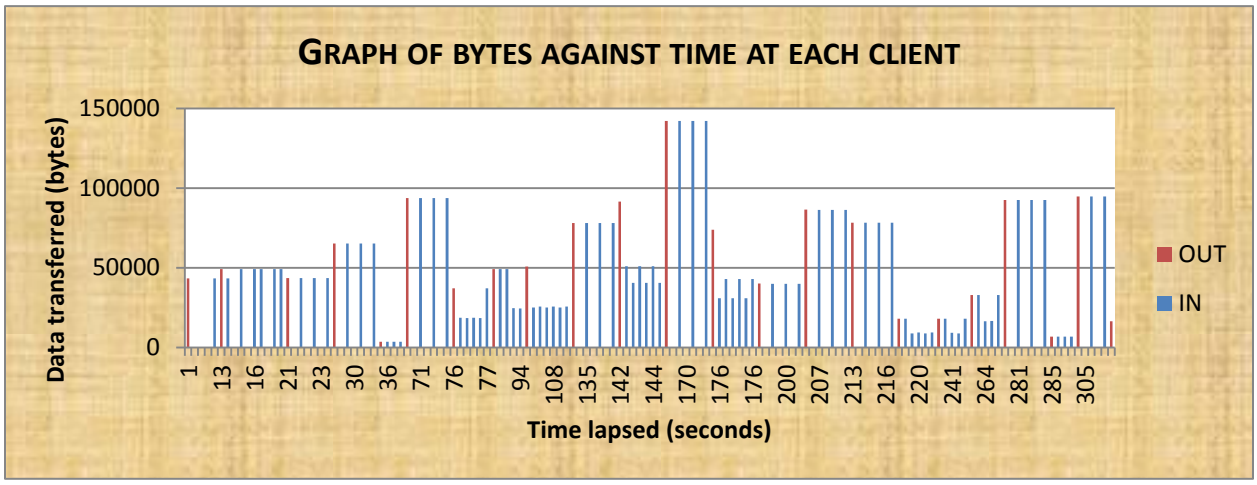
Desktop Only server Side

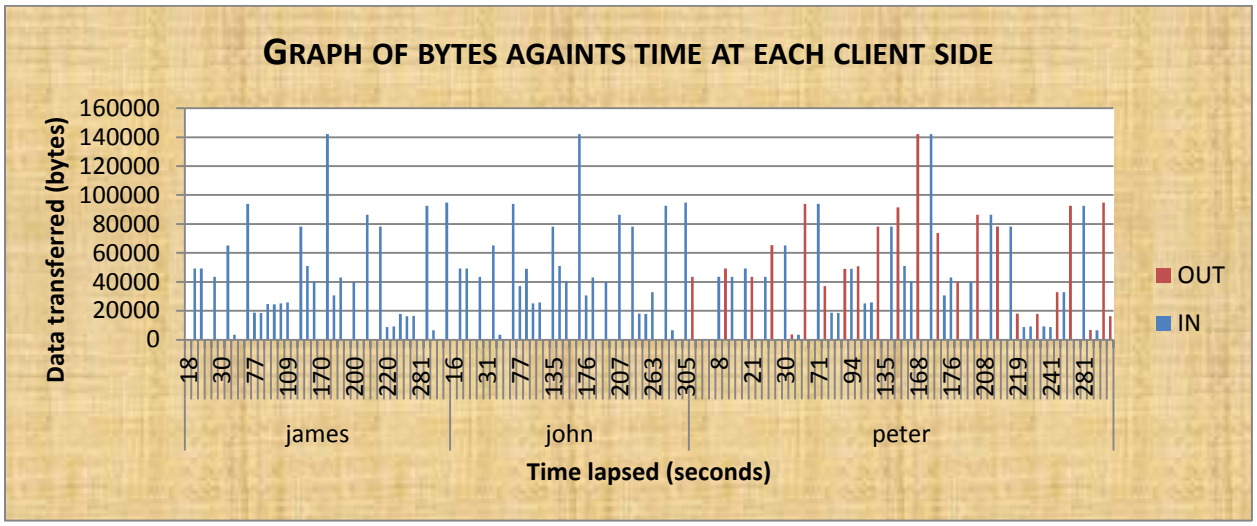


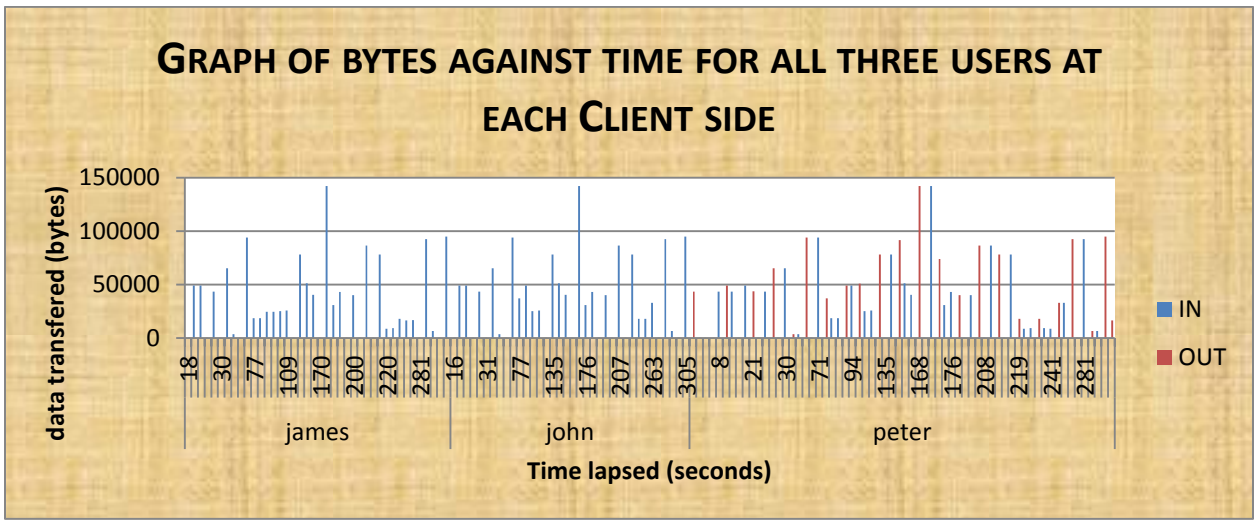


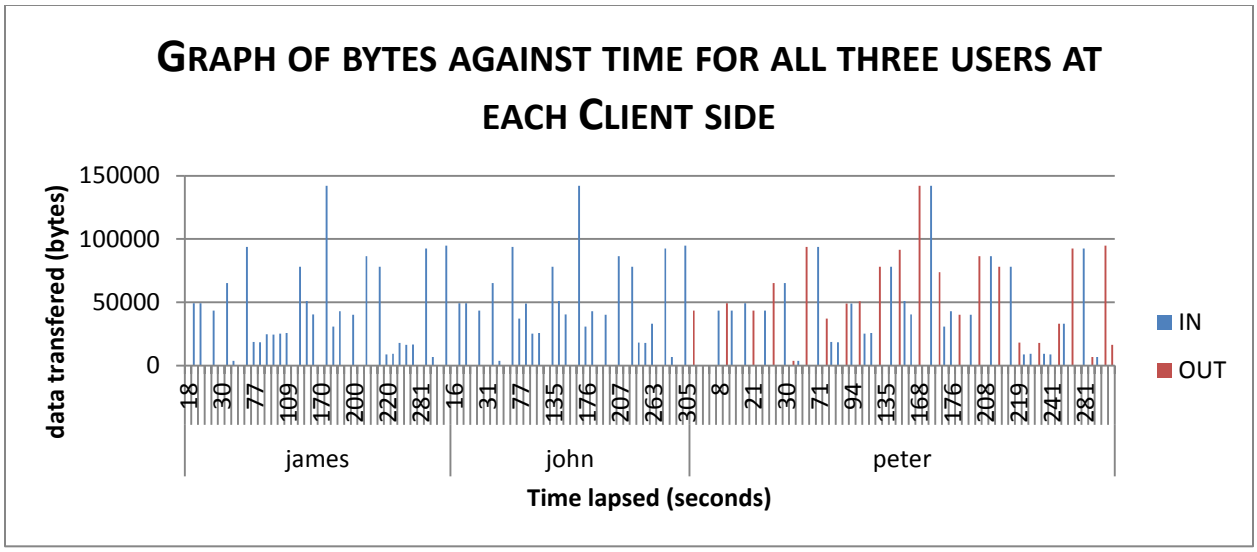
Row Labels	IN	OUT	Grand Total
james		**1310843**	**1310843**
john		**1302666**	**1302666**
peter	**1305041**	**1237361**	**2542402**
Grand Total	**1305041**	**3850870**	**5155911**

Desktop only clients sides


GRAPH OF BYTES AGAINST TIME AT EACH CLIENT
Data transferred (bytes)
150000
100000
50000
0
1 13 16 21 23 30 36 71 76 77 94 108 135 142 144 170 176 176 200 207 213 216 220 241 264 281 285 305
Time lapsed (seconds)
OUT
IN


GRAPH OF BYTES AGAINTS TIME AT EACH CLIENT SIDE
Data transferred (bytes)
160000
140000
120000
100000
80000
60000
40000
20000
0
18 30 77 109 170 200 220 281 16 31 77 135 176 207 263 305 8 21 30 71 94 135 168 176 208 219 241 281
james
john
peter
Time lapsed (seconds)
OUT
IN


GRAPH OF BYTES AGAINST TIME FOR ALL THREE USERS AT EACH CLIENT SIDE
data transfered (bytes)
150000
100000
50000
0
18 30 77 109 170 200 220 281 16 31 77 135 176 207 263 305 8 21 30 71 94 135 168 176 208 219 241 281
james
john
peter
Time lapsed (seconds)
IN
OUT


GRAPH OF BYTES AGAINST TIME FOR ALL THREE USERS AT EACH CLIENT SIDE
data transfered (bytes)
150000
100000
50000
0
18
30
77
109
170
200
220
281
16
31
77
135
176
207
263
305
8
21
30
71
94
135
168
176
208
219
241
281
james
john
peter
Time lapsed (seconds)
IN
OUT