UiT
THE ARCTIC
UNIVERSITY
OF NORWAY

Faculty or Science and Technology

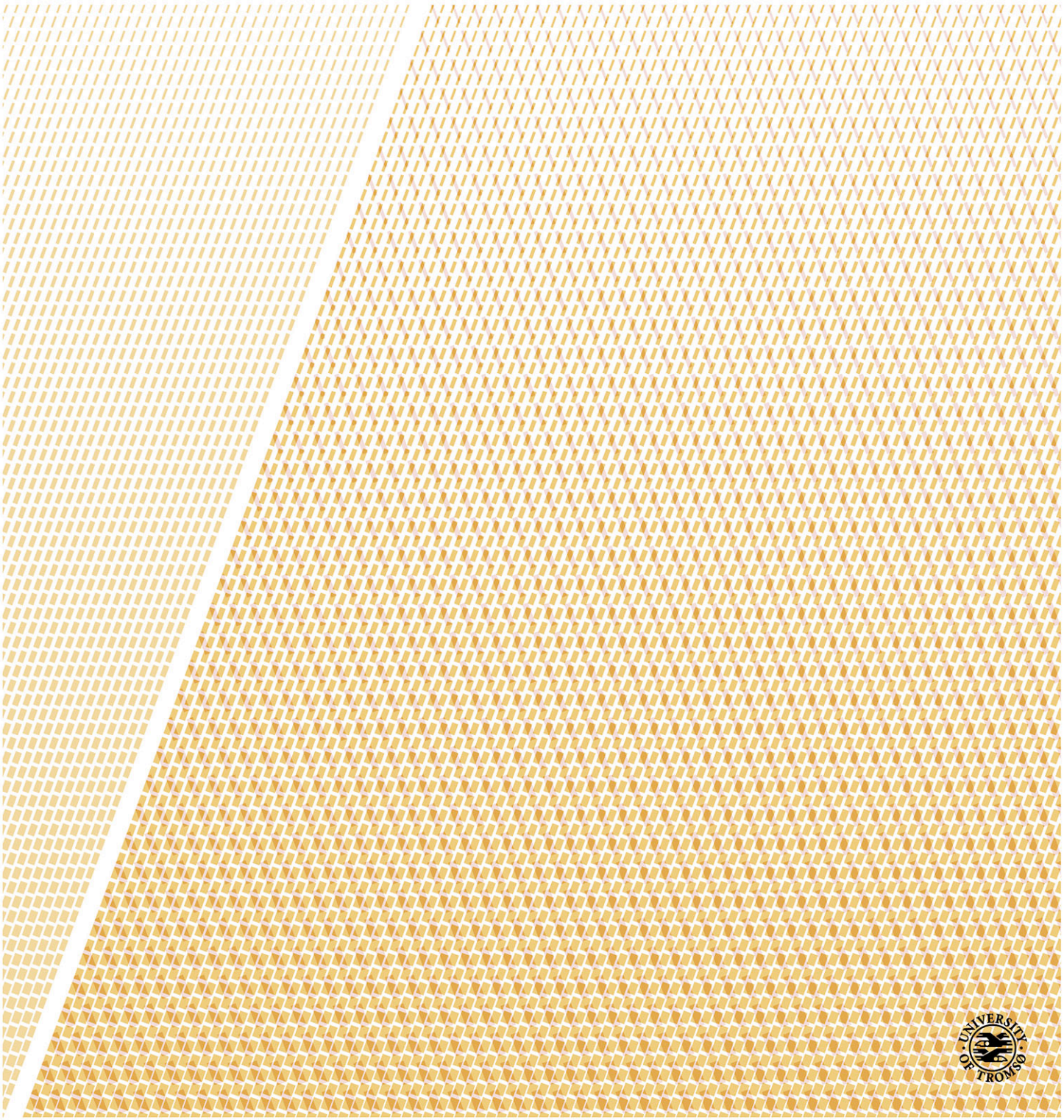
Department of Computer Science

A distributed remote presence system masking the effects of delays in human-to-human remote interaction

—

Fei Su

Ph.D. dissertation in Computer Science



Abstract

In a computer supported distributed stage performance, such as on theatre stages, actors are located in different locations. Actors in one location want to perform the performance with actors in another location. This can be achieved by merging actors into a virtual stage and audience will see actors on local stage interact with remote actors on the virtual stage. Thus, actors in different locations can act together. Meanwhile, actors in different locations should interact as if they are at the same place.

There are two major challenges to develop the above system. 1. **Amplified Interaction**: There is a significant distance between actors on and audience at a theater stage. Actor on a remote stage may not be able to understand what local actors are doing. Exaggerated movements and makeup are used to make an audience better understand what actors are doing. We term this Amplified Interaction. 2. **Masking the Effects of Delays**: The state about actor in one location will be delivered to remote locations. The remote locations may receive the state being delayed too much due to high network transmission latency or high process latency. The consequence is interactions between actors in different locations will be out of sequence. The delay can always be reduced, but can never be totally removed. The system needs to mask the effects of delays to make audience believe actors are interacting together.

The required functionalities for distributed stage performance include: 1. Create actor's representation on each stage, we term this **remote presence**. The remote presence can be a skeleton or robot that simulate the behavior of the actor. It can also be a visualization of an actor. 2. Detect and analyze actor's actions. The detected state and gestures about actors will be used to create the remote presence of actor and amplify actor's interaction. 3. Apply different techniques to mask the effects of delays. 4. Manage the handover of state to different stages.

Existing approaches include teleconferencing, network games and virtual environments. Commercial teleconferencing systems allow two or several persons to interact through instant text, video and audio as well as file transfer. Virtual immersive systems merge a human environment into a virtual one, allowing people to interact in the virtual environment. However, most of these systems do not consider the effects of delays

(SKYPE, Apple iMessage and etc). They normally let people interact and live with latency. The interaction may become awkward because of delays and limited remote representations of each other. Network games have techniques to mask the effects of delays, but those techniques have different drawbacks and do not totally fulfill the required functionalities described above. Network games typically use virtual entities, such as cars, robots, and monsters. They do not provide a remote presence showing the real actor.

This dissertation presents MultiStage, a human-to-human interaction system meant to be use by actors on a stage to interact and perform with actors on other stages as if they were on the same stage. MultiStage includes several local side stages and a global side. It uses a publish-subscribe model to handle the handover of data streams. Local side produces data streams about actors to global side. Local sides also subscribe to data streams from global side to create remote presence about actors. Global side receives data streams and sends data streams back to the local side according to subscriptions. When actors interact with remote actors, the system amplifies actors' actions by adding text and animations to the remote presences. When the remote presences lag behind too much because of network and processing delays, the system applies various techniques to mask the effects of delays, including switching rapidly to a prerecorded video or animations of individual actors.

The experiments show that the MultiStage system currently scales to at least three stages with a total of at least 12 outgoing and 36 incoming data streams across the Internet, and comprises in total 15 computers, 12 cameras, and several projectors. The resource usage in all cases is either very low or low. The implication is that the system is not resource limited. Several user studies are done to evaluate the effects of delays and different techniques to mask the effects of delays. The experiments show that we need to apply the techniques to mask the effects of delays for tight interactions (dancing or rapid hand movement, such as handshake).

Another use of the MultiStage video distribution model is called pVD. The Personal Video Distribution (pVD) system supports sending and viewing live and stored videos between any of a single user's computers, and allows for a smooth handover of play back between computers. The system avoids any third parties, and relies only on the user's personal computers. The architecture is comprised of functionality for sending videos, subscribing to videos, and maintaining the video playback state. The design has a local side sending and viewing videos, and a global side coordinating the switching and distribution of videos, and maintaining subscriptions and video state. A set of experiments was conducted to document the performance of the prototype. The results show that pVD global side has low CPU usage, and supports a handful of simultaneous exchanges of videos on a wireless network.

Acknowledgements

I wish to thank various people for their help and contribution on this dissertation.

First and foremost, I would like to express my very great appreciation to Professor Otto J.Anshus. He has provided me with valuable and patient guidance in my whole PhD period. He always leads me to the right direction. Without his help, I could not complete my dissertation.

I would like to thank to all my co-supervisors and co-authors John Markus Bjørndalen, Daniel Stødle and Phuong Hoai Ha. They provided lots of constructive guidance during my PhD work. They also gave patient guidance during experiment period and helped me finish writing papers.

I would like to thank to my colleagues Giacomo Tartari, Bård Fjukstad, Lars Tiede, Edvard Pedersen, Ibrahim Umar, and graduated Dr. Tor-Magne Stien Hagen and Yong Liu for their technical help in my PhD work.

I would like to thank to all the technical staffs in the department of computer science. Thank for Jon Ivar Kristiansen, Ken-Arne Jensen for help me build the experimental equipments. Thank for Kai-Even Nilssen for his technical support. Thank for Maria Wulff Hauglann and Lars Ailo Bongo for lending me experimental tools countless times. I would also like to thank for the administrative staffs, especially Svein Tore Jensen and Jan Fuglesteg, for their kind help.

I would like to thank to Professor Weihai Yu and Researcher Chun Li for their encouragement and valuable suggestions during my PhD study.

Finally, I wish to thank to my parents and all my friends for their support and encouragement throughout my PhD program.

Contents

List of Figures

List of Tables

Abbreviations

2D	Two-Dimensional. 12, 33, 41, 42, 53, 55, 56, 127
3D	Three-Dimensional. 6, 8, 12, 21, 30, 33–35, 40–42, 46, 49, 53, 55, 56, 81, 127
CDN	Content Distribution Network or Content Delivery Network. 110
CEP	Complex Event Processing. 63
DCEP	Distributed Complex Event Processing. 63
DIP	Distributed Immersive Performance. 32, 35, 60, 61, 100
DLNA	Digital Living Network Alliance. 111
DNS	Domain Name System. 61
DR	Dead-Reckoning. 4, 61, 62, 102
DSDS	Distribution of State Data Streams. 13, 26–28, 36, 37, 42, 44–54, 56, 65, 66, 71–78, 80, 81, 83, 85, 87, 88, 94, 127, 129, 133–135, 139
DVE	Distributed Virtual Environment. 4
FPS	Frames per Second. 9, 12, 36, 78, 85, 87
GSA	Global State Analysis. 22, 26, 27, 42–46, 50, 52, 74, 77, 81, 132
GSM	Global State Monitoring. 22, 26, 27, 37, 40, 42–46, 49, 50, 74, 77, 81, 132
HD	High-Definition. 118, 120, 122, 125
IoT	Internet of Things. 63
LAN	Local Area Network. 9, 21, 31, 68, 71, 81, 83, 87, 102, 104
LCD	Liquid Crystal Display. 32
LL	Local-Lag. 4, 61, 62, 102
LSA	Local State Analysis. 22, 24, 40–42, 45, 49, 52, 55, 56, 68, 74, 77, 81, 127, 132
LSM	Local State Monitoring. 22, 24, 39–43, 45, 74, 77, 81, 127, 132
NTP	Network Time Protocol. 26, 28, 36, 65, 68, 69, 71–73, 128, 139
P2P	Peer-to-Peer. 3, 35, 110, 111
PTZ	Pan/Tilt/Zoom. 32
pVD	Personal Video Distribution. 7, 9, 14, 16, 18, 107–114, 116–122, 124–126, 129, 133, 134, 138

PVRs Personal Video Recorders. 111

RPG Role-playing game. 100

RTS Real-time strategy. 100

TCP Transmission Control Protocol. 12, 27, 36, 50, 52, 56, 65, 117, 127

UDP User Datagram Protocol. 12, 26–28, 36, 41, 45, 50, 52, 65, 74, 77, 78, 80, 81, 83, 118, 127

UPnP Universal Plug and Play. 111

WAN Wide Area Network. 9, 21, 31, 81, 83, 87, 127

Chapter 1

Introduction

This project has built a system for human-to-human live interaction across distance masking the effects of delays. The purpose is to identify and document the architecture, design, implementation, and performance characteristics of such a system.

Today, remote communication between humans is supported by multiple networks, both wired and wireless. Humans can interact with others through audio, video and touch. Interaction in a virtual space is no longer a novelty. In human interaction, gestures and body movements are used to communicate and help people to better illustrate what they mean.

In particular, on a theatre stage, actors perform various actions together to collaborate. If we locate actors at different continents and let them see each other through data networks and software, the interaction may become awkward because of delays and limited remote representations of each other. Ideally, actors at different locations should interact with each other as if they are at the same physical location. Teleconferencing systems and network games are usage scenarios where humans at different locations interact with each other. However, network games normally use modals to simulate human behaviors. For human-to-human live interaction, actors may feel strange when they interact with modals. A teleconferencing system is normally used by several people to talk together. When more humans are involved, and interaction becomes rapid action-reaction, such as actors dancing together, the above systems may not be able to deliver data about actors fast enough to achieve human-to-human remote interaction because of the network transmission and computer processing delay.

In this dissertation, as part of the Verdione project [1], we present the MultiStage [2] system, for distributed human-to-human acting. Verdione (Virtually Enhanced Real-life synchronizeD Interaction - ON the Edge), aims to research on video processing and network support to merge virtual elements into real world. An actor is represented by a visualization transported to and viewed at remote stages.

Two main functionalities of MultiStage are amplified interaction and masking the effects of delays. The amplified interaction function is mainly done by my colleague Giacomo Tartari [3], I will only give a brief introduction to amplified interaction in this dissertation. This dissertation describes the architecture, design, and implementation of the MultiStage system with focus on masking the effects of delays.

This chapter provides an overview of the dissertation. First, several research questions are addressed. Second, we present a brief overview of the existing approaches related to the research questions. Third, the contribution of the work is summarized. Fourth, the methodology is summarized. Fifth, the limitations are summarized. Sixth, the publications are summarized. Finally, we present the organization of this dissertation.

1.1 Research Questions

Several sub-problems have been attacked to understand and characterize the architecture, design, implementation and performance of a system for human-to-human live interaction across distance.

1. How to do low latency detection of multiple human actors on the same stage. There will be multiple actors on each stage. Each actor will be at different place on the stage. The detection side needs to have a functionality to detect and generate data (include spacial information) about each individual actor. The data must also be generated in low latency to allow human-to-human remote interactions.

2. How to do low latency detection of gestures done by actors on the same stage. Gesture is a pre-defined command to the system to do some functionality. Gesture will be activated when actor do a special action (such as raise hand). To allow consistent human-to-human remote interaction between remote stages, gestures need to be detected accurately and fast enough. The problem will be complicated when gesture are required to be performed by more than one actor. The problem will be even more complicated when those actors are not on the same stage.

3. How to do low latency distribution of state about human actors and gestures between stages. To allow actor interact on different stages, data about actor on one stage need to be delivered to other stages. One individual stage may require all data streams or some of the specific data streams about actors. The system needs a functionality to manage the handover of data streams with low latency. The problem is complicated because the number of actors on each stage and the number of stages can increase. More data streams will be generated, and this will occupy more

Internet traffic. Each stage also needs to know the network location of other stages. When the number of stages increases, this problem will be even more complicated.

4. How to do low latency representation at a stage of remote actors. Actor should interact with the remote presences of other actors in a natural way as if they are at the same place. In a distributed performance, one actor will interact with multiple remote presences, and consider all of the remote presences' positions and actions. The interaction is not just between single user to single user, but will be a spatially dependent many-to-many interaction.

5. How to maintain for the remote actors, the illusion of being on the same physical stage. There is a non-zero delay from when an event happens until it can be observed. It takes time to process a data stream and transfer it to other stages. Sophisticated program will give higher process latency. The longer the distance, the higher the transmission latency is. Higher latency can make the interactions become awkward (For example, it may take longer time for a remote actor react to a handshake action done by a local actor. The reason is because of the delay is too large). Even if the delays can be reduced, they can never be removed.

6. How to provide humans with the state of the system to aid in recovery after failures. To make people interact on different stages, several computers will be distributed at each stage. The distribution of the system among different stages makes it hard to find out failures and recover from them because so many computers are involved.

1.2 Existing Approaches

A Peer-to-Peer (P2P) approach is often used to distribute videos and is able to maintain good video distribution performance when the number of clients increases [4]. P2P systems have a high degree of decentralization, each peer has both client and server functionality. The required resources such as bandwidth and storage are contributed by each node. Little or no configurations are needed once a node is introduced into the system. P2P systems support the use with many clients. It has few critical nodes compare to the large number of nodes. However, P2P systems lack a strong user. This will allow an attacker to add many nodes under his control. To make sure data is available, at least one node with the available data must be online. To keep the data durable, it must be constantly replicated to live nodes. This will consume network bandwidth. In contrast, the MultiStage system distributes data streams among a few locations with not many computers in total. It has a centralized server handling incoming and outgoing data

streams for a few stages. It has an administrator interface for administrator to control the whole system. Each computer on MultiStage system running a monitoring process, the monitored information is made available to the administrator interface. This can help user to determine where failure happens.

Commercial teleconferencing systems (SKYPE, Apple iMessage, and a Remote Camera system presented by [5]), allow two or several persons to interact through instant text, video and audio as well as file transfer. Virtual immersive systems ([6], [7], [8], [9]) merge a human environment into a virtual one, allowing people to interact in the virtual environment. However, most of these systems (SKYPE, Apple iMessage, [5], [7], [8] and [9]) do not consider the effects of delays. They normally let people interact with the latency achieved by the systems.

Network games and several Distributed Virtual Environment (DVE) systems use Dead-Reckoning (DR) [10] and Local-Lag (LL) [11] techniques to mask the effects of delays. For the DR technique, each node simulates a model, say, a moving car. And sends out its updates (include velocity, acceleration etc) to other nodes. The updates can be used to predict the behavior of the specific node. Because the latencies between nodes vary, the update from a node may appear at different times at other nodes. This results in an inconsistent view of the state of the node between nodes. The LL technique artificially delays the update of local node to wait for updates from other nodes, and applies all updates at the same time to keep the same view at each node.

For more detailed descriptions about related approaches, please check the related literature sections in the Chapter 2, Chapter 6 and Chapter 9.

The limitations of the above approaches include: 1. It is designed for two stages not for more than two stages. 2. It captures all actors in one scene, but does not distinguish between actors. 3. Lack of the function to split streams about actors and manipulate streams individually. 4. Lack of function to detect user gestures and amplify user interactions. 5. Application usually depends on a third-party service. User may not be able to access the service, if it is heavily loaded or temporarily down. Users may concern about security (private information stolen by others) and privacy issues (service provider may use personal information) when using third-party services.

In contrast MultiStage improves the situation by: 1. Each actor is represented by a separate data stream. Stages may not need the whole performance. It is more flexible to let each stage subscribe to data streams it wants. 2. Able to manipulate data streams at any stage of communication: begin point, end point, or during distribution. This gives the opportunity for computation of data streams. For example, multiple streams from different stages can be collected during distribution or on the end point. Further computation can be applied to find global gestures done by actors at different stages.

3. Able to recreate the remote presence of an actor from a data stream, and placed on a virtual stage in the same place or somewhere else from where the actor was. 4. Able to reduce or mask the effects of delays especially when actors interact across a very long distance.

1.3 Contributions

The contributions are summarized in this section and detailed in Chapter 11.

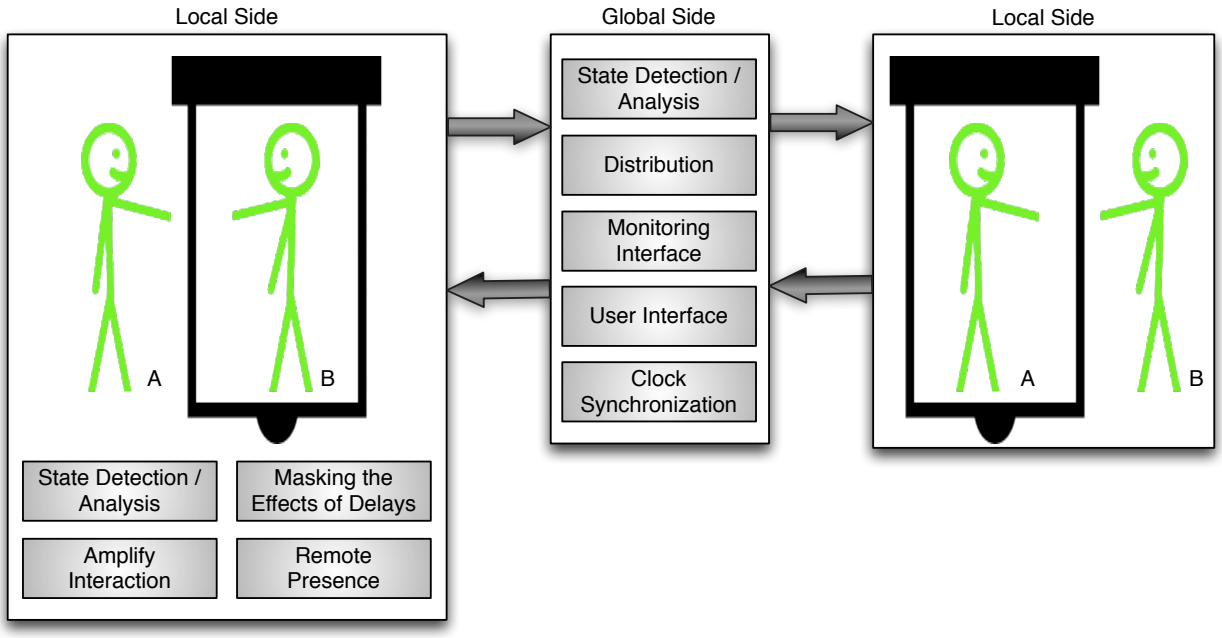


Figure 1.1: The required functionalities of distributed acting.

Figure 1.1 shows the proposed functionalities of distributed acting. The functionalities are mainly divided into local side functionalities and global side functionalities. It comprises 1. The local side includes functionalities to detect and analysis what local actors are doing, find out gestures performed by actors on local stage, amplify actors' interactions, masking the effects of delays and create the remote presence of actors on the display. 2. The global side receives data streams from local sides, it has functionalities to detect and analyze of all data streams, find out gestures performed by actors on different stages, deliver data streams to local sides, monitor the internal state of all computers, manipulate the system through a user interface and synchronize the clock of all computers. 3. Computers on both local side and global side have a monitoring process. It collects CPU utilization, latency, and other information. 4. An Administrator Interaction system provides interfaces to show the monitored information. It also provides

different options for the administrator, and allows the administrator to manipulate the whole system during performance.

1.3.1 Principles

1. **One Actor: One Data Stream**: One to one is the principle of having a separate data stream per actor from the begin point already, instead of having a data stream of everything and then do processing to extract each object/actor. To achieve this, we use a one to one mapping between each actor and the sensor (the prototype uses a single Three-Dimensional (3D) camera) used to detect an actor. Each camera will generate data, which will represent one unique actor. Instead of having a data stream of everything, this method will save the extra work of processing to extract each actor from the data stream.

2. Three models for remote interaction between actors have been identified:

   (a) **Act-By-Actor**: Actors react to the remote presences as if the latter were the actual actors. How good the interaction result is depending on the latency between stages.

   (b) **Act-By-Director**: Actors follow a script telling them when to start an action and what this action is. Actors are not interacting by observing and reacting to other actors, but an audience will see an illusion as if all actors do this.

   (c) **Act-By-Wire**: Software analyses and corrects the behavior of actors by modifying the behavior of remote presences. In the MultiStage system, the principle is implemented by blending in prerecorded data stream instead of using the data stream for the remote presence. This happens if the data stream is delayed more than a threshold value.

3. There are two different types of masking the effects of delays, coordination and substitution. **Act-By-Director coordinates** interactions to make local and remote events happen at the correct times. **Act-By-Wire substitutes** delayed data with data available at each stage, and tricks the audience to believe the actors are interacting with each other.

4. **Amplified Interaction**: There is a significant distance between actors on and audience at a theater stage. Exaggerated movements and makeup are used to make an audience better understand what actors are doing. For remote interactions across distance, there is not only a significant distance between actors and audience, but also between actors. Local actors usually interact with remote actors through a

display. Consequently, we need to amplify actors' interactions (such as gestures). For example, if computer detected an actor holding something, his remote presence on the display will be a knight holding a sword.

5. **Local vs Global Gestures**: A gesture represents a pre-defined command to the system to execute code to do some functionality. A gesture can be performed by actor locally on a stage. For example, one actor moves his arm up will be interpreted by the system to display the remote presence of this actor on the display. A gesture can also be performed by multiple actors located on different stages. For example, two actors on different stages raise their left arm together. This could be interpreted by the system to display a rainbow between those two arms.

6. **Reliability by Receiver Autonomy**: This is about the end-point being able to interpret and lay out the data streams as it sees fit instead of letting the begin-point dictate the layout. Sending stage will provide the receiving stages with individual data stream for each actor, and leave it to the receivers to use the data stream as they see fit. This can increase reliability of the system because stages can do local decisions on what to do with the stream in view of local needs, resource constraints and failures.

7. **Flexibility by Receiver Autonomy**: This is about the end-point being able to interpret and lay out the data streams as it sees fit instead of letting the begin-point dictate the layout. Sending stage will provide the receiving stages with individual data stream for each actor, and leave it to the receivers to use the data stream as they see fit. This allows each stage to do local decisions on how to present the distributed stage.

1.3.2 Models

1. **Decoupled Producer and Consumer with Subscription**: A publish-subscribe model [12] is found for benefit MultiStage system: It decouples stages. A stage produces a number of data streams. These streams can at will be picked up by other stages. This decouples the stages with regards to failures, local resource availability, and network bandwidth and latencies. MultiStage [2] (see Chapter 2) and Personal Video Distribution (pVD) (see Chapter 9) are two applications using this model.

2. **Masking the Effects of Delays through Coordination**: The model coordinates actions done by actors. It let actors do actions at the right time. It includes:

(a) Select one stage to be live, actors and data streams are synchronized at the live stage. (b) Delay local data to wait for data from other stages arrive at local stage.

3. **Masking the Effects of Delays through Substitution**: Data are manipulated to mask the effects of delays. Manipulations include just-in-time blending in of prerecorded data, and just-in-time blending in of on-demand computed data.

4. **Interactive System State**: At runtime monitoring of the state of all stages and of the end-to-end delays between stages. The state is used to modify the behavior of the system at runtime. Distributed acting requires a system with high availability and with knowledge of delays.

5. **Actor-Local Sensing**: To be able to provide data stream about actor, we need a technology sensing close by actors or being directed at each actor individually.

1.3.3 Artifacts

Sensor Suite: The sensor suite is built to do the **Actor-local sensing** model. The sensor suite comprises four 3D cameras, two computers, internal suite data networks and external network. Each stage has a sensor suite. The four 3D cameras are used to detect actors and analysis for gestures. The combination of four cameras has an almost 360-degree detection angel. Each computer connects to two cameras. Data about actor will be generated. Wired in-suite network: The computers are connected by wire to an access point. Wired or wireless external network: The access point is either connect by wire or wireless to the external network. We normally connect the access point to a 1 Gbit/s Ethernet.

Distributed State Detection: It detects actor and produces data streams about actor. Each data stream represents one actor. The data streams will be delivered to the distribution system.

Distributed State Analysis: It analyzes to find gestures done by actors from data streams. The data streams will be delivered to the distribution system.

State Distribution: The state distribution is based on the producer and consumer with subscription model. Data streams are delivered to the distribution system. The distribution system delivers data streams based on consumer subscriptions.

Remote Presence System: The Remote Presence system renders received data streams on a display. Local actor will interact with the remote presence of the remote actors.

Human Interaction and Collaboration System (Masking the Effects of Delays): The Human Interaction system is similar to a distributed director. It tells actors

when to start an action and what the action is. The Collaboration system applies various approaches to mask the effects of delays.

System Performance and State Monitoring System: The system is based on the **Interactive system state** model. The internal state about all computers running the MultiStage system will be monitored and displayed through a user interface. If an internal failure happens, the system can provide information to help computers recover from failure. The monitored latency and clock difference values are used by the Collaboration system to mask the effects of delays.

pVD: The pVD is developed based on the producer and consumer with subscription model. The system is designed for video sharing among a single user's devices. It avoids any third parties, and relies only on the user's personal computers.

1.3.4 Facts

Performance measuring experiments on the MultiStage system were conducted. The details are described in Chapter 8. The experiments can be divided into two parts: resource usage experiments and experiments about latency.

Resource Usage Experiment: In the largest configuration of MultiStage, we have three stages, and each stage has four cameras and each camera sends out data stream at 30 Frames per Second (FPS) and at a resolution of 5000 points. Each stage streams four data streams producing traffic of about 53 Mbits/s.

There is no measured loss of data on a Local Area Network (LAN) and only an insignificant loss of data on a Wide Area Network (WAN).

The computer with the highest CPU load in this case is still below 25 percent. This means the MultiStage system scales to at least three stages with a total of at least 12 outgoing and 36 incoming data streams. The CPU utilization and memory usage in all cases are either very low or low. We conclude that MultiStage can be supported by even low-end computers, and still have resources available for other applications and systems.

When stages connect through Wi-Fi and the global side is connected by wire to a Gigabit Ethernet, it supports 8 incoming and 8 outgoing data streams.

Latency Experiments: The System end-to-end one-way latency is the time between a physical event happening on a stage being picked up by the cameras, and a visualization of the actor being displayed on the same stage. It is about 100-158 ms when distribution computer is located at Oslo. The Actor-to-actor round-trip latency (different stages) is twice the system end-to-end one-way latency. It is 200-316 ms when distribution computer is located at Oslo. We subjectively observed that a user can notice that an action is delayed is about 190-225 ms delay were added. Tolerable latency is the actor can tolerate before the illusion of being on the same stage with other actors breaks. For rapid

hand movement, it is not tolerable when the actor-to-actor round-trip latency is 350-400 ms. For slow hand movement, it is not tolerable when the actor-to-actor round-trip latency is about 800 ms.

In Europe, it takes about 300 ms from an actor does an action until the actor sees the remote presence of another actor react. This means for the tight interaction like rapid hand movement, the system can expect to have to mask the effects of the delays.

We also measured the maximum system end-to-end one-way latency at which each masking approach is in principle at least partially successful at masking the effects of delays.

1. The Act-By-Actor approach just let actor interact with each other freely, no extra technique is added for this approach. The maximum system end-to-end one-way latency is about 190-325 ms.

2. The Act-By-Director approach has a script synchronizing all actors. But the remote presence about a remote actor can be out of synchronization with the local actor. This is because of it takes times to process and send data stream about remote actor to local. This approach is further explored and implemented into:

   (a) Live Stage approach synchronizes all remote presences on one live stage.

   (b) Delay Remote Presence approach synchronizes all remote presences, it also synchronizes all actors at every stage. But all actors and all remote presences at each stage can out of sync because the remote presences are delayed.

   The maximum system end-to-end one-way latency is about 390-525 ms.

3. The Act-By-Wire approach synchronizes all actors and remote presences at every stage by blend in on the fly created remote presence when data streams about remote actors arrive too late. No matter how high the latency is, this approach will always able to provide the illusion of actors is interacting together. But if latency goes too high, the original data streams about remote actors will be replaced.

1.3.5 Insights

Insights gained from the research.

- The set of functionalities we found that we needed for such a system is: 1. Detect actors and generate data about actors. 2. Analyze data for gestures. 3. Data distribution between stages. 4. Create remote presence on each stage. 5. Coordinate actors being at different locations. 6. Masking of the effects of delays when needed. 7. Monitoring of the state of the system.

- There are at least two ways to do the **Act-By-Director** approach:

    1. **Live stage**: Stages are divided into secondary stages and one live stage. Every stage runs a script telling actors when to do an action and what this action is. The latency between secondary stages to live stage is measured. Each stage receives an acting start time. The start time at a secondary stage will be adjusted to some time when the acting should start minus the delay between this stage and the live stage. Consequently, performances at secondary stages are started a little earlier than at the live stage such that when the live stage starts, the corresponding input from the secondary stages arrives just in time. Local actors at the live stage are in synchrony with the remote presences representing the remote actors on the live stage. However, local actors on a secondary stage will be out of sync with the remote presences of remote actors when latency goes high. Each stage can be the live stage for a time by switching which stage is the live stage at suitable points in the performance. The switch of live stage can happen during the intermission of the performance.
    2. **Delay local remote presences** and **delay locally the remote presences until data for the most delayed remote presence arrives**: Each stage starts their performance at the same time and follows a local director script. The remote presences at a stage are delayed until the data streams from the remote stages arrive. The delay can either be an average value or the largest value to wait until the last data arrives. The remote presences of actors on a local stage and the remote presences of actors on remote stages will be synchronized. However, the actors on a stage can be out of synchronization with the remote presences at the stage.

- The **Act-By-Wire** approach will create an illusion of an actor to make audience believe actors still interact with each other when several data streams arrive late. The illusion can either be an on-demand blend in of *prerecorded* data streams locally or on-demand blend in of a *computed* remote presence locally.

- Several approaches to mask the effects of delays from actors and audience can be applied. A study on different approaches to mask the effects of delays was done. Each approach makes interaction success at different delay thresholds. Different approaches to masking the effects of delays should be expected to be needed at different situations.

- Experiments of the latency investigated on different types of latencies: network latency, system latency, actor-to-actor latency, human noticeable latency and human

tolerable latency. We did informal user studies and found out that the actor-to-actor roundtrip latency is typically 300 ms in Europe. And the human tolerable latency for rapid movement such as handshake is typically less than 300 ms. This implies when interactions become fast and rapidly, different approaches to mask the effects of delays should be applied.

1.3.6 Mapping of Contributions to Research Questions

1. How to do low latency detection of multiple human actors on the same stage: Each detection side uses several 3D cameras to detect actors at a rate of 30 FPS. Each camera captures one individual actor, and data stream representing the individual actor is generated. The data stream can be 3D point cloud, Two-Dimensional (2D) image or a control commands indicate the arm movements about the actor. Each frame of data stream is timestamped. It includes both time and spatial information. Assuming the distributed system uses a shared clock, these information can later be used when recreate actor in a remote location.

2. How to do low latency detection of gestures done by actors on the same stage: MultiStage system includes both local gestures and global gestures. Local gestures are performed by actors on the same stage. Global gestures are performed by actors on different stages. After a data stream is generated, the system will analyze the data stream and look for gestures. Local gestures are detected locally on one stage. Global gestures are detected at a place where data streams from all stages are collected together.

3. How to do low latency distribution of state about human actors and gestures between stages: MultiStage system controls the network bandwidth usage by set each local stage to four cameras generate maximum four data streams about actors. The system uses a centralized method to manage the handover of data streams. Each stage pushes data streams to the centralized server, and receives data streams from the centralized server. The advantage is that all stages just need to know the location of the centralized server. When each stage sends messages to the centralized server, the server will know the location of this stage. To decrease the latency of state distribution, User Datagram Protocol (UDP) is used instead of Transmission Control Protocol (TCP). Because the packet retransmission technique in TCP will increase the transmission latency.

4. How to do low latency representation at a stage of remote actors: To let actors interact with remote actors, the remote presences of remote actors are created using data streams received from the distribution server. Each data stream represents about one individual actor. The data stream also contains spatial information used to create the actor into the right place on a display.

5. How to maintain for the remote actors, the illusion of being on the same physical stage: After the remote presence finished display data streams about actors, it may already been delayed too much. Actors on local stage may not be able to interact with the remote presences of remote actors being delayed too much. Therefore, the system should have function to **mask the effects of delays** as seen by actors and audience. And make them feel actors are interacting as if they are at the same place.

6. How to provide humans with the state of the system to aid in recovery after failures: The system has functionality to collects information including CPU utilization, memory and bandwidth usage, and latency between computers. The information will be shown to the user from a graphical interface. User can manipulate and control all stages from the interface. For example, send out commands to start and stop the performance.

1.4 Methodology

This dissertation uses a systems research methodology. A prototype system is developed, and experiments are conducted to objectively characterize its resource usage and performance characteristics.

1. Resource Usage Experiment: Experiments were conducted to measure several performance metrics of MultiStage system. Factors include: number of stages, resolution of images from cameras, number of cameras on each stage, and the location of the data distribution server, we name it Distribution of State Data Streams (DSDS) (LAN in Tromsø / WAN in Oslo). The extreme case is all cameras in all stages send the highest resolution of images to DSDS. Using the Python psutil module [13], we measured the CPU utilization, amount of physical memory in use, and incoming and outgoing network traffic (sent and received data) on all computers. Each computer has a monitor process keep logging the above information every one second. Each experiment was running for about five minutes. We calculated the average value of each performance metrics.

2. Experiments about Latency: We also conducted experiments to identify some of the effects of latency on actors. A high frame rate camera was used to capture both of the motion of real objects and the motion of the same objects in the display. In some of the latency experiments, we artificially delayed data about actors to simulate the effects of latency. The measurements were either done by count the frame difference between the real object and the same object in the display, or subjectively decide the value.

1.5 Limitations

1. Stage means a room, not a stage as found in a theater or a music hall. The advantages to use a stage include: more realistic environment, and we could get feedback from actors. However, the research is focus on document basic performance characteristics of the system rather than doing user studies. A follow up research could have expanded the current research into stages and actors.

2. Each stage contains four cameras. There is a one-to-one map between each camera and each individual actor. This means presently there is support for a maximum of four actors on each stage. The detail about the mapping is described in Chapter 5.

3. The system uses a centralized server for data distribution. The detail is described in Chapter 4.

4. For human-to-human interaction across long distance, the latency can never be totally removed. We try to create an illusion of actors being on the same stage. The detailed explanation is in Chapter 6.

5. Experiments were conducted to measure several objective performance characteristics of MultiStage. The research focuses on presenting the architecture, design, implementation and performance of a system for human-to-human live interaction across distance. In the experiments about latency, the values are subjectively decided by the researcher himself, his colleagues or his supervisor. This is an informal user study and it is based on a few people's opinion. No formal user studies were conducted.

1.6 Publications

This chapter lists all published papers and the contributions of each paper. Several tables are used to connect contributions to papers, and connect papers to chapters.

To describe the contributions of each paper, the dissertation's structure of contributions is used. The chapter numbers are used to show where in this dissertation each contribution is related.

The publications include:

1. MultiStage: Acting Across Distance.

2. Masking the Effects of Delays in Human-to-Human Remote Interaction.

3. pVD - Personal Video Distribution.

1.6.1 MultiStage: Acting Across Distance

This publication reports on a prototype system helping actors on a stage to interact and perform with actors on other stages as if they were on the same stage. This publication gives a brief introduction about the MultiStage subsystems. The contributions are listed below.

- Principles
  - One Actor: One Data Stream (Chapter 3).
  - Amplified Interaction (Chapter 2.7).
  - Local vs Global Gestures (Chapter 2.7 and 3).
  - Reliability and Flexibility by Receiver Autonomy (Chapter 5).
- Models
  - Decoupled Producer and Consumer with Subscription (Chapter 4)
  - Interactive System State (Chapter 7.2).
  - Actor-Local Sensing (Chapter 3).
- Artifacts
  - Sensor Suite (Chapter 2.7).
  - Distributed State Detection, Analysis (Chapter 3).
  - State Distribution (Chapter 4).
  - Remote Presence System (Chapter 5).
  - System Performance and State Monitoring System (Chapter 7.2).
- Facts
  - Resource Usage Experiment (Chapter 8.3.1). The experiments conducted on wired network are described in this publication.

1.6.2 Masking the Effects of Delays in Human-to-Human Remote Interaction

This publication reports the subsystem of MultiStage to masking the effects of delays. The contributions are listed below.

- Principles

  - Three models for remote interaction between actors: Act-By-Actor, Act-By-Director, and Act-By-Wire (Chapter 6).
  - Two different types of masking the effects of delays, coordination and substitution (Chapter 6).

- Models
  - Masking the Effects of Delays through Coordination (Chapter 6.2).
  - Masking the Effects of Delays through Substitution (Chapter 6.2).

- Artifacts
  - Human Interaction and Collaboration System. Human Interaction system coordinates and tells actors when to start an action and what this action is (Chapter 7.4). Collaboration system applies various techniques to mask the effects of delays (Chapter 6).

- Facts
  - Latency Experiments (Chapter 8.3.2 and 8.4).

1.6.3 pVD - Personal Video Distribution

This publication presents the architecture, design and implementation of the pVD prototype. pVD supports sending and viewing live and stored videos between any of a single user's computers, and allows for a smooth handover of play back between computers. The system avoids any third parties, and relies only on the user's personal computers. pVD is another implementation of the decoupled producer and consumer with subscription model. The contributions are listed below.

- Models
  - Decoupled Producer and Consumer with Subscription (Chapter 4).

- Artifacts
  - pVD (Chapter 9).

- Facts
  - Resource Usage Experiment. The experiments conducted on wireless network are described in this publication (See Chapter 9.5.2). A comparison of resource usage on wired and wireless network is described in this publication. (See Chapter 9.5.3)

1.6.4 Mapping of Contribution and Publications

Table 1.1 links contributions to each publication.

Contribution		Publication		
		MultiStage [2]	Masking [14]	pVD [15]
Principles	One Actor: One data stream	✓		
	Act-By-Actor, Act-By-Director, Act-By-Wire		✓	
	Amplified Interaction	✓		
	Local vs Global Gestures	✓		
	Reliability and Flexibility by Receiver Autonomy	✓		
Models	Decoupled Producer and Consumer with Subscription	✓		✓
	Masking the Effects of Delays through Coordination, Substitution		✓	
	Interactive System State	✓		
	Actor-Local Sensing	✓		
Artifacts	Sensor Suite	✓		
	Distributed State Detection, Analysis	✓		
	State Distribution	✓		
	Remote Presence System	✓		
	Human Interaction and Collaboration System		✓	
	System Performance and State Monitoring System	✓		
	pVD			✓
Facts	Resource Usage Experiment	✓		✓
	Latency Experiments		✓	

Table 1.1: Map of each publication to contributions. The name of each paper is shortened.

1.6.5 Mapping of Publications and Chapters

Table 1.2 maps publications to chapters of this dissertation.

Publication / Chapter	2	3	4	5	6	7	8	9
MultiStage: Acting Across Distance	✓	✓	✓	✓		✓	✓	
Masking the Effects of Delays in Human-to-Human Remote Interaction					✓	✓	✓	
pVD - Personal Video Distribution							✓	✓

Table 1.2: Map publications to chapters of this dissertation.

1.7 Organization

The rest of the dissertation is organized as follows:

Chapter 2: This chapter gives a brief description about the architecture of, design of and implementation using the MultiStage system. More detailed introduction about each subsystem is described from Chapters 3 to 7.

Chapter 8: This Chapter describes the experiments conducted to evaluate the MultiStage system. Several studies are also conducted to verify how much latency can be tolerated by using the approaches of masking the effects of delays.

Chapter 9: This chapter describes the architecture of, design of, implementation of, and experiments using the pVD system. This is a prototype system for video streaming for a single user's devices.

Chapter 10: This chapter discusses the research questions in this dissertation. The pVD and MultiStage systems are also discussed in this chapter.

Chapter 11: This chapter summarizes the contributions of the dissertation.

Chapter 12: This chapter summarizes the whole dissertation.

Chapter 13: This chapter describes ideas for future work.

Appendix: Published papers are listed in the appendix.

Chapter 2

Overview of MultiStage: Acting Across Distance

2.1 Introduction

We envision computer-mediated collaborative performances where actors at physically remote locations, interact and coordinate their actions as if they are next to each other on the same stage or in the same room. Through various means, including audio, video, and animations, each actor has a remote presence at one or several remote stages. As illustrated in Figure 2.1, four dancers on three different stages dance together. Each stage is equipped with sensors to detect actors and a display to visualize the remote presence of all the performers. The rope and knot represent the global system binding the stages together. We are interested in how to mask the effects of delays and distance.

We describe a system that does this for the visual side of a remote presence: MultiStage collects state, including video, about each stage through various sensors, including cameras and microphones. MultiStage also analyzes the observed state to identify information including actor gestures. State data and information are streamed between stages to maintain a remote presence for each actor. The data is also used to monitor and control the system.

Each stage has several incoming data streams that are used to create the remote presence of the remote actors. Actors on a stage watch and react to the remote presence of the other actors. There can also be several third parties, audiences, just observing, and not directly participating. Audiences can be physically present at any of the stages, or be on the Internet. An audience local at a stage can watch the local physical events unfolding, and watch visualizations of both the local and remote events.

However, what audience and actors observe will differ to some extent because of delays from when an action happens on a stage until it can be viewed and reacted upon on the

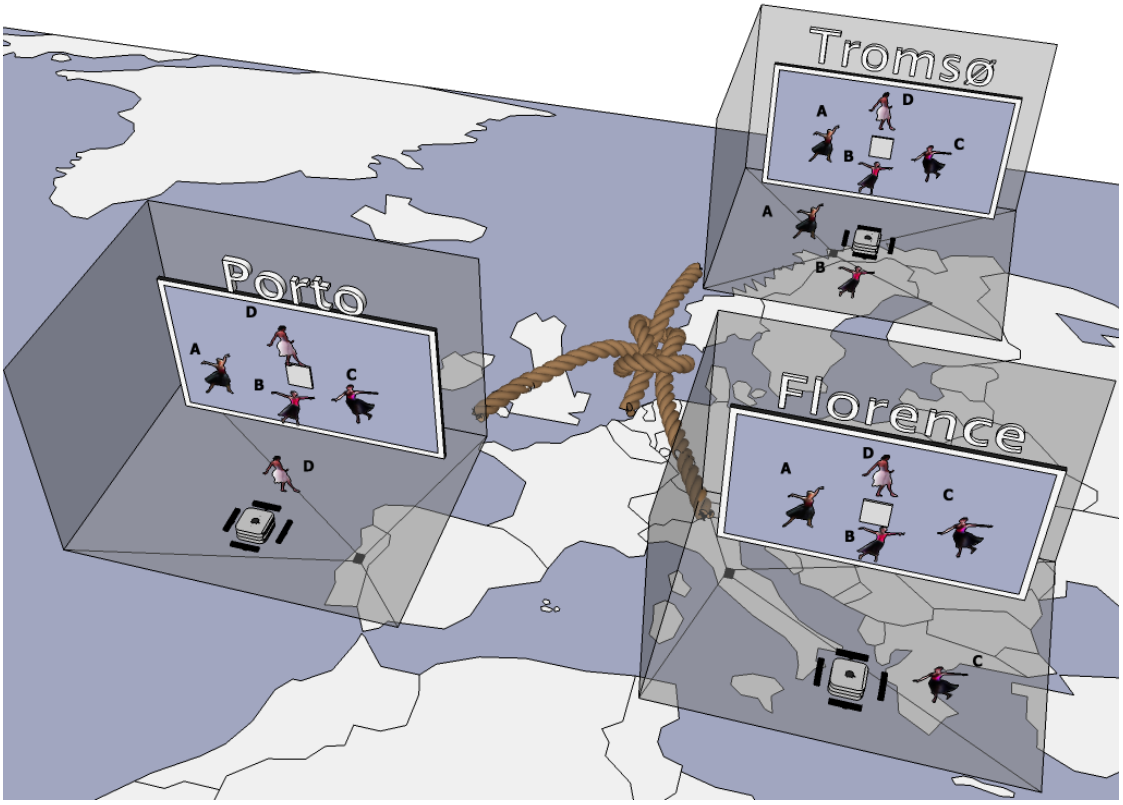


Figure 2.1: Four dancers on three different stages dance together. Each stage is equipped with sensors to detect actors and a display to visualize the remote presence of all the performers. The rope and knot represent the global system binding the stages together.

remote stages.

In principle, there will always be some delay from when an event happens until it can be observed. Light alone needs 134 ms to travel the length of earth's equator. In practice, the total delay when observing a remote event includes delays coming from the sensors, transferral of data from sensors to computers, processing of the sensor input, network transmission, on-route processing, receiving and processing the received data, and processing and visualizing the data locally. Even if the delays can be reduced, they can never be removed. Consequently, we have to live with the delays, and find ways of reducing the effects they have on the actors and the audiences. The effects of the delays can be reduced through different techniques including on-the-fly manipulation of the remote presence representation of actors.

We must also mask the effects of distance. On a theater stage the actors use several techniques including costumes, makeup, and exaggerated movements to reach out to the audience. We propose to let a user instruct the system through gestures as well as automatically add enhancements to the remote presence. For example, a given arm movement could be turned into a text bubble above the visualization of the user, or a

glowing halo around the arm. We term this *amplified interaction*.

There are many commercial teleconferencing and messaging systems where two or several persons interact through instant text, video, and audio, as well as file transfer. The latencies can be tolerable. However, teleconferencing systems are best when used in interaction without interactively fast-synchronized movements of participants. Teleconferencing systems are typically not flexible with regards to manipulating remote presences, and how they are arranged on, for example, a display. They also lack functionalities for amplified interaction.

This dissertation reports on MultiStage, a multi-stage collaboration system masking the effects of delays and amplifying the remote presence of actors. It is designed to scale to at least three stages connected with both LAN and WAN (Internet). Each stage can have multiple actors. We use four 3D cameras per stage glued together to give approximately 360-degree coverage. Each stage also has at least one display, but can have several more. The system extracts a 3D recording of each actor. The system also sends data streams into the system for distribution between the stages. The data streams are analyzed on-the-fly for information such as gestures. Each stage receives several data streams, and creates remote presences of the remote actors. The system applies several techniques to mask the effects of delays, including on-the-fly blending in of prerecorded data streams or on-demand animations into the visualization of remote presences if delays prevent the data streams from arriving in time.

2.2 The idea of MultiStage

The stages on MultiStage will be located in different countries or at different continents. Data streams about actors on local stage will be transferred to other remote stages with delays. Because the delay can never be removed, the system needs to mask the effects of delays and keeps the causal order of interactions between Actors.

Amplified Interaction: On a theater stage, with a significant physical distance between actors and the audience, bold makeup, clothes, and exaggerated movements are used to better project to the audience what the actors are doing.

The MultiStage includes multiple functionalities: functions locally to a stage (local function), functions for binding stages together (global function), and functions that are included by both local and global function.

- Local function: 1. Each stage has functions to detect and analyze what local actors are doing. 2. create the remote presences of both local and remote actors with their interaction amplified and delays masked.

- Global function: The global functions include: gather and analyze stage messages (find out global gestures), distribute the messages back to the stages, and manage the whole MultiStage system.

- Local and global function: In order to keep causal order of interactions between actors, all stage computers must have the same view of the time. In order to mask the effects of delays and maintain the same clock, there should be a function running on each computer keep monitoring the latency and clock difference between computers.

2.3 Architecture of MultiStage

Figure 2.2 shows the MultiStage subsystems. Local State Monitoring (LSM), Local State Analysis (LSA), Global State Monitoring (GSM), Global State Analysis (GSA) and Remote Presence are mainly researched by my colleague Giacomo Tartari. For more information, please refer to the MultiStage [2], and Global Interaction Space [3]. The Distribution of State, Collaboration Management, Controllable Temporal Synchronization, Shared Clock, System Management, and System Performance and State Monitoring are mainly researched by me.

A - LSM: The local state is detected and several streams of state are produced for further analysis.

B - LSA: The state is analyzed on-the-fly to detect significant state like the number of users on a stage, and user gestures. Collective and collaborative gestures comprised of gestures from several humans are detected as well. Several new state is produced representing detected local state.

C - GSM: The local state from each stage is collected and aggregated at GSM. The state will be used at GSA to analyze global behaviors including gestures.

D - GSA: The states from all stages are analyzed on-the-fly to detect distributed state like collaborative gestures comprised of gestures from multiple stages. Several new states are produced representing detected global state.

E - Distribution of State: Remote state is made available at the stages. A stage has control over which state is made available to it. The extreme case is to make all state available at all stages.

F - Remote Presence: The local state and the generated global state are used to create remote presences of actors.

G - Collaboration Management: A functionality to coordinate all users interactions. This includes informing a user on when a specific action, like moving an arm up

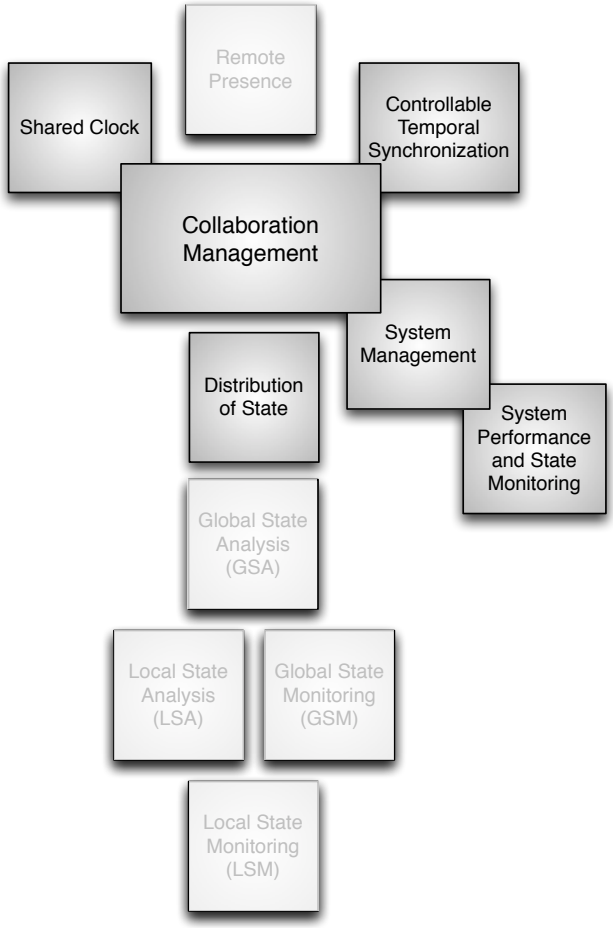


Figure 2.2: The architecture of MultiStage. The light grey box indicates the MultiStage subsystems which are done by my colleague Giacomo Tartari.

or down, should be executed and what they should do or be aware of. The functionality will also let humans interact with each other through gestures and input devices to coordinate their actions including letting a human tell the system at which time in the future a certain action or sequence of actions will be initiated and executed.

H - Controllable Temporal Synchronization: Local and remote users and their remote presences are temporally synchronized through a range of techniques to mask the effects of delays, including adding delays to the remote presences of local actors, scheduling of the play back of data streams, and blending prerecorded and on-the-fly created animations of users with live data streams.

I - Shared Clock: All stages share a reference clock so that activities can be ordered in time.

J - System Management: A functionality letting a human manage the system through system wide commands including booting, synchronizing local clocks, and getting

visualization of internal system-wide state.

K - System Performance and State Monitoring: The collaboration system is monitored to identify its internal state including failures and readiness, as well as measuring performance metrics including clock difference, network latencies and bandwidth, and CPU loads.

2.4 Design of MultiStage

The MultiStage prototype, (Figure 2.3), has a local side and a global side. Each side comprises several systems. The capital letter in bracket in the subchapters shows the mapping to architecture.

2.4.1 Local Side

The local side primarily focuses on what is happening locally on a stage. The local side has sensors to detect actors' movements, identifying relevant gestures from the movements. After receiving the data streams from the global side, the local side then displays the remote presence of an actor on a display with techniques to mask the effects of delays and amplify the actor's interaction.

1. The Local State Detection system includes LSM (A) and LSA (B). Every stage has a set of sensors, recording of actors' actions on the stage. Several streams of state data are produced by LSM. LSA does on-the-fly analyze of the data to find interesting objects and events in the data streams. The data is streamed to the global side for further analysis and distribution to other stages.

2. The Remote Presence system (F) subscribes to data streams, and creates remote presences of remote actors. The streams of state data from remote stages comprise information used to do remote presence of users and their actions through various techniques including playing and manipulating videos, creating animations of remote events, and reacting to remote gestures possibly having a local physical effect through actuators and robots. Presently the primary remote presence technique is to visualize remote actors on a very large display on each stage.

3. The Human Interaction system (G) coordinates actors on different stages by informing them about when they should start actions, and what this action is, such as moving arms, according to a given script.

4. The Collaboration system (H) applies various techniques to mask the effects of delays.

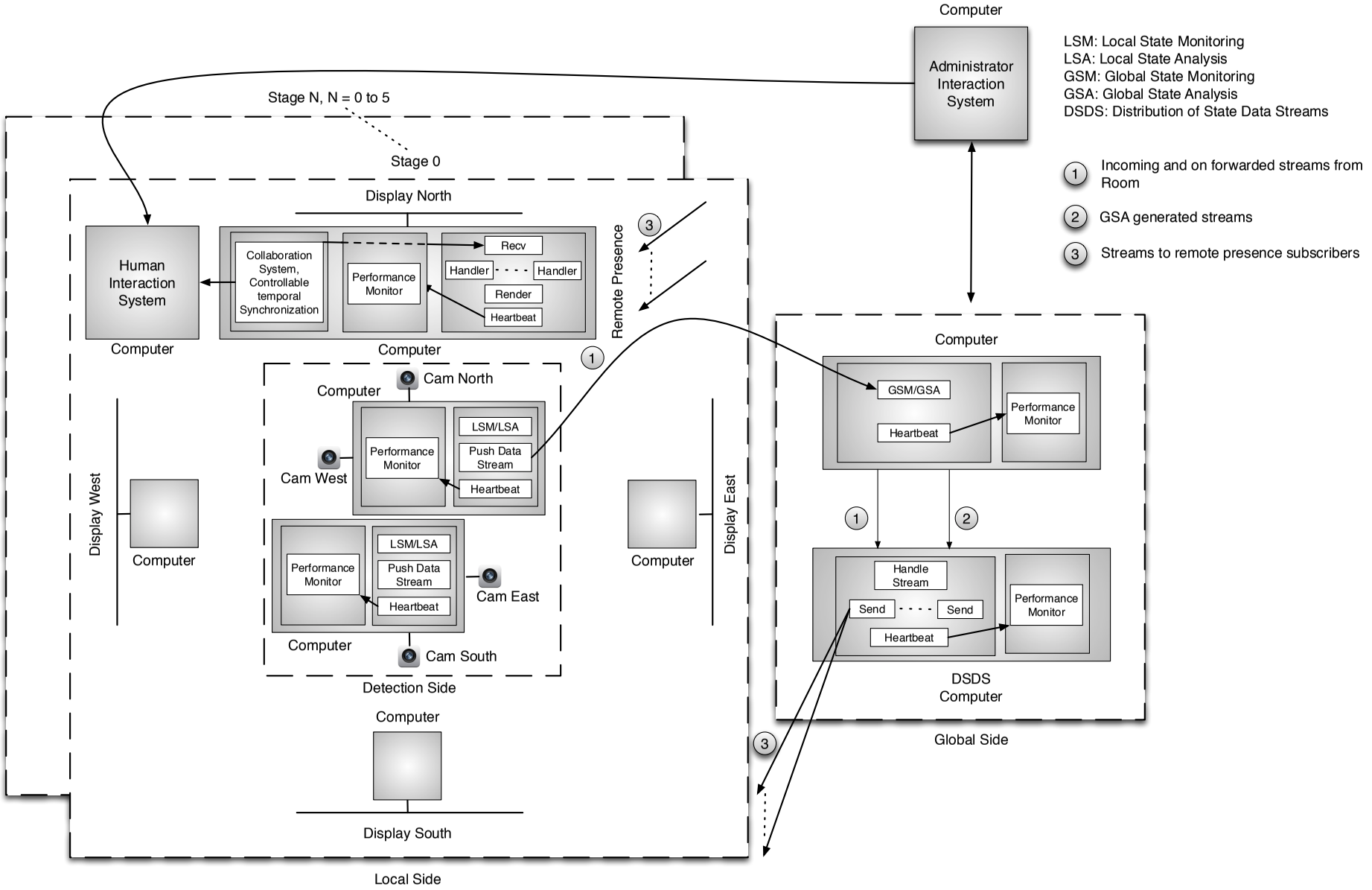


Figure 2.3: The design and implementation of MultiStage showing the system at each stage and the global systems binding the stages together.

2.4.2 Global Side

The global side is the glue binding the stages together, taking care of the distribution of data between stages, and analyzing of data from multiple stages. The global side receives data streams from the stages and redirects the data streams to the right subscriber at local side stages. The global side also synchronizes the clock among all stage computers and monitors the internal state of all computers. The global side includes:

1. The Administrator Interaction system (J) lets an administrator manage the system, and administrator can also set start times for performances.

2. The Global State Detection system includes GSM (C) and GSA (D). GSM collecting data from all the stages, and making it available for on-the-fly GSA. GSA detects distributed state-like collective gestures and collisions when actors on different physical stages occupy the same volume on the virtual stage.

3. The DSDS (E) manages subscriptions from stages for data streams, and does the transmitting of data to the remote presence computers locally to the stages.

2.4.3 Other Systems

Both the local and global side executes the System Performance and State Monitoring system (K), doing live performance measurements of several metrics including latency and bandwidth. These are made available to the global side's Administrator Interaction system. The performance measurements are also made available to the Collaboration system.

Shared Clock (I): The clock on every running computer will be synchronized. This can be achieved by instructing each computer to do a simple clock update from a common time signal like Network Time Protocol (NTP) [16].

2.5 Implementation of MultiStage

2.5.1 Local Side

The systems implementing the local side include the following:

1. The Local State Detection system has functions to detect and analyze the local state. Data streams represent the local state will be streamed to the global side. The communication protocol is UDP.

2. The Remote Presence system has functions to send out subscription messages to DSDS, receive data streams from the Collaboration system, and display the received data streams on the display. The communication protocol to receive data streams is UDP. The communication protocol for manage subscriptions is TCP.

3. The Human Interaction system informs when and what the actors should perform an action on a display. In the current implementation, a countdown is displayed on the display. After the countdown finishes, an image will appear on the display and tell actors what to do. Actors will perform actions according to what is shown on the display.

4. The Collaboration system runs on the same computer with the Remote Presence system. It receives data streams from DSDS and applies different techniques to mask the effects of delays. Then it will forward the data streams to the Remote Presence system.

2.5.2 Global Side

The global side includes the following systems:

1. The Administrator Interaction System has functions to receive measured data from the System Performance and State Monitoring system, show the measured information to the administrator through a graphical interface, and send control commands to manage the whole MultiStage system.

2. The Global State Detection system has functions GSM and GSA will monitoring and analyze the received data streams from local side and find out global gestures. New data streams representing global gestures can be generated. The data streams will be forwarded to DSDS. The communication protocol is UDP.

3. The DSDS has functions to receive data streams from the Global Detection system. After DSDS receives data streams, it will transmit the data streams to the remote presence computer on the local side. The communication protocol for transmitting data streams is UDP. DSDS also manages subscriptions from local stages for data streams. Each data stream has an unique id. The Remote Presence computer on the local side sends subscription message includes the data stream id to DSDS. DSDS will forward data streams according to local side subscriptions. The communication protocol for handling the subscriptions is TCP.

2.5.3 Other Systems

System Performance and State Monitoring system: As shown in Figure 2.3, each computer has a thread keeps telling if they are executing or not. This will help to find out internal failure as soon as possible. The monitor keeps measuring local computer's internal state including CPU loads, bandwidth, and memory usage. It also keeps measuring the latency and clock difference between DSDS and other computers. This information will be made available to the Administrator Interaction System for further use. The communication protocol is UDP.

Shared Clock: There are several ways to synchronize the clock on different computers. Each computer has an internal NTP protocol keep synchronizing the clock with its local server. The administrator can manage the Administrator Interaction System to send out command to all computers. It can also send command to tell all computers do NTP clock update. The System Performance and State Monitoring system will keep measuring the clock difference between DSDS computer and other computers. Before performance start, each computer will adjust their clock according to the newest measurement.

The systems were implemented on the operating systems Linux and Mac OS X. The programming languages include C, Python and the Go programming language [17]. The OpenKinect libfreenect library is used to fetch RGB and depth images from the Kinect cameras.

We will expand on the design and implementation of each component from Chapter-3 to Chapter-7.

2.6 Temporal Causal Synchrony between Actors

Some actions by actors are causally related. One actor performs an action, and some time later another actor performs an action because of the first action. A system must preserve the order of these actions when they are causally related.

Even if causality is preserved, there is a delay between an action and the corresponding reaction(s), and the system should ideally keep the delay low enough to make actors experience interactions as if they were on the same physical stage. Assuming that causality is preserved, how large the delay is indicates how well actors are in temporal causal synchrony.

We define actors to be in *loose* temporal causal synchrony with each other when there are no special demands on delays. This is typically the case in unstructured interaction where it does not matter a great deal if actions by actors are slightly delayed. This will typically be the case in teleconferencing with approaches such as Skype.

However, for structured interaction with coordinated movements, as in synchronized

dancing and in rapid action-reaction situations such as, for example, martial arts, correct causal ordering and short delays become critical in preserving the illusion that the actors are on the same stage. We define *interactive* temporal causal synchrony to be when actions by an actor are seen in causal order and as fast as actors are used to when being on the same stage.

Delays are unavoidable, and they can be large and even varying enough that interactive temporal causal synchrony cannot be achieved. In these cases we must mask the effects of the delays to create an illusion of synchrony. The system provides several approaches to mask the effects of delays. This topic will be expanded in Chapter 6.

2.7 Amplified Actor Interaction and Gestures

This part is mainly researched by collaborator Giacomo Tartari. A brief introduction can be found at MultiStage [2]. Further details can be found at Global Interaction Space [3].

In remote interactive performances there is distance not only between the actors and an audience, but also between the actors. Consequently, the actors need their remote appearance, movements and gestures to be amplified such that they become easier to see and understand both for the other actors and for the audience. In this way we extend the range of human interaction to remote locations and enrich the communication between them.

To be able to detect what an actor is doing, we surround him with an interaction space [18]. An interaction space detects human movements, and analyzes them looking for gestures. A gesture represents a predefined command to the system to execute code to do some functionality.

A gesture can be simple, such as raising an arm, or complicated such as doing two-arm movements. A gesture can also be active such as walking in a specific direction or passive as in standing still posturing. A collective (collaborative) gesture is a combination of gestures from multiple actors. Collective gestures can happen on the same stage, or be distributed, comprised of gestures from multiple stages. For example, when two actors on different stages, within some short time span, raise their left arms above their head, this can be interpreted as, for example, a command to the system to animate a flash of lightning between the two raised arms and display it on all the displays.

Based on the gestures, we can create effects in the remote presence manifesting itself in remote rooms. A user's arm movement can in the remote presence be amplified by having a text bubble appear in the video, and by adding other visual effects to the representation of the user. The user's remote presence can even be enhanced by executing a model of the user and using its output to create a remote presence.

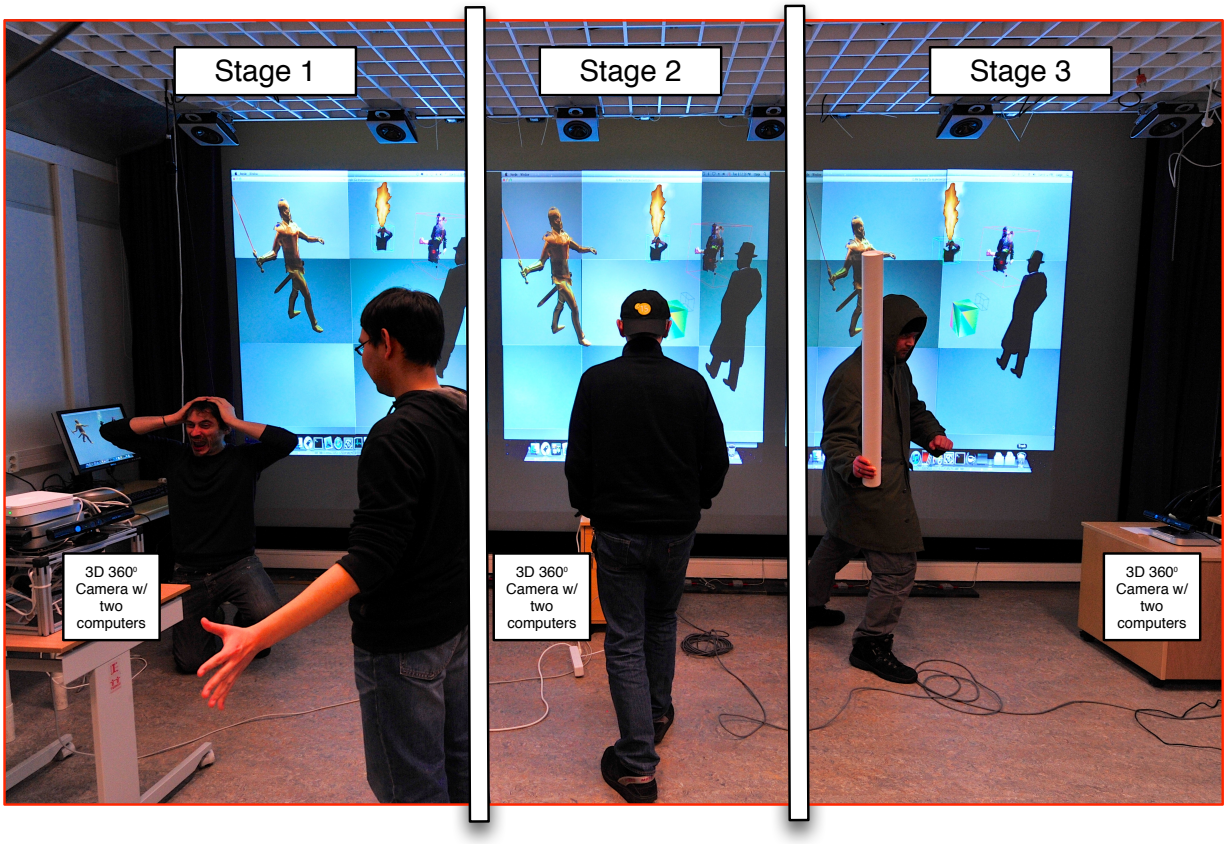


Figure 2.4: MultiStage is set up with four actors on three stages. Each stage has its own camera rig. Each stage displays all actors. The global system binding together the stages are located either locally connected to the same LAN at Tromsø or on a remote computer across the Internet. Note: the flame animation has been enhanced in the figure for better visibility. In order to illustrate the idea, the three amplified remote presences in this figure were predetermined to be what they are.

To experiment with the system, we set up three stages: Stage 1, 2, and 3, (Figure 2.4), in a single room. There were two actors on Stage 1, and one actor on each of the other two stages. Even though all three stages were co-located in the same room, they each occupied a different area of the room, and they each had their own interaction space and display. The animations and 3D models are rendered using the Horde3D graphical engine [19]. Each interaction space used four Kinect 3D cameras, (Figure 2.5). The cameras were arranged in a square with two computers receiving camera output and processing the images. Four Kinects arranged in a square cover almost 360 degrees. We typically placed the camera rig in the middle of a stage, and acted around it. The room where the stages are located has a large 6-meter by 3-meter display wall. Each stage displayed the remote presences of local and remote actors onto its assigned area of the display wall.

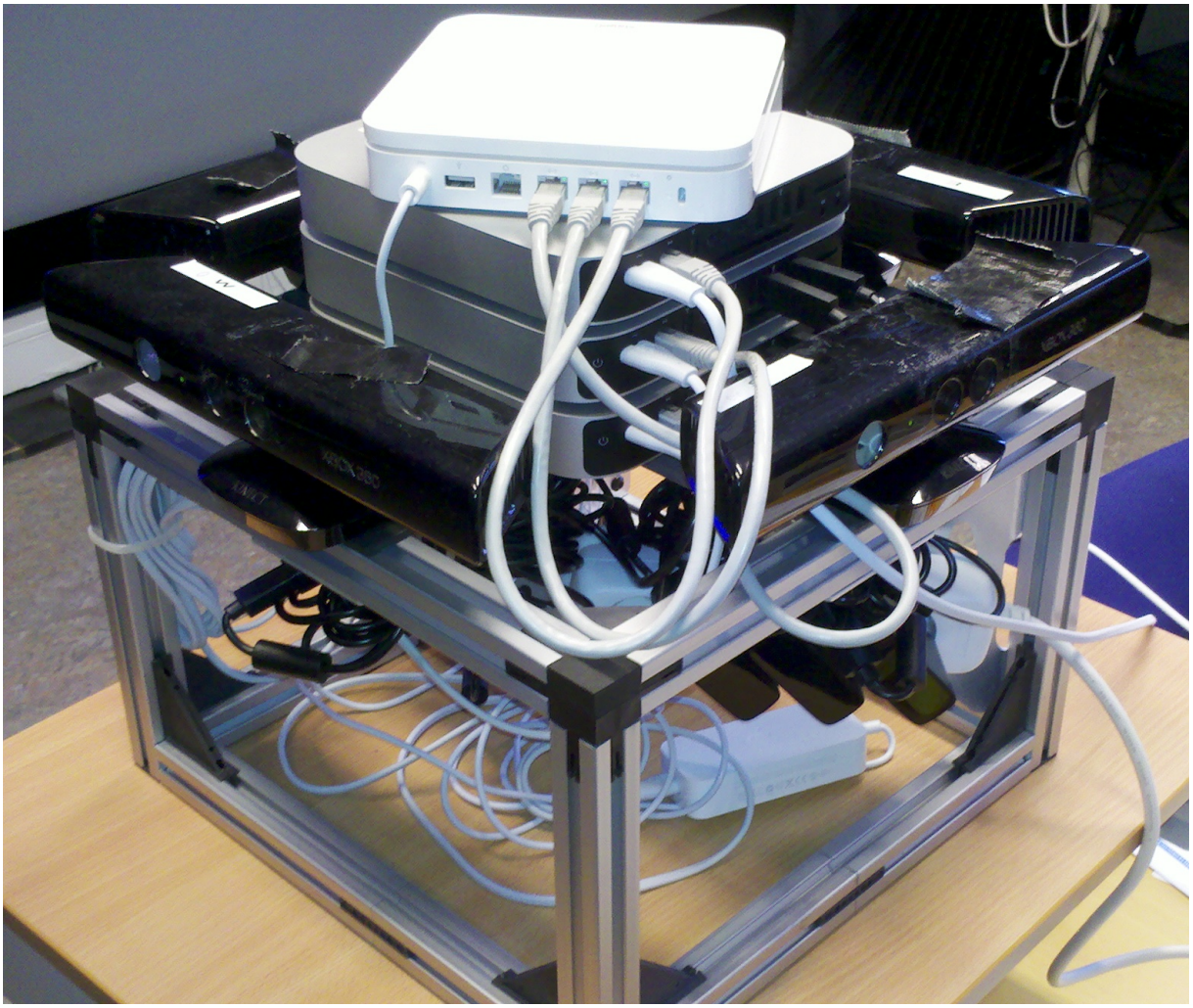


Figure 2.5: The four 3D Kinect camera rig used on each stage for almost 360-degree coverage.

To simulate both the situation when all stages are on the same LAN as well as when they are connected through a WAN, the Internet, we locate the global side handling the distribution of data between the stages either locally at Tromsø or at a computer in Oslo.

The images picked up by the cameras are analyzed and sent as data streams to all stages. This data represents the actors and to some degree what they are doing. The data is used to create a remote presence of each actor. This can take the form of a simple video, a manipulated video, or an animation of the actor as illustrated in the Figure 2.4. Each stage has a display where the remote presence of each actor is displayed inside the same virtual stage.

On the virtual stage, three of the actors have been amplified. On Stage-1 the kneeling actor with hands on his head is interpreted by the system as showing agitation, and the system has added an animated fire above his remote presence. The other actor on Stage-1

does nothing the system recognizes, and a low-resolution video of him is displayed on all stages. The actor on Stage-2 knows that if he keeps his hands in his pockets, has a hat on, and emulates walking, his remote presence will be that of an animated figure of a walking man with a long dark coat and a hat. The actor on Stage-3 knows that if he has something looking like a sword in his right hand, his remote presence will be that of a knight with a sword.

2.8 Related Literature

Several research systems for collaboration exist. The Distributed Immersive Performance (DIP) system [6] and [7] is a real-time, two-site distribution system for live and interactive musical performance. Musicians at two different locations interact with each other using media streams. DIP is more focus on audio. They also try to mask the effects of delays on audio. The MultiStage system masks the effects of delays on video, the system has three stages and DIP just has two stages. We discuss the DIP system in more detail in Chapter 6.

Telegnosis [5] is a remote camera system for teleconferencing. The system has an omni-directional camera as well as a Pan/Tilt/Zoom (PTZ) camera and supporting user cooperation between a local and a remote room. The system has a stream switching function, this function mixes multiple video streams from different rooms into one image and the desired image can be shown by changing the transparent rate from 0 to 100%. Using the above function the system allows users watch 360-degree conference room using the omni-directional camera as well as let users see more specific objects by remote control the PTZ camera. According to all three applications of the Telegnosis system, it has multiple senders send out video streams to one receiver. The receiver has the stream switching function allow users to view panorama images from all rooms or watch specific images. In all of the applications, the system gathered and watched video images at the receiver's place. It is more suited for the monitoring use case. In MultiStage, each stage sends out local data streams to all other stages as well as receives and displays data streams from other stages. Rather than just monitoring stages, the MultiStage system allows actors on one stage interact with actors on other stages.

Three's Company [9] is a fixed three-room distributed collaboration system, allowing three people to collaborate over a virtual workspace. In each room there is a multi-touch table, camera, speaker, microphone, and two Liquid Crystal Display (LCD) monitors to display the two other rooms. Three people doing collaborative work through a multi-touch table. The paper focus on let people collaborate through a virtual space. For example, in one of their applications, they let participants to construct logos together by

using the given tiles. Such application can tolerate up to few hundreds ms delay. It is not very time critical. In our application, actors may perform fast-synchronized actions with other remote actors. In such case, latency becomes a very critical factor, because actors must see the remote presence of remote actors fast enough. Our system must consider the effects of delays.

Geminoid HI-1 [20] is a remote-controlled android system using a human-like robot. The state of the android includes idle, speaking, listening, left-looking, and right-looking. A teleoperator controls the android's behavior by choosing its state. People can communicate with the android robot. The authors conclude that using an android gives a strong remote presence to the communicator. This system has the function similar to the amplified interaction function in our MultiStage system. But this system only uses one android to create a remote presence of a single person. The MultiStage system amplifies the interaction of multiple actors. It also supports user gestures and gives user more ways to interact.

Mingsong Dou et al. [21] propose a system intended for informal meetings between rooms. The system merges the images from panorama cameras acquiring the background of a room, with a camera acquiring the users when they are close to the display. The system amplifies the remote presence of the users by allowing users to maintain eye contact during a conversation with users in the other site. This system limited to two sites, and more focus on the study of user's eye contact. In their implementation, they ignore the network transmission latency by connect displays and cameras on two sites into one same computers. MultiStage system supports different gestures, it also studies different types of latencies and try to mask the effects of delays.

Grimage [22] is a multi-camera real-time 3D modeling system for telepresence and remote collaboration. 3D models of users are computed from 2D images from multiple cameras, and the 3D models are streamed to remote rooms where users are visualized in a virtual 3D environment. Computing and visualizing collisions and reaction forces between virtual objects in the virtual space strengthens the remote presence. The system is built on top of a middleware that simplifies the use of a compute cluster to obtain 3D meshes and textures from the cameras. The system place two platforms in the same room, but in the real case, the two platforms can be placed at different places with high communication delay.

Slim Essid et al. [23] proposed a multi-modal corpus for research into human-to-human interaction through a virtual environment. The virtual environment is defined as a virtual dance studio where a dance teacher can teach students choreographies. Both teacher and students are represented in the virtual studio by 3D avatars. The corpus consists of the recordings of the 3D avatars and outputs from other sensors, such as cameras, depth

sensors, audio rigs, and wearable inertial measurement devices. A dance instructor and a musician also provide some ground truth annotations for the corpus. Although the above system supports interaction such as virtual dance. But MultiStage has amplified interaction function and gestures.

3D helping hands [24] is a telepresence system. It mixes virtual and physical reality together and enables more than two participants to interact together. A remote helper can assist a physically distant worker to perform manual tasks by using remote hand gestures. The hands of remote helper are mixed together with the physical worker's hand together. Then the worker can mimic the movements of the remote helper's hands to perform tasks. The system supports real-time collaboration and uses of Mixed Reality techniques to enhance the worker's workspace and show it to the remote helper. The collaboration of the 3D helping hands more focus on let the remote user provide guidance to the worker. It lacks of the support for a two-way interaction, such as act and react type of interaction.

LiveMask [25] is a telepresence system that tracks the remote user's face and extracts their head motion and face image. The remote presence is a human face-shaped screen that conveys a user's non-verbal communication, such as the user's direction of gaze. The system focuses on user's head gestures, and face tracking. While the MultiStage system focus more on user's body movements.

TeleHuman [26] is a cylindrical 3D display portal for life-size human telepresence. The system supports 360-degree motion parallax. The viewer moves around a cylindrical display and the stereoscopic 3D display displays the remote presence of a remote person. The 3D videoconference system provides 3D capture, transmission, and using a lightweight approach, displays the remote person. A novel implementation by put human into a cylindrical display. MultiStage also supports gestures and consider the effects of delays when user interact across distance.

Table 2.1 compares MultiStage with other systems. MultiStage has a 360-degree multiple-camera rig. Users can roam a stage freely. The camera detects both background objects, and foreground objects such as users, and encodes them into separate data streams. The data streams are analyzed to detect gestures, and to do visualizations with the purpose of achieving remote presence. Each individual visualization can be mapped onto any display at any stage, making for a very flexible combination of background, objects, and users.

The system can scale to at least three stages, each with four cameras and multiple users, when the stages are connected through a switched Ethernet or through the Internet. No processor, memory, or network bottlenecks are encountered.

System	#stage	#user	Arch	AV	Interaction	Usage	Detection	Characteristics
MultiStage	3	4 vs 4	Publish & Subscribe	Video	Yes	User interaction	nearly 360°, 4 cameras/site	Masking the effects of delays on video, Amplified interaction, State monitoring
DIP	2	1 vs 1	Client & Server	Both	Yes	User Interaction	not 360°, 2 cameras/site	Interactive and collaborative environment, Storage and playback of streams, Masking the effects of delays on audio
Telegnosis	3	1 vs N	P2P	Both	No	Monitoring	360°, 2cameras/site	Stream switching function
Three's Company	3	1 vs 1 vs 1	N/A	Both	Yes	Interaction, collaboration	not 360°, 3 cameras/site	Distributed Collaboration
Geminoid HI-1	1	1 vs 2	N/A	Both, robot	Yes	Telecommunication	not 360°, 5 cameras/site	Human telepresence
Mingsong Dou et al. [21]	2	N/A	N/A	Video	Yes	Telepresence, Interaction	360°, 8 cameras/site	Maintain eye contact during conversation
Grimage	2	N/A	N/A	Video	Yes	Virtual immersive, collaborative	8 cameras/site	3D Virtual presence
Slim Essid et al. [23]	2	total 26	N/A	Both	Yes	Dance	4-5 cameras/site	3D real-time realistic interaction in virtual environment
3D helping hands	2	1 vs 1	N/A	Video	Yes	Collaboration	1 camera/site	Mixed Reality system
LiveMask	1	total 8	N/A	Human face	Yes	Nonverbal communication	1 camera	Head gestures
TeleHuman	2	1 vs 1	N/A	Video	Yes	Telepresence	360°, 10 cameras	360°motion parallax, stereoscopic life-sized 3D images of users

Table 2.1: The comparison of MultiStage and other systems.

To achieve different degrees of temporal causal synchrony we use existing approaches as well as through explicit information given to users and through scheduling of threads and processes so that we can do on-the-fly splicing of prerecorded videos into the live video streams, and on-the-fly computations creating content substituting for delayed content coming from remote stages. We also have a distinct interface that allows administrators to manage the system, and humans to inject commands into the system through gestures. Finally, we amplify a user's actions remotely so that they become more visible at remote stages and to an audience.

2.9 Discussion

The camera rig (Figure 2.5) we used in MultiStage system supports an almost 360-degree detection angle. And each stage has three computers and four cameras. Each of the two computers was connected by two cameras. And the third computer was responsible for display the remote presences of actors on display. This setup makes the stages very portable, it is very easy to bring and setup the systems at different places.

The subsystems implementing the local side execute on computers local to a stage. This is done to achieve low local latencies, and reduce the use of network bandwidth. It also distributes the global workload, and isolates the stages such that if one stage fails, the other stages have a higher probability of not being affected. The subsystems implementing the global side execute on computers that are located near the stages to achieve high bandwidth and low latencies. The Administrator Interaction system is located on a computer, which is convenient to use by a director.

MultiStage is a distributed system, and the computers can have different clock values. MultiStage uses NTP to synchronize each computer's clock. The MultiStage system also has several techniques to ensure all computers have the same view of the clock. See Chapter 6.6 and 7 for discussion about NTP and clock synchronization.

UDP protocol is used to transmit data streams between stages. UDP do not guarantee safety delivery of messages to the other end. But it does not have packets retransmission like TCP. The reason we use UDP to send data streams is that each computer sends data streams about 30 FPS. We can tolerate several frames loss. And we want the data streams arrive as fast as possible to achieve fastest transmission latency. The data resend technique TCP has will increase the network transfer latency if we experience packet loss.

TCP protocol is used to send subscription message between DSDS and other stage computers. The reason is that TCP always guarantee the data will be sent to the other end. In this case, we want to make sure that the subscription message is sent to DSDS.

In the configuration, all stage computers just need to know global side computers'

network location without the need to know other stage computers' network location. The stage computers just need to send data streams and subscription messages to global side (data streams are first sent to GSM and subscription messages are sent to DSDS). We also do not need to hard code the network location of stage computers to global side DSDS. When the subscription messages sent from stage computers to DSDS, DSDS will store the network location of this computer and handle the subscriptions. When data streams arrive, DSDS will deliver the data streams according to the subscriptions.

MultiStage will also in the future enable an actor to give gesture input to control a remote physical presence, such as a robot, and manipulating how the actor is displayed. To control a robot, we only need to send few control commands. It will save network bandwidth compare with send data streams over network. However, we need further user study on whether users prefer to interact with the remote presence of actual people or a robot.

Chapter 3

State Monitoring and Analysis

This part is mainly researched by my colleague Giacomo Tartari. I will give a brief description. Because it is a part of a larger whole, and it is needed to see my research in context.

3.1 Local State Monitoring

3.1.1 Motivation

In order to create the remote presence of remote actors on local stage and to amplify their interactions, the state of a stage is needed at other stages. What happening locally on a single stage is monitored by LSM. The monitored information can be used to analyze different gestures from local actors.

3.1.2 Idea

Every stage has a set of sensors, such as cameras, recording local activities. The monitored data streams represent the state of each local stage including physical objects such as walls, furniture, humans, and events such as movements and audio.

There is a one to one mapping between each actor and each camera. The LSM system produces an individual data stream for each actor. Each local data stream represents unique information about one unique actor.

Actor Local Sensing: Using the above principle, the system assumes only a single actor within the field of view for each camera.

In LSM functionality, some extra information will be added to the state, such as timestamp of when each state was captured, in which stage and which camera the state was generated. This extra information will help MultiStage system determine where to

display the state, and help to mask the effects of delays. Each individual data stream can be mapped onto any display in any stage, making for a very flexible combination of background, objects, and users. There could be something else than to use a display to show data streams. For example each individual actor can be a robot, if actor moves his arm, so does the robot.

3.1.3 Architecture

Figure 3.1 shows the connection between LSM and LSA. The LSM includes functionalities to record local events and to identify state that represent the recorded events for further analysis and playback. The produced state will be further analyzed by LSA and sent to GSM.

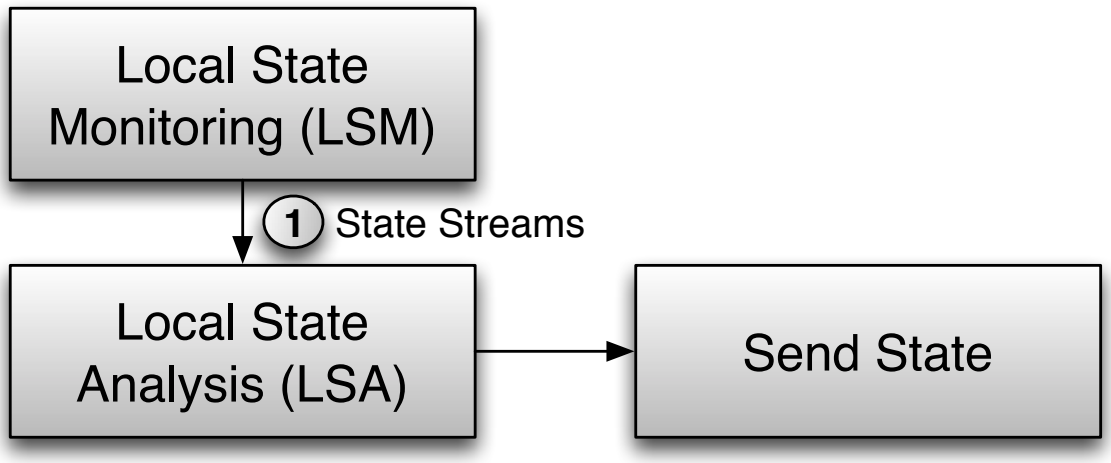


Figure 3.1: The connection between LSM and LSA.

The LSM and LSA have three functionalities. One functionality detects local state and generates state streams. The state streams will be available for LSA. One functionality analyzes the state streams to find out gestures. The analysis is focusing on what happening locally on a single stage. One functionality forwards the state streams to GSM for further analysis. The GSM will focus on find out gestures performed by actors on different stages.

3.1.4 Design and Implementation

Figure 3.2 shows the design and implementation of the LSM and LSA. LSM and LSA are surrounded by a red rectangle. Each stage has a 360-degree multiple-camera rig. The system extracts a 3D recording of actor from each camera, and sends data streams into

the system for distribution between the stages. The data streams are analyzed on-the-fly for information such as gestures.

The detection side has two computers, each with two cameras. Each computer has two processes. One process running LSM and LSA functions, push data streams to global side. One process running the System State and Performance Monitoring process. The communication protocol for push data streams to global side is UDP. A heartbeat function keeps telling the System State and Performance Monitoring process that the LSM and LSA are running.

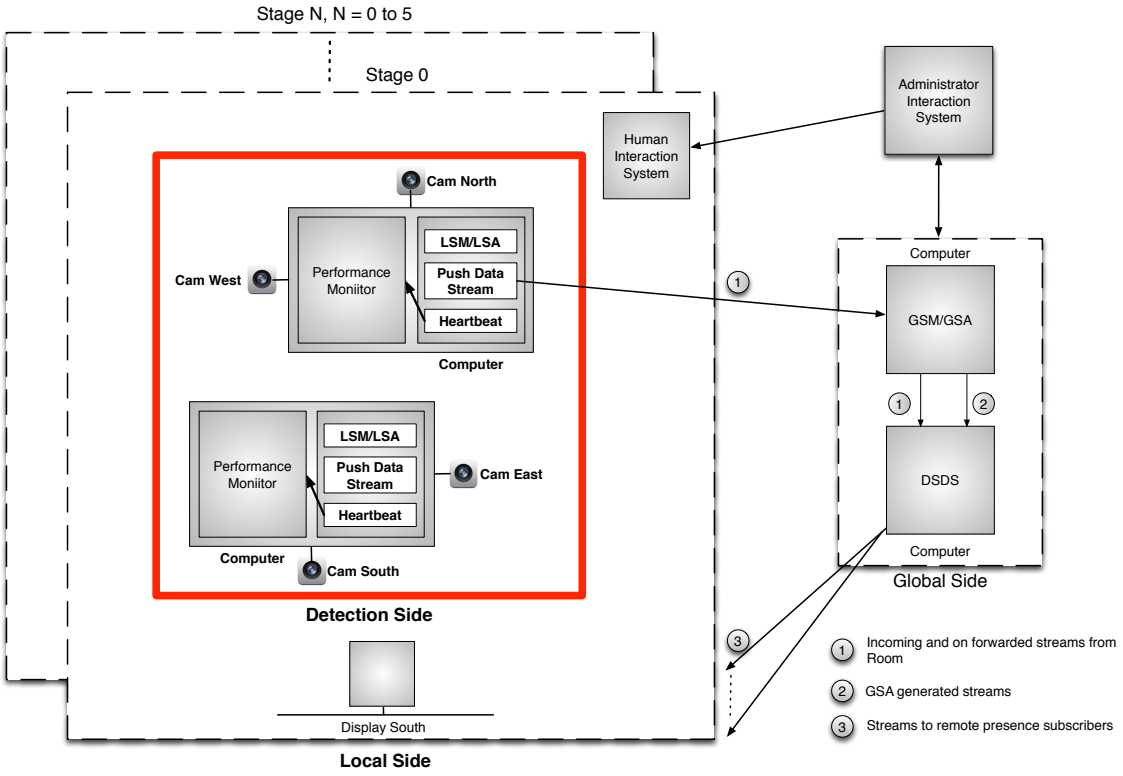


Figure 3.2: The design and implementation of LSM and LSA.

In the present prototype, an individual data stream for each actor is produced by using a Kinect camera per actor. The camera uses RGB and depth images to create actors in a 3D point cloud or 2D RGB images. The system assumes that just a single actor is within the 3D field of view for each Kinect camera. All objects outside of this 3D space are ignored. The implementation uses a one to one mapping between each actor and each camera. Each individual data stream sends from local side contain information about each individual actor. Presently the prototype supports four actors per stage using four Kinect cameras arranged back to back. This allows for a nearly 360-degree detection angle. The user must be inside the reach of the cameras, and about 1-2 meters away.

Each data stream has an ID identifying the originating stage, camera, the type of stream (2D RGB video, 3D point cloud or gesture), and it also includes a sequence number

and a timestamp for each frame. Other stages can make subscriptions to different data streams at DSDS by using this ID. The timestamp indicates the time of when the data stream was captured and it will be used by the Controllable Temporal Synchronization system (Collaboration system) to mask the effects of delays. OpenKinect libfreenect library [27] is used to generate the 3D point cloud and 2D images.

3.2 Local State Analysis

3.2.1 Motivation

Gestures performed by actors locally on a single stage are analyzed by LSA. Actors may want to perform some specific actions to show specific meanings to other actors. The computer must do specific rendering for these actions. For example, when an actor on a local stage raises his right arm, a special image can be displayed. In order to detect these actions, the local state recorded by LSM needs to be analyzed, and then to determine whether there are gestures contained in the recorded data streams. Gestures are to be used as input to processing devices at the stages.

3.2.2 Idea

The idea is that LSA does on-the-fly local analysis to find interesting objects and events in the state detected by LSM. The data is then streamed to the global side for possibly further monitoring and analysis, and for distribution to the stages. The system supporting doing gestures. But we did not investigate on gestures are liked by actors or audiences.

3.2.3 Architecture

LSA does on-the-fly analysis to detect interesting states, such as single and multiple human gestures. The data is then passed to GSM, and global behaviors will be analyzed at GSA.

3.2.4 Design and Implementation

Figure 3.2 shows the design and implementation of the LSM and LSA. LSM and LSA are surrounded by red rectangle.

For 3D point cloud, the system detects specific movement and if this matches a predefined gesture, a specific rendering can be made on the display side. For 2D RGB image, it has a motion detection function to detect actor's movements. If this function is not used, there is no analysis about the image. The raw video image will be sent to distribution.

If the motion detection applies, when an actor does an action, a command will be sent to the display side and the remote presence of this actor will be modified accordingly. Python OpenCV library is used to do the motion detection. After LSM fetches the RGB image, it will compare the image with previous image. If something has moved, it will sends a control commands to the global distribution side. In the current implementation, the control command will control the hand movements of a human skeleton displayed on the remote presence.

3.3 Global State Monitoring

3.3.1 Motivation

The MultiStage system contains several local side stages located at different place. Actors on different stages may need to collaborate together to perform collective gestures. To find out global gestures performed by actors located on different stages, all data streams from different local stages are collected together at GSM for further analysis. GSM then pass the data streams to GSA to analysis global behaviors.

3.3.2 Idea

LSM focuses on what happened locally on a single stage. GSM collecting state from all the stages, and making it available for GSA to do on-the-fly analyze of global behaviors.

3.3.3 Architecture

Figure 3.3 shows the connection between GSM and GSA.

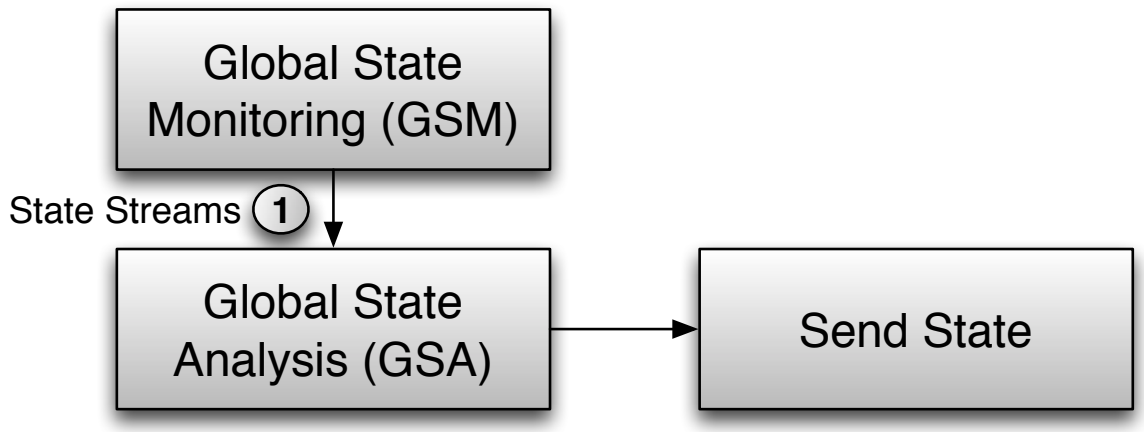


Figure 3.3: The connection between GSM and GSA.

The GSM and GSA have three functions. One function collects all data streams from local side. One function analyzes all data streams and finds out global gestures. New data stream represents the global gestures will be added. One function forwards all data streams to DSDS.

3.3.4 Design and Implementation

Figure 3.4 shows the design and implementation of the GSM and GSA. Only the architecture and design are done for GSM and GSA. The implementation for the two parts is not done yet. GSM and GSA are surrounded by red rectangle. The MultiStage system has several local sides and a centralized global side.

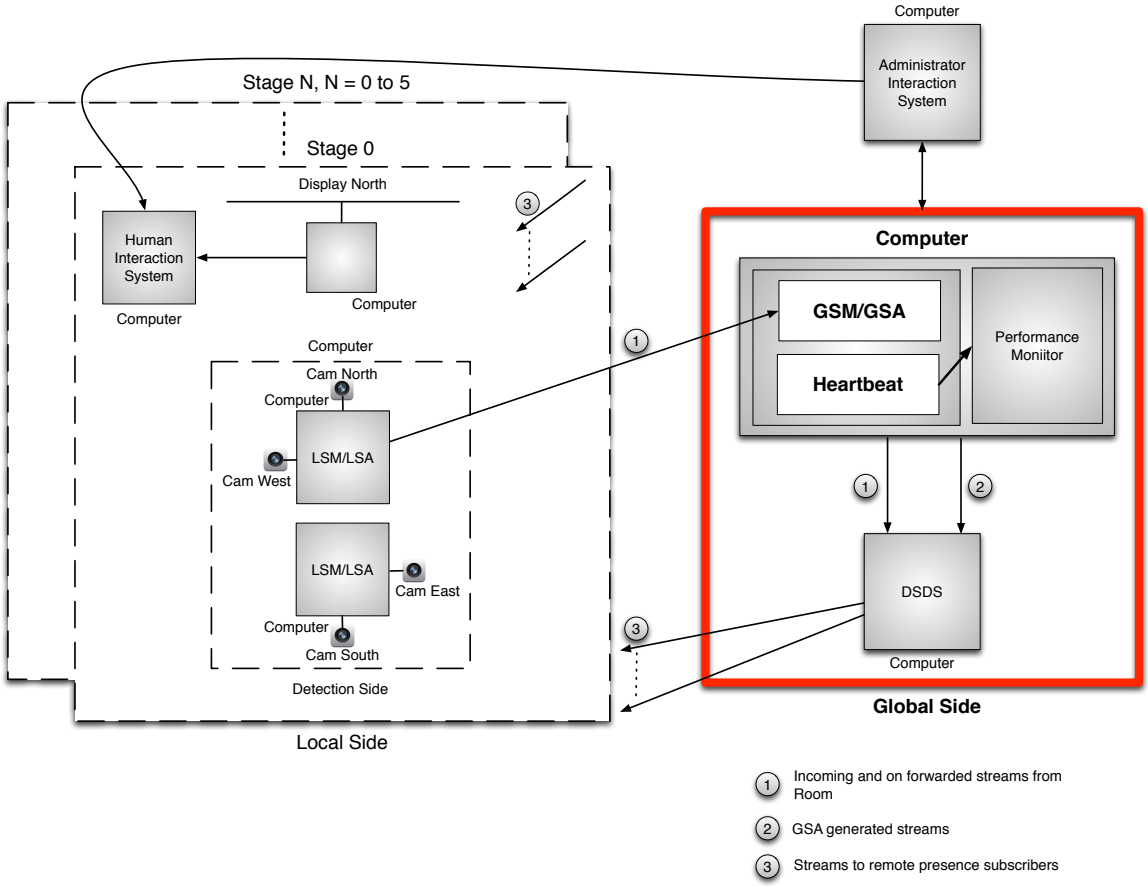


Figure 3.4: The design and implementation of GSM and GSA.

In the design, the computer running GSM and GSA has two processes. One process running GSM and GSA functions, push data streams to DSDS. One process running the System State and Performance Monitoring process. A heartbeat function keeps telling the System State and Performance Monitoring process that the GSM and GSA are running. All local side computers know the IP address of GSM. Data streams can therefore be

forwarded from local side to global side GSM. The communication protocol is UDP.

3.4 Global State Analysis

3.4.1 Motivation

The LSA have data about what happened locally on a single stage. Special actions performed together by actors on different stages are called global gestures. Compare to LSM and LSA, the two functionalities just detect and analyze what has happened locally on a single stage. What if actors on different stages perform actions and gestures together? GSA analyzes data streams from all stages collected by GSM and determines global gestures.

An interesting global state is a collective gesture. It is comprised of several gestures done by several actors possibly on different stages. The idea is that when a given number of actors have done certain gestures, this should result in actions being taken on the stages, such as, for example, turning on a light, or doing some modifications to the remote presences.

3.4.2 Idea

The GSA receives data streams from GSM and does on-the-fly analysis to detect the distributed state, such as collective gestures and collisions when actors on different physical stages occupy the same volume on the virtual stage.

The GSA does analytics on the data streaming in from the stages, looking for global state.

3.4.3 Architecture

GSA includes functions to analyze the state from all stages on-the-fly to detect distributed state, such as collaborative gestures comprised of gestures from multiple stages. Several new state streams are produced representing the detected global state.

All the state streams will be forwarded to DSDS. DSDS provides state about stages to the stages. The remote presence of each actor, and their actions will be amplified and displayed on the local stages based on the received state.

3.4.4 Design and Implementation

Figure 3.4 shows the design and implementation of the GSM and GSA. GSM and GSA are surrounded by red rectangle.

GSA discovers global collaboration from actors on different stages. Global gestures are detected by analysis of multiple input data streams from GSM. It will only analyze the 3D point cloud in the current design. The GSA system also forwards all data and information about global gestures to DSDS.

Chapter 4

Distribution of State Data Streams (DSDS)

4.1 Motivation

In MultiStage, the distributed stages generate data streams, and need data streams from other stages. The purpose of DSDS is to bind together all the stages and to manage the distribution of data streams to the stages. All stages send data streams to DSDS and subscribe data streams from DSDS. Each stage can either receives data streams about individual stage or receives data streams from all stages.

4.2 Idea

DSDS uses publish-subscribe model. Each stage is a producer that publishes data streams to DSDS. Each stage is also a consumer subscribes to data streams from DSDS. To send the subscription messages, all stages just need to know the location of DSDS. DSDS knows the locations of all stages when it receives the subscriptions from local stages.

The MultiStage system has several local sides and a global side. DSDS is a centralized server in the global side. DSDS manages incoming and outgoing data streams for every stage. Each stage acts both as a producer and a consumer. Data streams are published to and subscribed from DSDS. The problem will be even more complicated when more stages are involved. The most extreme case is all stages ask for all data streams.

4.3 Architecture

Figure 4.1 shows the architecture of DSDS.

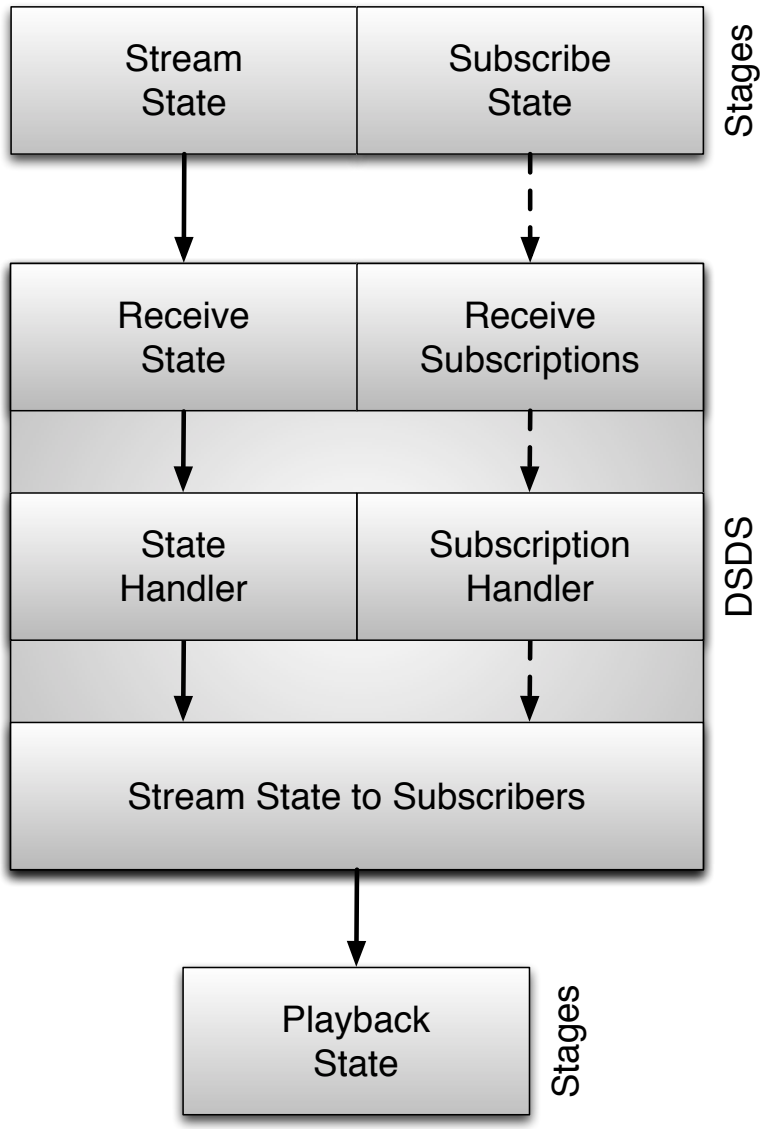


Figure 4.1: The architecture of DSDS.

MultiStage uses a publish-subscribe model to handle the streaming of state. DSDS has functions to receive state from all stages, handling subscriptions from all stages, and stream state to stages according to the subscriptions.

4.4 Design

Figure 4.2 shows the design and implementation of DSDS. The MultiStage System has several local sides and a global side. In the current prototype, DSDS is designed as a centralized server on the global side.

The local side pushes data streams to, subscribes to data streams from DSDS and receives subscribed data streams from DSDS.

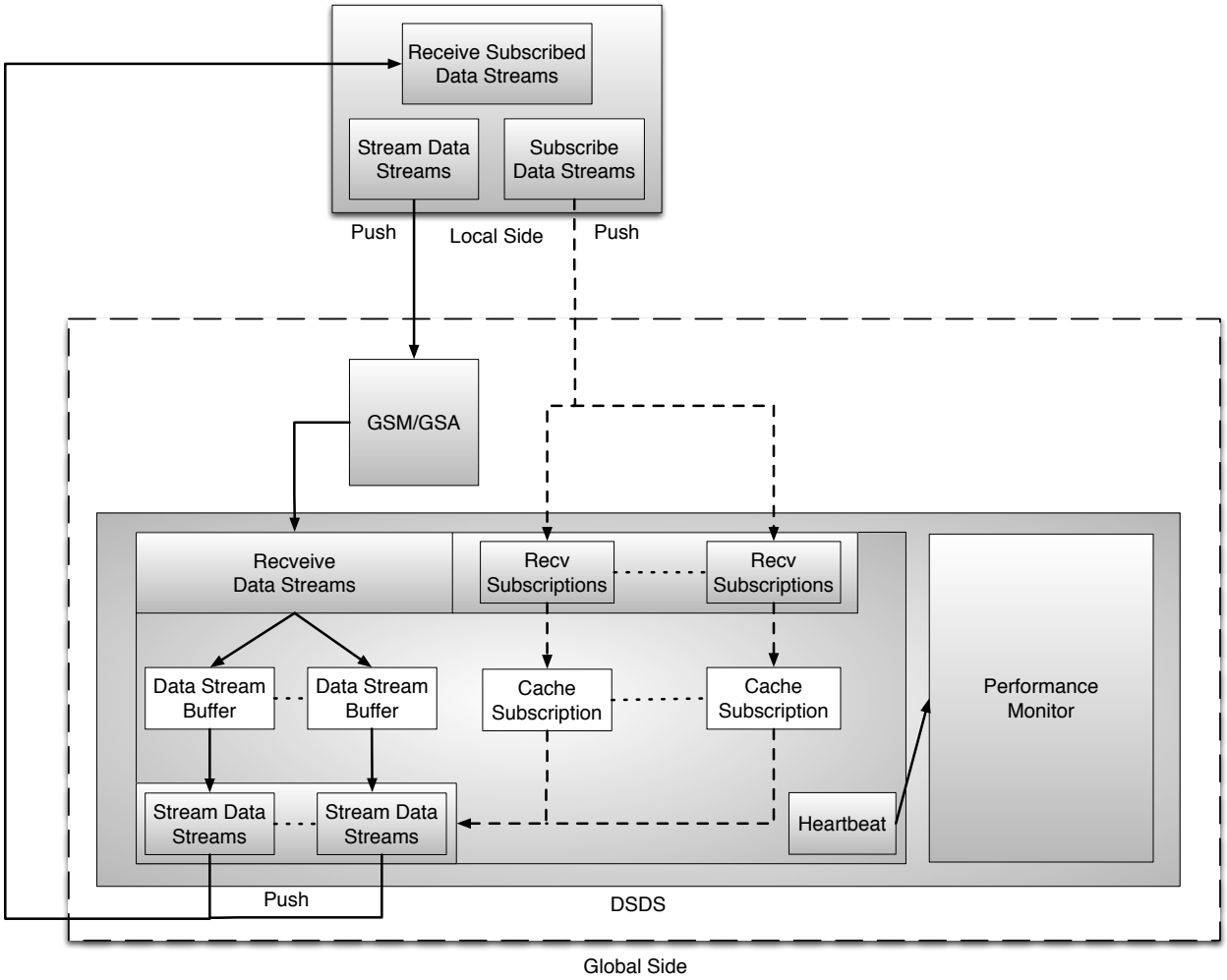


Figure 4.2: The design and implementation of DSDS.

DSDS receives data streams and subscription messages from local sides and pushes data streams back to local side according to local side subscriptions.

Each subscription message contains a unique stream ID to subscribe to a unique data stream. Figure 4.3 shows the data structure of state data packets sent across different stages. Each data packet includes a four-byte ID: information of the packet + stage ID + sensor ID + type of stream. The information in the packet is used to distinguish different data packet. The data packet can be data stream or a subscription message. A subscription to a data stream contains a stage ID and a sensor ID tells DSDS which data stream the subscriber wants. The remote presence computer can use this information to subscribe to different data streams from DSDS. The type of data stream can be video images, 3D point cloud, and commands (used to control the arm movement of a human model) when LSA enables the motion detection. The Counter stores the sequence number for each image. It can be used in the future to check if the images arrived in the right sequence. The timestamp stores when the image is captured at GSM. This

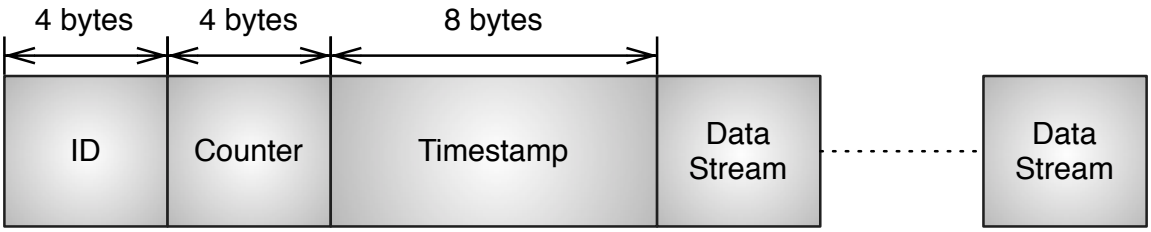


Figure 4.3: The structure of state data packets.

information can be used in the Controllable Temporal Synchronization - Collaboration system (Chapter 6) to determine if the techniques to mask the effects of delays will be applied.

For DSDS, TCP protocol is used to receive the subscription messages from local side computers. Local side computers can subscribe to an individual stream or to all available streams from DSDS. UDP protocol is used to receive data streams from local side and send data streams back to local side according to subscriptions. Memory buffers are used to store the local side computers' IP addresses and store different data streams. These data streams will be sent back to local side based on the local side subscriptions.

4.5 Implementation

In figure 4.2, the DSDS computer has two processes. One process running the System State and Performance Monitoring system. One process running DSDS functions to distribute data streams. It includes a thread receives all data streams from GSM and GSA. Each data stream will be stored in an individual buffer. In the current implementation, Kinect cameras are the sensors used to generate data streams. Several threads receive subscription messages from local side. Each thread manages subscriptions for one client subscriber. Each thread caches the subscriber's IP addresses into a queue. The communication protocol is TCP. Several threads forward data streams to local side based on local side subscriptions. Each thread sends data streams about one specific actor to local side subscribers. It fetches the IP addresses from the cached queue. The communication protocol is UDP. A Heartbeat thread keeps telling the System State and Performance Monitoring system if the process is running or not.

4.6 Discussion

Figure 4.4 shows different designs for DSDS. It can be implemented as a centralized server or be distributed across local sides. The current design and implementation uses a centralized DSDS, local stages communicate with one global DSDS service. Everything

will be managed at centralized DSDS. To send subscription messages to global side, the subscriber just need to know the network location of the centralized DSDS. But this design contains a single point of failure. If DSDS is down, everything will be down.

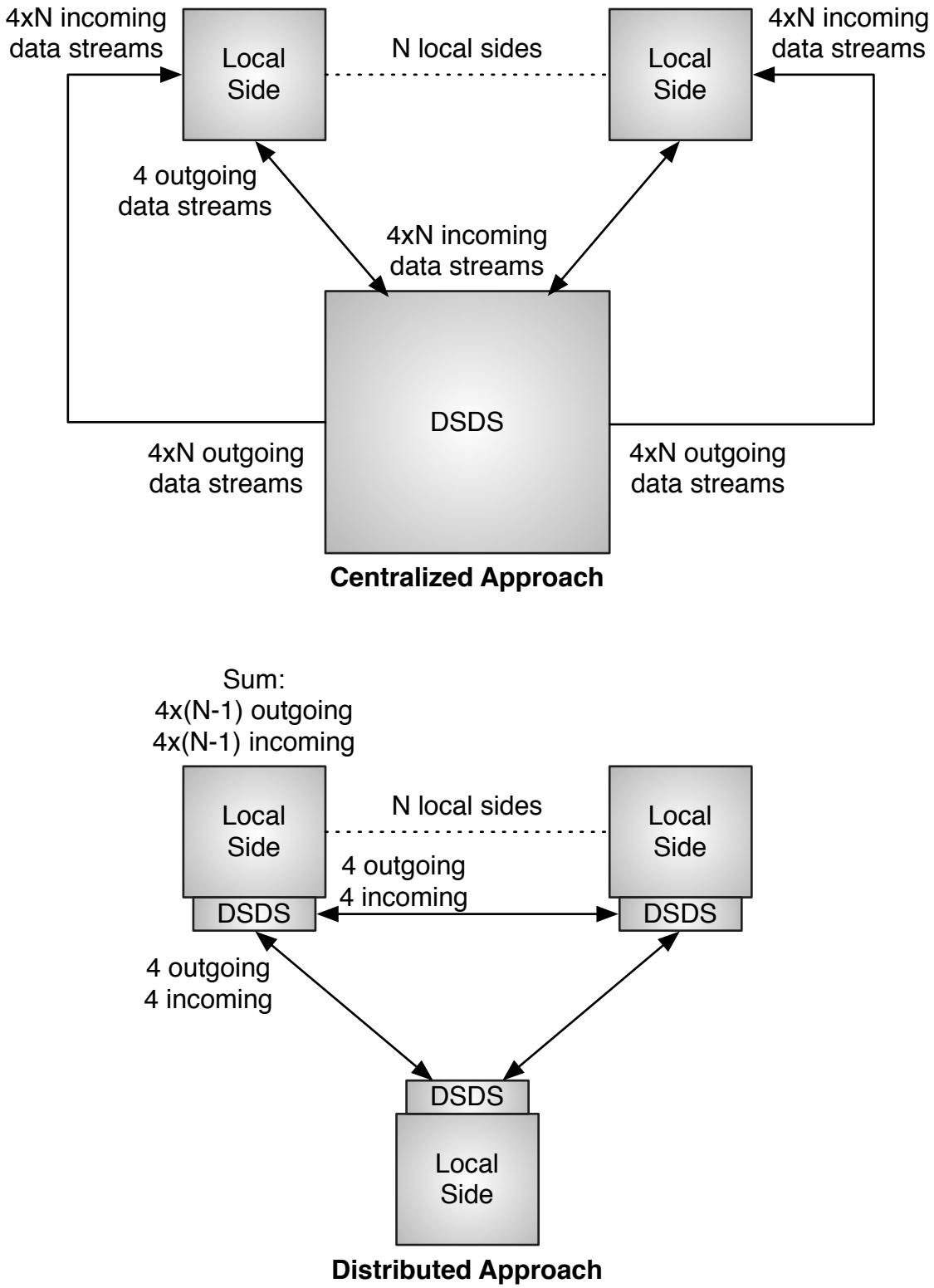


Figure 4.4: Different design for DSDS.

In a distributed approach, DSDS will be placed at every stage. DSDS at every local side stage will communicate with other DSDS at other stages. It also avoids single point of failure, if one DSDS is down, it won't affect other stages. In the centralized approach, all local sides just need to know the IP address of one DSDS. But in distributed approach, DSDS will be distributed into every local side. We need to find a way to let DSDS able to know where the other DSDSs are located.

Figure 4.4 shows the outgoing and incoming data streams at DSDS and local sides. There are four Kinect cameras on each local side, each camera generates one data stream. In the extreme case, each local side will generate four outgoing data streams. The N local sides indicate there are N local side stages. We assume all local sides subscribe to all data streams from all stages. In the centralized approach, DSDS will have 4xN incoming data streams and 4xNxN outgoing data streams. Each local side will have 4 outgoing data streams and 4xN incoming data streams. In the distributed approach, each local side DSDS will have 4x(N-1) outgoing data streams to and 4x(N-1) incoming data streams from Internet. The reason is DSDS do not need to send data streams back to itself. The above discussion did not consider data streams represent local or global gestures generated by LSA and GSA. There can be more data streams if gestures are detected.

DSDS uses TCP protocol to manage the subscription for messages. Because TCP has a reliable data transmission, message can be retransmitted if the previous message is not received. UDP protocol is used for transfer data streams. Because the MultiStage system requires fast data transfer to achieve low latency. And the TCP message retransmission technique will add delay. Although UDP do not has message retransmission technique. But in MultiStage system, each Kinect camera capture 30 images per second, so it is ok to loss few packets.

In the current design and implementation, to subscribe to a data stream, local side must include stage ID and sensor ID into the subscription message. It is cumbersome to subscribe many data streams and the subscribers need to know about how a remote stage is rigged. In the design and implementation, if the subscriber gives 255 and 255 as the stage ID and sensor ID in the subscription message, it means the subscriber want data streams from all sensors and from all local side stages. In short, subscriber wants all data streams. In the future, more protocols can be developed. For example, subscriber can subscribe to all data streams from one local side.

Chapter 5

Remote Presence

This part is mainly researched by my colleague Giacomo Tartari, I will give a brief description. I developed the function to create human skeleton.

5.1 Motivation

The purpose of the Remote Presence is to create remote representation of a physical actor and allows actors on each stage interact with actors on other remote stages. This can be done by create remote presences of all actors. Actors on one stage can interact with the remote presence of other actors. Audience can see both local and remote actors' remote presences. Special rendering for detected gestures are done in remote presence to make the audience and other actors better understand the meanings of actors' actions.

5.2 Idea

A display is used to create remote presence for actor. Remote presence computer fetches data streams from DSDS. Each data stream is captured by one Kinect camera on one stage. Each data stream has a unique mapping to one actor. This gives the receiving side a great flexibility on where to locate the remote presence of actor created from data streams. This also gives the receiving side a great reliability on making decisions on what to do with the data streams (For example, able use pre-recorded data streams when Act-By-Wire approach is enabled). Multiple remote presences will be displayed if the remote presence receives multiple data streams. To avoid overlapping, the remote presence of an actor will be located at different place on the display.

The streams of state data from remote stages comprise information that is used to create remote presence of actors by use various techniques. The techniques include playing data streams. The data streams can be 2D video showing an actor. 3D point could uses

point to recreate an actor on the display. And possibly having a local physical effect through actuators and robots, if an actor moves his arm, so does the robots. The remote presence can also create animations of remote events and reacting to remote gestures.

5.3 Architecture

The functionalities of the Remote Presence system include:

1. Sending subscription messages to DSDS. It can subscribe several states or all states from DSDS.

2. Receiving states from Collaboration system (H - Controllable Temporal Synchronization in figure 2.2). The state received from Collaboration System can be: (a) The state received from DSDS. (b) The pre-recorded state. If the original state was delayed too much, specific approach to mask the effects of delays will be enabled.

3. Creating the remote presence of actors on the display. The remote presence of actor will be displayed in the correct location on the display according to the stage and camera ID. This allows (a) All stages have the same layout of remote presences. (b) Can also be stage specific.

5.4 Design

Figure 5.1 shows the design and implementation of the MultiStage system. The Remote Presence is surrounded by a red rectangle.

1. The Remote Presence system subscribes to data streams using unique ID contains stage and camera ID. Each data stream contains a unique ID that is described at figure 4.3 from Chapter 4. If number 255 is used as stage and camera ID, it means the Remote Presence system subscribes to all data streams from DSDS.

2. The Remote Presence receives data streams from Collaboration system. If masking the effects of delays approach is enabled on Controllable Temporal Synchronization - Collaboration system, the Remote Presence system will receive pre-recorded data streams instead of live data streams.

3. The Remote Presence system will create remote presences from the received data streams on the display. Depending on the stage and camera ID include in the

data stream, the Remote Presence system will display the remote presences on different places on the display. Presently, the primary remote presence technique is to visualize remote actors on a very large display per stage. In the future, we may add physical devices such as robots to the remote presence.

There are three different methods used by the prototype to create the remote presence of an actor. It can be 2D streaming videos based on color images captured by four Kinect cameras at each stage. Alternatively, 3D streaming point cloud videos can be used. These are created using color and depth images captured by the Kinect cameras. Finally, if the motion detection function is enabled at LSA, the remote presence can be visualized as an animated human skeleton created locally at each stage. The data streams will contain information about control commands to control the movements of the human skeletons.

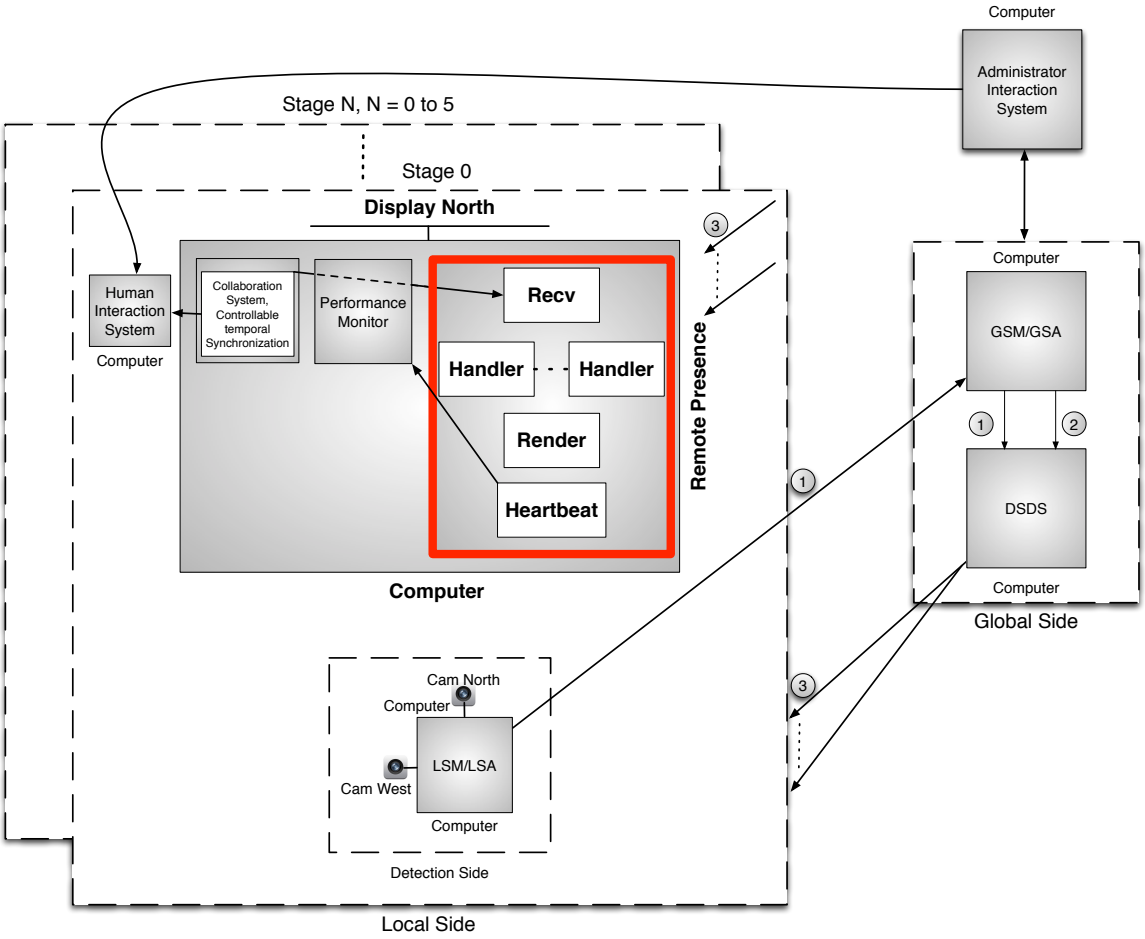


Figure 5.1: The design and implementation of Remote Presence.

5.5 Implementation

The Remote Presence process contains four threads.

1. One thread sends out subscription messages of data streams to DSDS. The protocol for subscriptions is TCP. In the current implementation, the Remote Presence subscribes to all data streams.

2. One thread receives data streams through UNIX socket from Controllable Temporal Synchronization - Collaboration system. If masking the effects of delays approach is enabled on Controllable Temporal Synchronization - Collaboration system, for the 2D streaming videos and 3D streaming point cloud videos, the Collaboration system will send related pre-recorded videos. For motion detection, the Collaboration system will send pre-determined control commands to Remote Presence system.

3. One thread renders received data streams on display. All received data streams will be rendered at the right location depending on the stage and camera ID of that data stream. For 2D videos, the Remote Presence system will display videos about actors on display. Each individual video represents one actor captured from on unique camera from one stage. For 3D point cloud video, remote presence of an actor will be created in the correct location on display. If the motion detection function is enabled on LSA to identify actor's body movements, the data about this makes its way to the Remote Presence system, and the computed human skeleton moves accordingly.

4. A Heartbeat thread keeps telling the System State and Performance Monitoring system if the process is running or not.

OpenKinect libfreenect library is used to create 3D point cloud and 2D images. OpenGL library is used to render point cloud on display. Python OpenCV is used to do motion detection. Python Pygame [28] is used to render the 2D video images and the human skeleton.

In the current implementation, the process of Controllable Temporal Synchronization - Collaboration system and the process of Remote Presence system are running at the same computer. Data streams are passed from Collaboration system to Remote Presence system through UNIX socket.

Chapter 6

Controllable Temporal Synchronization - Collaboration System

6.1 Motivation

In distributed acting, actors on different stages, physically separated by distance, interact to create a coherent play. The interaction can be lazy, allowing for large delays without breaking the illusion of being on the same stage. This is, for example, the situation when actors do a relaxed handshake, or do not interact directly at all. The interaction can also be eager, where even small delays break the illusion. This is, for example, the case when actors do fast action/reaction with causally related movements between each other, or move in synchrony as done in dancing.

Distributed acting is complicated by each stage having multiple clocks (one per computer), and by communication delays and jitter. The clock on each stage can easily be synchronized with a reference clock, but delays and jitter are unavoidable and are the result of the finite speed of light, and of the technologies and systems applied to create a distributed stage gluing together the individual stages.

1 km	3.3 μs	Between buildings
1000 km	3.3 ms	Between cities
40000 km	134 ms	Around equator
2.4 x 10^{19} km	2.5M years	To Andromeda Galaxy

Table 6.1: Travel time at the speed of light

The speed of light defines the lower bound of a non-zero delay from when an event happens until it can be observed. Table 6.1 shows the time needed for light to travel distances that may be typical in distributed acting. It takes about 3 μs between buildings,

3.3 ms between cities, and about 134 ms around earth's equator. The time it takes for light to travel from an actor to another and back is twice this amount of time. However, the delays experienced by actors interacting through a computer-based system are even higher.

Figure 6.1 indicates the total delay between an event happen and the event being visualized on the display. Every processing phase will add some delays. The delays can be significantly larger than what is indicated in the figure if more processing is applied. These delays can be reduced and partially masked, but they can never be removed. In Chapter 8, we measured the accumulated delay in **System end-to-end one-way latency** part and the network latency in **Global-to-local round-trip latency** part.

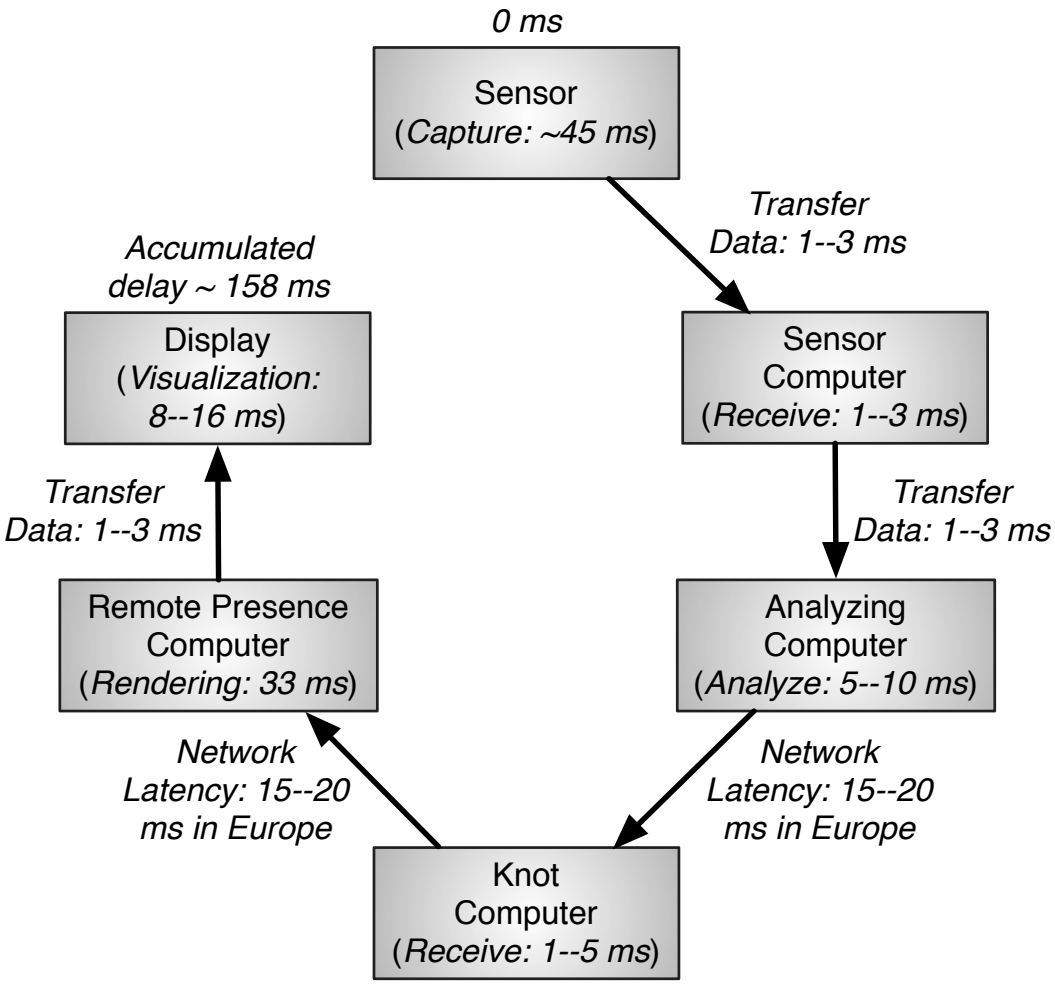


Figure 6.1: Every Phase will add delay

The length of delays is an important consideration when people interact. For tight interaction, it has been documented [29], [30], [31], [32] that people consider delays below 200 ms to be insignificant. When the delays grow beyond 200 ms they become harder and harder to ignore, and actors can be expected to have problems interacting as if they

were on the same physical stage.

The goal of the Controllable Temporal Synchronization system is to provide different approaches to mask the effects of delays on the actors and audiences.

6.2 Masking the Effects of Delays

In Chapter 2 and [2], we define **loose temporal causal synchrony** to be when actions by actors happen causally in the correct order, but with no special demands on delays. **Interactive temporal causal synchrony** is when actions by an actor are seen in causal order and with delays as actors are used to when being on the same stage face to face.

However, for structured interaction with coordinated movements, as in synchronized dancing and in rapid action-reaction situations such as, for example, martial arts, correct causal ordering and short delays become critical to preserve the illusion that the actors are on the same stage.

To achieve temporal casual synchrony between actors even with delays and jitter being unavoidable, the idea is to mask the effects of delays as seen by the actors.

In the **Act-By-Actor**-approach, the actors react to the remote presences as if the latter were the actual actors. How the interaction looks and how it feels to actors and audiences depends on how large the delays are, by how much they vary, and by how good the actors are at compensating. Only loose temporal causal synchrony can be expected to be achieved.

In the **Act-By-Director**-approach, the clocks of all computers are synchronized, same performance start time is set and a countdown is begun at each stage. When the countdown finishes, each actor starts acting according to a script defining what the actor should do and when the actor should do it (for instance, two actors shaking hands). The script is a director keeps time and tells actors when to do actions according to a shared script or to a script for each actor. Even if the actors act on command it will seem to an audience as if they interact freely with each other.

A variant is to select a stage to be the **live** stage. The others are secondary stages. The start time for a performance at a secondary stage is the start time for the live stage minus the delay between them. Consequently, performances at secondary stages are started a little earlier than at the live stage such that when the live stage starts, the input from the secondary stages arrives. At the live stage, the actors and audience will experience a performance where local actors are in synchrony with the remote presences representing the remote actors. However, actors on a secondary stage will be out of sync with the remote presences of actors on other stages. By switching which stage is the live stage at suitable points in the performance, each stage can be the live stage for a time.

A second variant of this approach is to **delay each local remote presence** at a stage. A local remote presence is the remote presence of an actor shown and heard at the stage where the physical actor is. The effect is that an actor and an audience will experience a local and a remote event at the same time because they have both been delayed by an equal amount. To make this approach practical, the delay cannot be so long as to make the actors and audience notice it too much. Because delays between stages in practice tend to be different, this approach is most practical for just two stages with about equal delay between them.

A third variant is to **delay all remote presences** at a stage until data for the slowest remote presence arrives. With varying delays between the stages, they will soon be out of synchronization with each other. However, the local and remote presences at a stage will be synchronized. The delay waiting for the slowest remote presence can be long enough to be noticeable for actors and an audience. Consequently, the actors on a stage can be out of synchronization with the remote presences.

In the **Act-By-Wire**-approach, the clocks are synchronized, and the stages start performance at the same time. Remote presences are manipulated to mask the effects of delays when delays reach predefined threshold values. Manipulations include just-in-time blending in of prerecorded videos of remote presences of actors, and just-in-time blending in of on-demand computed remote presences. A prerecorded and an on-demand computed remote presence would to a varying degree succeed in creating the illusion of short insignificant delays. If there is a script of what an actor should do at a given time, then a prerecorded remote presence can be created and played back at the correct time when delays become too high. When instead of using a static prerecorded video, a computation is run to create the remote presence. A wide range of possibilities are in principle available. These include blurring the movements of an actor such that delays are not so obvious and predicting what an actor is going to do. We have not explored these possibilities yet.

6.3 Related Literature

Several systems try to enable interaction between local and remote users. The DIP [6], [33] and [7], is a multi-site interaction and collaboration system for interactive musical performances. In experiments, the local stage was artificially delayed and it was found out that:

1. The tolerable latency for slow-paced music is much higher than for fast-paced music.

2. To help performers pick up aural cues it is better to have a low audio latency than

synchronizing video and audio.

3. A round-trip video delay of more than 230 ms makes synchronization hard for the users.

In [34], a series of experiments on the DIP system is described with focus on the audio delay, and how the delay affects the cooperation of musicians. An artificial delay of 50 ms to the remote room's audio stream was tolerable. With the same latency added in both rooms it became possible to easily play together with a delay of up to 65 ms. While the authors report on the effects of delays on audio, we report on the effects of delays on videos, and how they can be masked.

Other distributed collaboration systems include [35], [9], and [36]. These do not consider the effects of delays and how to mask them when users interact across distance.

Several techniques [10], [37], [38], [39] and [40], exist to reduce or hide network latency in network games and in distributed systems. The DR technique is used in distributed simulations and to hide latency mostly in network games. For example, in a network car racing game, each car is an entity. Computers that own an entity will send unique information about the entity to other computers on the network. The information includes the position, velocity, and acceleration of the entity or more. Each computer simulates the movement of the entity. The computer, which owns the entity will also simulate the entity as well as check the real state of the entity. When the simulated value and real value differs more than a threshold, the computer will send updated information to the other computers. The DR technique is a general way to decrease the amount of messages communicated among the participants.

IDMaps [41] measures distance information on the Internet. This is used to predict latencies. King [42] uses recursive Domain Name System (DNS) queries to predict latency between arbitrary end hosts. In [43], a structural approach to latency prediction based on the Internet's routing topology is proposed. In [44], the network latency is reduced based on estimates of the network path quality between end points. These approaches can be useful even if we do not mask latencies themselves, but the effects of delays. Predicting the very near future latency can be useful because we can start the masking right before large delays happen. The LL technique [11] provides for better fairness between local and remote players by making all see approximately the same delays. A local operation is delayed for a short time. During this short time period the operation is transmitted to remote computers participating in the game, and all computers can then execute the operation closer in time to each other. However, with more than two participants seeing significantly different latencies, the fairness cannot be maintained for all computers. In [45] and [46], the LL is integrated with DR to synchronize participants

and keep better consistency among all computers.

In [38] and [45], some of the drawbacks of the previously mentioned DR and LL techniques are identified. While the LL technique ensures fairness for two players, or for multiple players with the same latencies between them, the fairness is not preserved when the latencies become too different. The same is the case for the DR approach because when a computer does an update, the time it takes to have data about this delivered at the other computers will vary depending on the latencies between the local computer and each of the other computers. This can result in a situation where a local player and some of the remote players can do actions earlier than other remote players.

Even if it is worthwhile to reduce network latencies and other delays, and do overlapping between communications and processing, delays cannot be removed. In this dissertation, we present several approaches to mask the effects of delays, and we also measure the cost of applying each approach.

There are several projects which have studied the effects of latency when remote users interact, including [47], [48], [30], [49], [32], [31], and [50]. When the latency from when a user does an action until it is reflected in, for example, a game, is more than 200 ms, the user will notice the delay and his actions and score will be impacted by it. In a first person shooter game there is a 35% drop in shooting accuracy at 100 ms of latency, and the accuracy drops sharply when the latency increases further. More than 200 ms of latency should be avoided. For some sports and role-playing games, a latency of 500 ms can be acceptable. Consequently, latency reduction and hiding techniques should aim at achieving end-to-end latencies less or equal to these numbers. When this cannot be achieved, then masking the effects of the various delays becomes of interest to apply as well.

In [51], a comparison is made between the end-to-end latency of an immersive virtual environment and a videoconferencing system. The tolerable latency for verbal communication was found to be 150 ms. This was achieved by the teleconferencing system, but not the virtual environment system. A video was made capturing a person repeatedly moving an arm up and down. A video was also made of the same person as represented by the system. Synchronized cameras were used to synchronize the two videos. The latency from when the person moved an arm until it was reflected through the system was measured to be 100-120 ms for the teleconferencing system, and 220-260 ms for the virtual environment when the avatar for the user had been preloaded.

In [52], several techniques were used to reduce the latency for the head tracking system of an immersive simulation system. The techniques included disabling buffering and having a more direct path to the tracker hardware. This resulted in an almost 50% reduction in latency, from around 90 ms to around 50 ms.

Packet jitter [53] is the variation in the packet delay. Variations in packet size, buffer delay, and routing create packet jitter. The influence of the jitter in games is measured in [49], [54], [55], and [56]. They conclude that jitter had only a minor impact on the win probability, the scores, and the user experience. However, when jitter increases, the tracking accuracy of a target, the user's ability to keep a small and consistent distance between the center of the target and the cursor, declines.

In [57], the authors consider unfairness created by the cumulated errors between players. The system improved fairness by equalizing for all players, the errors of where an object of the game was placed and what it was doing. This resulted in a significant improvement in consistency between what players observed even for 100 ms of delay between players at different computers.

Paper [58] presents Complex Event Processing (CEP) and Distributed Complex Event Processing (DCEP) mechanism for Internet of Things (IoT). CEP is a method to analyze data, sense the events, and generate response actions for IoT. However, in IoT applications, data will be gathered from different sensors with different frequency. It is inappropriate to collect all data in a centralized server and process it, because of failures and network disconnection can happen in the centralized server. Also because to transfer large amount of raw data requires high network bandwidth, and network quality cannot always be guaranteed. It also gives high computation load on the central server. This paper also presents the use of DCEP engine on a smart building. It expands the CEP mechanism to distributed system. It uses client-server architecture. Data is processed and filtered at the client side first. Only relevant data will be sent to the server for further processing. The system reduces the network traffic and the computing load on the server side by separating the load into client side.

6.4 Architecture

The Controllable Temporal Synchronization - Collaboration System has the following functionalities:

1. Clock synchronization: To let performance start at the same time, all stages must have the same view of the clock. A **shared clock** is assumed by the system.

2. Delays and clock Difference: **System Performance and State Monitoring System** measures and computes the communication delays and clock difference between all computers. The measurement will be used by the Collaboration System to adjust the performance start time.

3. Several approaches to masking the effects of delays: It includes approaches to adjust performance start time, delay local remote presence to wait for the state from remote stages and playback of pre-recorded state.

6.5 Design and Implementation

Figure 6.2 shows the design and implementation of the MultiStage system. Controllable Temporal Synchronization is surrounded by red rectangle.

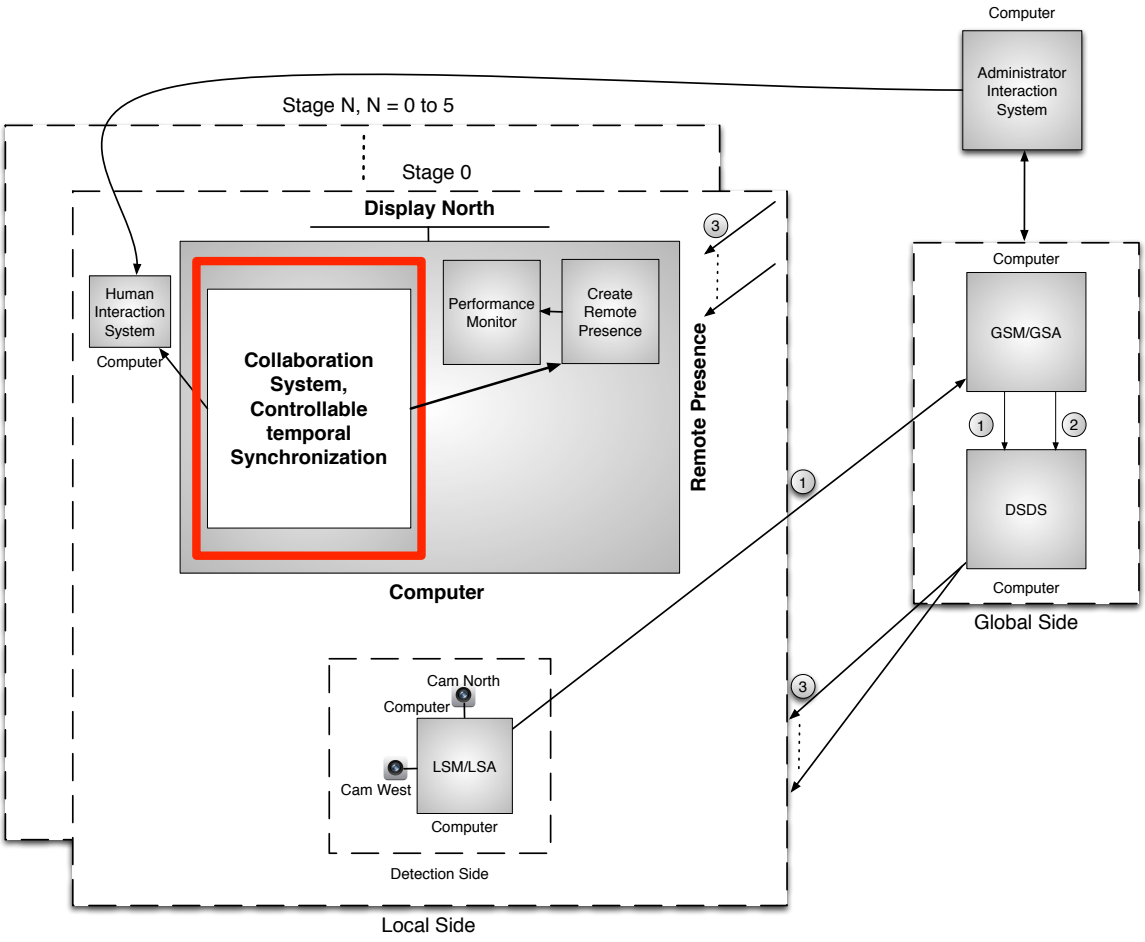


Figure 6.2: The design and implementation of Controllable Temporal Synchronization.

The Controllable Temporal Synchronization - Collaboration system looks for incoming data about remote actors being too delayed to do remote presences without the local actors noticing the delay. If the delays are too large, it will apply several approaches to mask the effects of the delays as seen by the actors. A limited form of masking is also done outside of the Remote Presence system. In this case the Collaboration system gets information collected by the System Performance and State Monitoring System, and use the information to adjust the performance start time.

To do masking, several functionalities must be realized at each stage. The clock is synchronized with sufficient accuracy by using NTP [16] to set the local clocks. The System Performance and State Monitoring System measures the delays and the clock differences between the computers at a stage and the DSDS distribution computer. These information will be used to adjust the performance start time on each stage. Detailed measurement will be described in the Shared Clock section and System Performance and State Monitoring section in Chapter 7.

A **shared and individual performance start time** is distributed by using the Administrator Interaction system to send a message with the performance start time to each stage. We assume that when needed, there are predefined **actor scripts** available telling each actor what and when to do an action. In the prototype, a display at each stage shows a countdown until when the next action is to be done, and visualizes what the action expected of the actor is with a simple drawing.

The design and implementation of the masking approaches are shown in Figure 6.3. One listener thread listening to remote commands and receives data streams from DSDS. It also receives start play command from the Administrator Interaction system. If the Act-By-Director approach (approach A) is used, it will also adjust the individual start time based on the received data. When the function receives the performance start command, it will invoke a countdown thread. After the current time reaches to the performance start time, the Controllable Temporal Synchronization system will invoke the Human Interaction system. And an actor script will be displayed and actor can start performance according to the actor script. The Human Interaction system and the Controllable Temporal Synchronization system is integrated together in the current implementation.

The listener also stores the received data streams about actors into queues. One thread keep reading data streams from queues and forward the data streams to the Remote Presence system. In the Act-By-Wire approach (approach C and D), for each received data stream, it keeps checking if the data stream arrived late by compare the current time and the timestamp of when this data stream been captured. If a certain percentage of data arrived late in a period, the function will send prerecorded data streams to the Remote Presence system. The communication protocol is UDP. The reason to use UDP protocol is because we want the data being delivered as fast as possible and we can tolerate several data loses during transmission. The TCP message retransmission technique will delay the data transfer.

For all approaches we assume that the stages have already initiated subscriptions to data streams from each other, and that the streaming is in effect.

Live Stage: The Administrator Interaction system uses the performance monitor

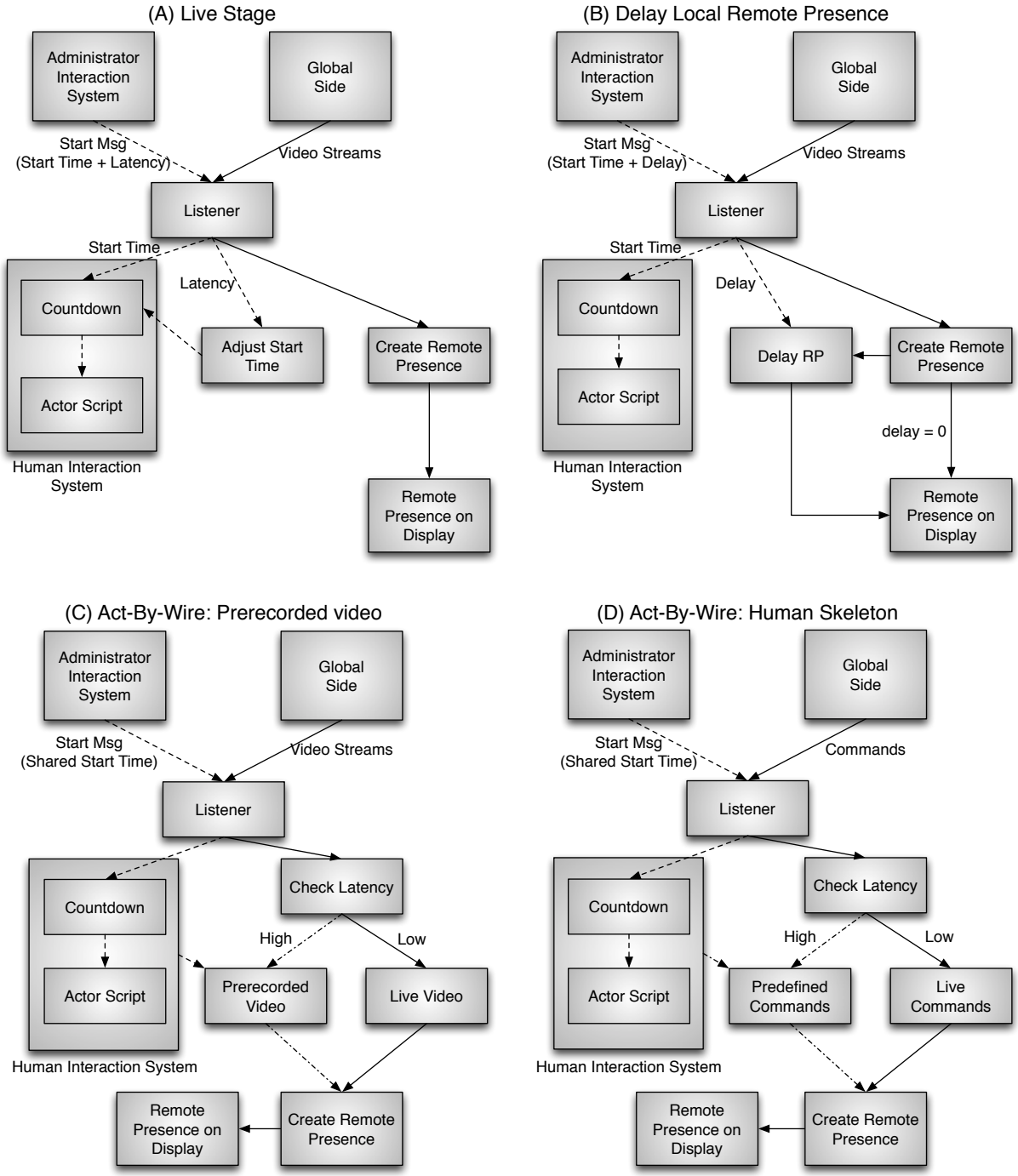


Figure 6.3: Design and Implementation of the approaches to mask the effects of delays: (A) Live Stage, (B) Delay Local Remote Presence, (C) Act-By-Wire: Prerecorded video and (D) Act-By-Wire: Human Skeleton

to measure the latency from the detection computer at each secondary stage to the distribution server (DSDS). It also measures the latency from the distribution server to the remote presence computer at the live stage. The effective latency from a secondary stage to the live stage is the sum of these two latencies. A secondary stage's performance

start time is the start time at the live stage minus the latency between the live and the secondary stage.

The Administrator Interaction system now sends a message to each stage with the start time of the performance and the latency that should be deducted from the start time for that particular stage. The Human Interaction system at each secondary stage will now do a countdown with the start time of the live stage modified by the latency to the live stage. When the countdown ends, a visualization of what each actor should do is displayed. The human interaction system acts as a director, counting down to the next action of each actor, and then visualizing the action.

Delay Local Remote Presences: The Administrator Interaction system uses the performance monitor to measure the delay from the detection computer at each stage to the distribution server. It also measures the delay from the distribution server to the remote presence computer at the stage. If the delays are similar, an average delay is computed, the Delay Local Remote Presence approach can be applied. The Administrator Interaction system sends a message to each stage with the start time of the performance and the average delay between the stages. The Human Interaction system starts a countdown at the given start time. At a stage, each remote presence representing a local actor on the stage is locally delayed by the average delay. The remote presences from other stages are not delayed by the receiving stages.

Locally Delay the remote presences until data for the most delayed remote presence arrives: The Administrator Interaction system uses the performance monitor to measure the delay from the detection computer at each stage to the distribution server. It also measures the delay from the distribution server to the remote presence computer at the stage. The effective delay from the detection side of a stage to the display side of a stage is the sum of these two delays.

The Administrative Interaction system sends a message to each stage with the same start time, and sends the delay from every stage to the stage receiving the message. Each stage calculates by how long time the remote presences from each stage should be delayed to play back close in time to the remote presences coming from the stage with the longest delay. The Human Interaction system starts a countdown, and tells the actors what to do and when to do it. The Remote Presence system creates remote presences as fast as it can, but remote presences from each stage are individually delayed by the calculated amount for each stage.

Act-By-Wire, blend in prerecorded video or compute a remote presence: The Administrator Interaction system sends the same start time to the Human Interaction system at each stage. It starts a countdown and tells the actors what action to do and when to do it. For every image (or video frame) arriving to be used to create a remote

presence, a check is made to see whether the time between the send timestamp of the image and the receive time is more than a threshold to warrant masking. If more than a threshold percentage of images are late, we start masking. If the percentage decreases to a certain value, we stop masking. The threshold values to be used can be found by subjectively trying the system on humans with different delay values, and determining when humans notice the delays in relevant contexts. We typically use a delay of about 280 ms as the threshold for starting masking, see when to start masking and when to stop masking section in Chapter 8.

To mask short-term delays, the system checks for delays over the last few seconds. The exact number of seconds used is tunable, depending upon how sensitive humans in a particular setting are to delayed remote presences.

The video used to mask the effects of delays is prerecorded. The Human Interaction system does a countdown, and tells an actor what to do and when to do it, and a video is recorded. When later the same script is used during a performance, and the delay go above the threshold, the prerecorded video blends in and takes over for the streaming video coming from a remote stage.

The Collaboration system keeps the prerecorded video ready in memory, and when masking is determined to be needed after checking the latency, it streams the prerecorded video to the Remote Presence system instead of the live streaming video.

Alternatively, instead of using a prerecorded video, a model of an actor can be used. Instead of streaming a prerecorded video to the Remote Presence system, the Collaboration system streams the output from an implementation of the model. The model can receive input about detected body movements from the LSA (through the distribution server) of the remote stage. It can also use the script from the Human Interaction system to determine what an actor is meant to do. Presently, just a simple human skeleton model is used with arm movements taken from a script defining what an actor should do. It is future work to explore models and predict actor behavior more fully.

6.6 Discussion

Some of the masking approaches we applied need a synchronization of the clocks at every computer in, and consequently at, every stage of the system. The NTP provides time accuracy in the range of 1-30 ms. The exact accuracy is highly dependent on the location of the computers versus the NTP servers. If the computers are on the same LAN, this will bring them close, around 1 ms, to each other. If they are separated by the Internet, the clocks can be synchronized to within tens of milliseconds of each other. However, network congestion and routing can cause the clock value used by each computer to be

off by hundreds of milliseconds. Therefore, we do frequent NTP-based clock settings and check explicitly for the clock difference between the computers to see whether the clocks are more than 10 ms off. If they are, we repeat using NTP to try to get all clocks within 10 ms of each other. To further ensure that the clocks are close enough, before the performance start time is sent to each stage, we again check the clock difference between the computer distributing data to all stages and the remote presence computers at every stage. The clock difference relevant for a stage is included in the message sent to each stage. A stage can then correct its performance start time accordingly if needed.

Different approaches to masking the effects of delays should be expected to be needed based on what the actors are doing. When the actors do slow movements and the delays are small, the Act-By-Actor approach can be sufficient. However, it cannot mask the effects of larger delays. The Act-By-Director approach tells actors what to do and when to do an action. All actors are as such seen by an audience at a stage to be synchronized. But the actors are following the director script and they are not interacting with each other. This approach can mask the effects of large delays. The live stage approach will make just a single stage look synchronized. The other secondary stages will typically be out of synchronization with the live stage and each other. The approach of delaying the local remote presences by the amount of the delay to the remote stages will make all stages synchronized if the artificially added delay is smaller than 65 ms [34] for audio and 300-400 ms [14] for video.

The approach of letting each stage do local delays of every remote presence and waiting for the most delayed will make each stage synchronized (the remote presences act together), but the stages will not be inter-stage synchronized (the real actors on the stage will be out of sync with the remote presences). The Act-By-Wire approach can synchronize actors and the remote presence of actors on all stages. However, it makes use of prerecorded videos and creates on-the-fly remote presences. These can be quite different from, for example, a video of the real actors.

All the masking approaches were tried in the prototype system. However, they are primarily documented as principles. To evaluate where they fit best in an interaction, they should be used with real acting in formal user studies.

The most advanced masking approach, Act-By-Wire approach using a model of the human to create the remote presence, can be applied with much more complex models than a human skeleton. This is for future research. However, when a computable model of an actor is used, its execution should ideally produce results fast enough to not create further delays. If the model demands too long a running time to create the needed output, a simpler model may have to be used. Alternatively, predictive techniques may be needed to have output ready when it is needed. The predictions can be based on pre-written

scripts defining what a human is meant to be doing at any given time, or it can be based on analyzing the human's actions in the recent past. Predicting the behavior of an actor in the MultiStage system is future research.

6.7 Conclusion

In computer supported human-to-human interaction across distance, delays cannot be avoided. Consequently, while reducing the delays is well worth doing, sometimes they still become too large to ignore for humans. When this is the case, some of the effects of delays can be masked to create an illusion for the humans interacting, and for observers, that they are in the same room or on the same stage. However, the illusion created by masking has several limitations depending on which masking approach is used. There are principally two different types of masking. One type coordinates the interaction at suitable times to create a better illusion. The other frequently monitors the delays, and substitutes delayed data with data already available at each stage. Depending on the type of interaction, a suitable masking approach should be selected. The most complex approach, Act-By-Wire, will in all situations in principle create an illusion where interacting humans are fooled into believing that there are no significant delays perturbing the interaction. However, this approach can also create unexpected representations of remote humans, and when this happens it becomes clear that what is shown is only an approximation of the remote reality.

Chapter 7

Miscellaneous MultiStage Subsystems

7.1 Shared Clock

7.1.1 Motivation

The MultiStage is a distributed system. Each computer can have different view of the clock. The difference in the time between computers may have a great impact on when actors do tight interactions. Assuming all stages start performance at the same time, if the clocks at stages differ too much, some stages will start earlier and some will start later. The result is actors on different stages cannot perform actions together. Therefore computers at stages need a shared reference clock to order activities in time, as well as to keep the clock difference low enough to do different types of interactions.

7.1.2 Idea

1. Each MultiStage computer has NTP, it keeps synchronize the clock of all MultiStage computers. 2. The MultiStage system can also synchronize the clocks by instructing each computer to do a simple clock update from a common time signal. 3. The MultiStage system has function keep checking the clock difference between DSDS and other computers. And compensate the clock difference by adjust each stage's performance start time.

7.1.3 Architecture

The computers of MultiStage use NTP [16] to synchronize its clocks. NTP keeps the clock difference between computers within tens of milliseconds on the Internet and less than one millisecond on the LAN.

Several approaches are used to synchronize the clocks at the different stages. The clock difference between DSDS and the other stages is periodically measured at the System Performance and State Monitoring system and displayed in the Administrator Interaction system. If the clock difference is too large, the administrator using the Administrator Interaction system, can let computers do an on-demand NTP clock update.

7.1.4 Design and Implementation

Figure 7.1 shows the design and implementation of MultiStage system. Clocks are synchronized by utilizing the systems marked in red.

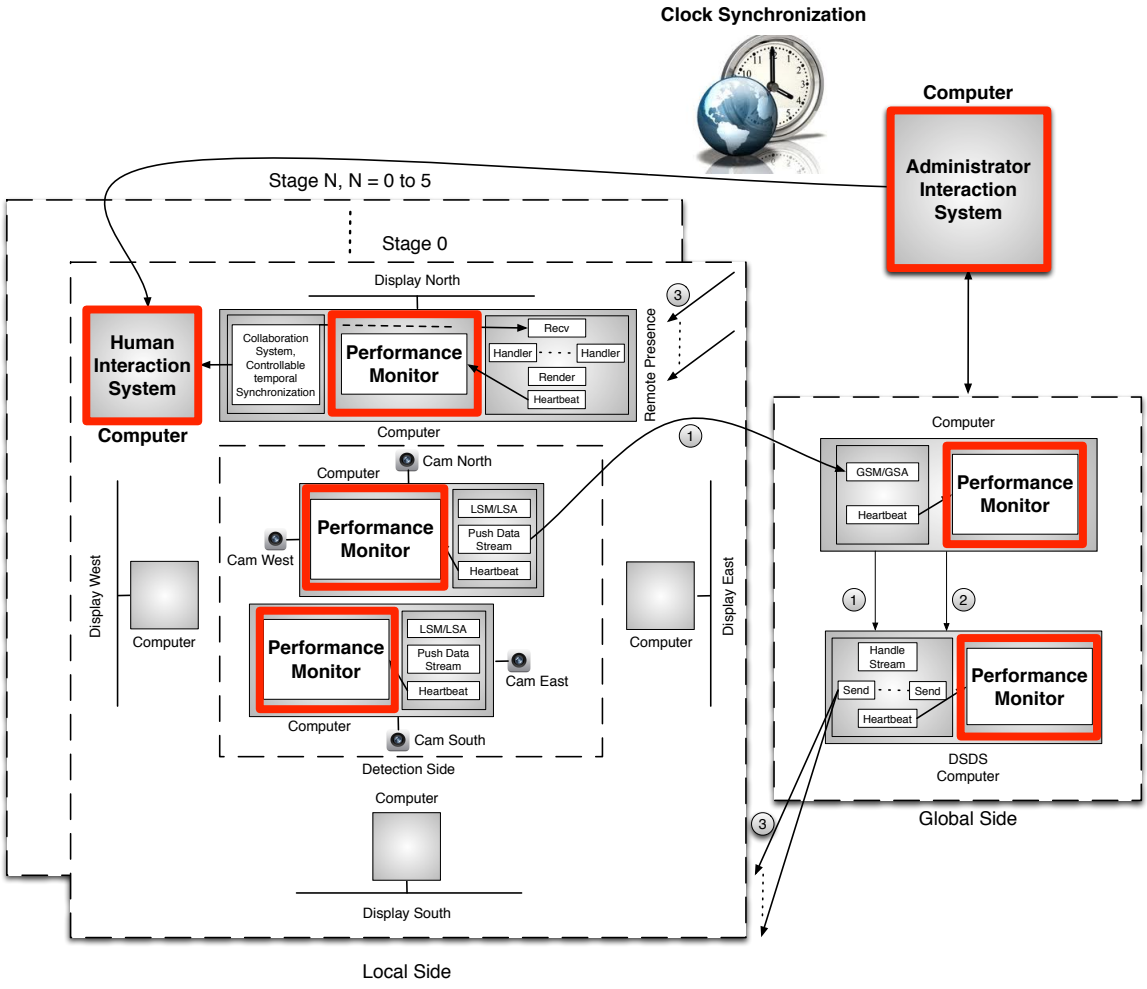


Figure 7.1: The design and implementation of MultiStage.

There are three ways to synchronize the clocks.

1. Each running NTP process on every computer will do periodically clock update.
2. Because NTP synchronizes clocks by letting a computer gradually adjust its clock close to the reference clock. It may take some time for the local clock to adjust to

the reference clock. Therefore, the MultiStage system keeps measuring the clock difference between DSDS and other computers (How measurement is done is described in Section 7.2). The clock differences are shown at the Administrator Interaction system. The administrator can send commands to every computer. Let them do NTP clock update immediately.

3. The MultiStage system also adjusts each stage's performance start time based on the latest measured clock difference. When the Administrator Interaction system sends out the performance start message, the clock difference between the receiving computer and DSDS computer is also included in the message. Each stage can adjust their performance start time based on the received message.

To measure the clock difference, the monitoring process at DSDS sends out a message to every stage to check the clock difference between DSDS and the other stage computers. The time when this message is sent is logged. After a remote stage receives this message, it sends a response message back to DSDS together with its local time when the first message from DSDS was received at the remote stage. The time when this message is received at DSDS is also logged using DSDS's local clock. The DSDS calculates the clock difference based on the logged data. A detailed description and implementation about the calculation is described in Section 7.2.

7.2 System Performance and State Monitoring

7.2.1 Motivation

In order to inform the administrators of a live MultiStage system the state of the systems at every stage, and to recover the computer from failures faster by inform administrator to fix the problem, some of the internal state of every running computer is monitored. Information about the monitored internal state of every computer in MultiStage will be collected and shown through the Administrator Interaction system.

7.2.2 Idea

Each stage computer is monitored by a monitoring process to check and help recover from failures. The idea is from FALCON spy network [59]. The monitoring process measures several performance metrics including end-to-end delays, network latencies, network bandwidth usage, and CPU utilization. These measurements are made available to the Administrator Interaction system. The Administrator Interaction system will display the measured data on the administrator interface to administrator. Administrator will view

the measurements and determine where problems happen. The measured latency and clock difference will be sent to stage computers to adjust their performance start time.

7.2.3 Architecture

The System State and Performance Monitoring system include the following functionalities:

1. Measure the local computers' state including CPU utilization, network bandwidth usage, and memory usage.

2. Measure the global state including latency and clock difference between DSDS and other local side computers.

3. Measure the status of the computer running MultiStage systems, including both local and global side computers.

7.2.4 Design

Figure 7.1 shows the design and implementation of the MultiStage system. The System State and Performance Monitoring system is surrounded by red rectangle.

The System State and Performance Monitoring system is divided into several local sides and a global side. All monitors check its internal CPU utilization, bandwidth usage, and memory usage periodically. The LSM/LSA, Remote Presence, GSM/GSA and DSDS processes will keep reporting to their local monitoring process if they are online. Local side monitors push the measurement to global side monitor. The global side also checks the latency and clock difference between DSDS and the other stages periodically. All the measured data will be pushed from global side to the Administrator Interaction system.

The measurements about latency and clock difference show in figure 7.2. To simulate real case scenario, when measure the latency and clock difference between DSDS and LSM/LSA, the message is sent to LSM/LSA via GSM/GSA. First, a message was sent from DSDS to each local stage monitoring processes. A send time, Ts, is logged at DSDS. When the local stage monitoring processes receive this message, the local monitoring processes send a reply message back to DSDS including its local time, Tr. When DSDS receives the reply message from the local stage monitoring processes, it logs the receive time, Te. The round-trip latency from DSDS to the stages is Te-Ts. We assume that the time when stage monitoring process receives the message sent from DSDS is half of the round-trip latency. Therefore, an estimate of the clock difference between DSDS and one stage is: Te - latency/2 - Tr. All of the measured data will send to the Administrator Interaction system. The transmission protocol is UDP.

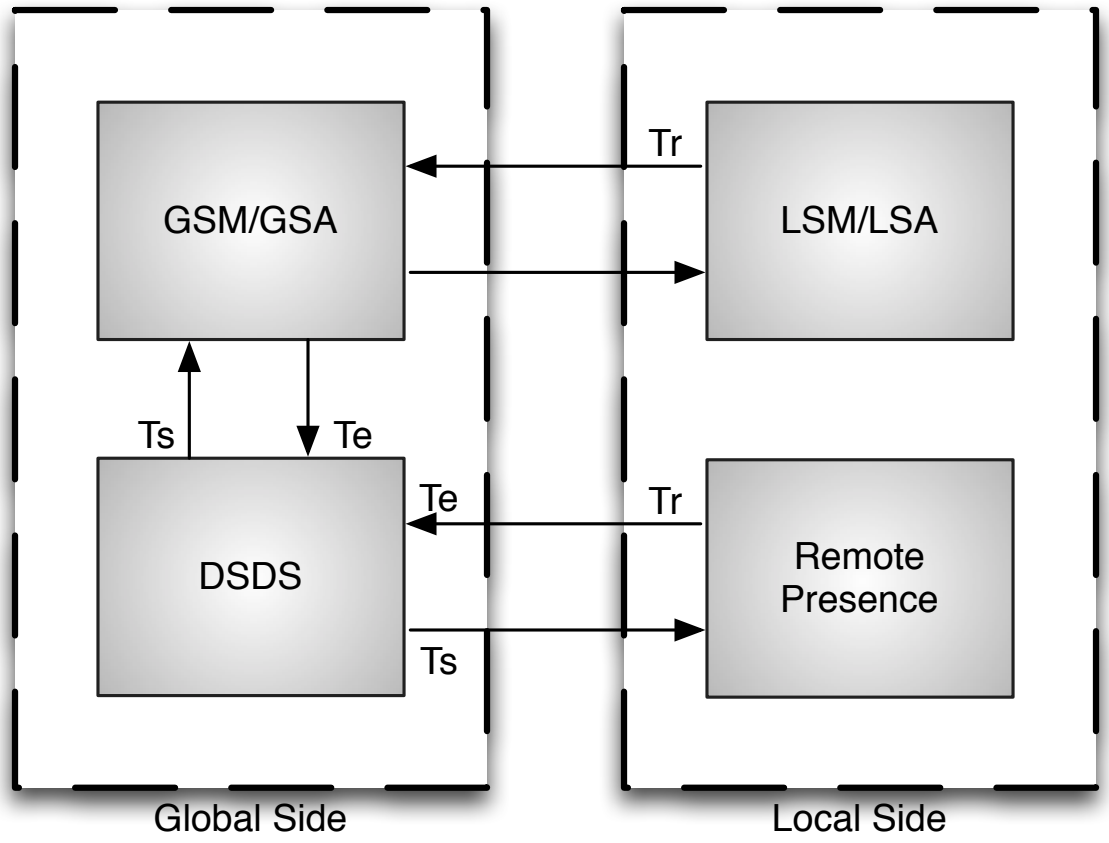


Figure 7.2: The measurement for latency and clock difference.

7.2.5 Implementation

Each local side computer has a local monitoring process and a global monitoring process running on DSDS. Every computer at the local stages has a local monitoring process, it keeps checking on the local state.

We measure the total CPU load, per CPU load, used memory in percent, the sent and received data from Internet. The Python psutil module [13] is used in the measurements. Python psutil provides functions to get total CPU load, per CPU load, Total memory, used memory and total sent and received data through Internet. To get used memory in percentage, simply use (used memory/total memory). To get sent and received data per second, we measure the total sent and received data. Value oldSent and oldRecv indicate the sent and received data in the previous measurement, and oldTime indicates the time when we get this measurement. Value sent and recv indicate the current sent and received data, time indicates the time when we start the new measurement. So we get sent and received data per second by using (sent-oldSent)/(time-oldTime) and (recv-oldRecv)/(time-oldTime). In the current implementation, the measurements are performed every one second. The measured results are sent to the performance monitor

at DSDS. DSDS will forward these results to the Administrator Interaction system.

7.3 System Management - Administrator Interaction System

7.3.1 Motivation

MultiStage system needs functionality to let a human manage the system through system-wide commands including booting, synchronizing local clocks, and getting visualization of internal system-wide state. The measured information from the System Performance and State Monitoring system is made available to the Administrator Interaction system. Administrators can view the data from administrator interaction interface. Some of the commands are sent out from the Administrator Interaction system to Collaboration system running at other local side stages. The commands include performance start time, latency and clock difference between DSDS and other computers on stages. Collaboration system will use the information to adjust local performance start time. When the administrator interface sends out performance start time message to the other stages, the latency and clock difference information received from the System Performance and State Monitoring system is included in the message. Every stage can adjust the shared performance start time to have its individual performance start time.

7.3.2 Idea

The idea is to let the Administrator Interaction system run at a computer operated by the administrator. The administrator can view the monitored data about all stages. Based on the monitored information, the administrator will know what has happened locally at each stage, and be able to recover system from failures or issue commands to stages on time. For example, if the administrator finds out the clock difference between DSDS and one stage is too high by checking the measurements displayed on the administrator interface. He can send out a command from the administrator interface to let stage computers update their clock. The administrator is also able to send out performance start messages, let each stage knows when the performance start.

7.3.3 Architecture

The Administrator Interaction system needs the following functionalities:

1. Receive the monitored information about all stages from the monitoring process running at DSDS computer.

2. Display this information on an administrator interaction interface.

3. Send out control commands to all stages. The commands include: Let all stages update their clock, stop all stages after performance start, and send out performance start time to start performance.

7.3.4 Design and Implementation

Figure 7.1 shows the implementation of the MultiStage system. The Administrator Interaction system is surrounded by a red rectangle.

There are three different messages sent from System performance and State Monitoring system running at DSDS to Administrator Interaction system. One message contains latency and clock difference between DSDS and other computers on stages. One message contains CPU usage, memory usage and incoming and outgoing network bandwidth usage. One message contains information about if a computer is online. Because there are several local side computers, to know from which computer the data come from, a computer ID is added to each message.

On local side, computers running the LSM and the LSA, the ID is "00", "01", etc. The first number indicates the stage and the second number indicates the computer. Computers running the Remote Presence system, the ID is "0", "1", etc. The number indicates the stage. There is only one computer running Remote Presence system on each stage.

On global side, the key for computer running GSM and GSA is "GSA". The key for computer running DSDS is "DSDS". The administrator interface will use the above ID to put the data into right place on the interface. The communication protocol is UDP.

The System Performance and State Monitoring process at DSDS pushes data to the Administrator Interaction system. Figure 7.3 shows the administrator interface used by administrators. The figure shows the Administrator Interaction system interface configured for two stages. The interface also supports configuration for three stages. The place row shows the global side and local sides. Stage columns are local side detection computer running LSM and LSA. Viewer columns are local side computer running the Remote Presence system. Stage01 indicates the computer running at stage-0 and computer-1 (each stage contains two detection computers, each computer contains two cameras, so that there are total four cameras on each stage). The global monitoring process will gather information about CPU usage, memory, network bandwidth usage, latency, and clock difference. The information will be sent to the Administrator Interaction system and

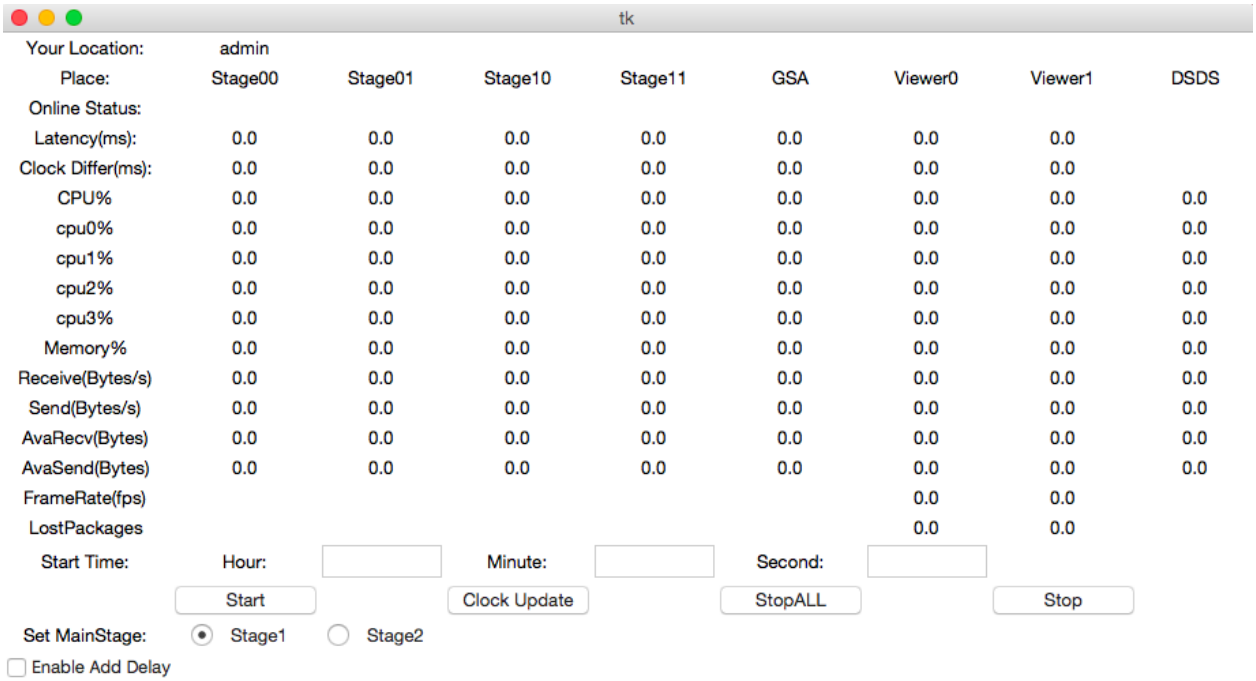


Figure 7.3: The administrator interaction interface, this interface can be managed by system administrators.

displayed on the interface. The online status row indicates if the processes used for MultiStage interaction are running or not. The latency and clock difference between local computer and DSDS is also displayed. Then CPU usage, memory usage and used network bandwidth are displayed. The CPU% row indicates total CPU usage, and the cpu0% row to cpu3% row indicates CPU usage on each core. The sent and received data is displayed, and average sent and received data is not implemented. The FPS and lost packages row only shown on the interface, the monitoring process does not monitor these data in the current implementation. The system can sends out performance start commands, administrator inputs the Hour, Minute and Second information into the interface, and then clicks the Start button. The Clock Update button will send command to let every MultiStage computer update its clock. The StopALL button will stop all running processes. The Stop button just stops the administrator interface. The rest two lines are used to enable some of the approaches to mask the effects of delays. The SetMainStage option will set the live stage for the **Live Stage** approach. The Enable Add Delay option will enable the **local delay** approach. The **Act-By-Wire** approach will be running on the Collaboration system. Therefore, no option for this approach is displayed on the interface. The communication protocol is UDP. The Administrator Interaction system interface is implemented using Python TkInter [60].

7.4 Collaboration Management - Human Interaction System

7.4.1 Motivation

For musical performances, there's often a director to coordinates and tells musicians when to perform an action and what this action is. In the MultiStage system, stages and actors are located at different locations. To let actors perform performance together, it is difficult for one director coordinate all actors that are located at different places on earth. Therefore we need a function to coordinate all actors on stages.

7.4.2 Idea

After receiving the performance start time from the Administrator Interaction system, the Human Interaction system at each stage starts a script according to the performance start time. The script tells actors when they should start an action and what this action is. Actors follow the script, and because the clocks are synchronized, actors in different locations will see the script at the same time or at a different time as adjusted by the masking the effects of delays approach by using the information received from the Administrator Interaction system.

7.4.3 Architecture

The Human Interaction system has the following functionalities:

1. Function to receive performance start command from Administrator Interaction system.

2. Function to calculate performance start time based on the received performance start time, latency and clock difference.

3. Function to display actor script to let actors do their actions based on what is displayed.

7.4.4 Design and Implementation

Figure 7.1 shows the implementation of the MultiStage system. The Human Interaction system is surrounded by red rectangle.

The message about performance start command includes:

1. Hour, minute and second about when performance begins. This is the shared performance start time.

2. Clock difference between local computer and DSDS. The shared performance start time is adjusted by using the clock difference value.

3. If **Live Stage** approach is enabled, data about how early the performance begin at local stage is included. If **Delay Local Remote Presence** approach is enabled, data about how long time local stage has to wait before performance begins is included.

At performance start time, an actor script will be shown on the display. It keeps time inform actors what to do, and provides them with a countdown for when they should start performing an action.

The communication protocol to receive the performance start command is UDP. In the current implementation, one display per stage is used to visualize actions for all actors on the same stage. The Human Interaction System is running together with the Collaboration system on the same computer. When countdown finishes, a handshake image will be displayed on the display. Actors can do remote handshake by following the actor script. The actors' script is implemented using Python Pygame [28].

Chapter 8

Performance Experiments using MultiStage

8.1 Type of Experiments

Two types of experiments (Objective and Subjective experiments) were conducted on and with the MultiStage system. To identify potential bottlenecks, resource constraints and latencies that can influence human interactions through the system.

Objective experiments were done to measure a set of performance metrics characterizing the performance of the system with different factors. These experiments measured the resource usage and latencies of the MultiStage system. The main contributions of latencies come from the objective experiments and not from the subjective experiments. The objective latency experiments provide more accurate values because we know exactly when the experiment begins, and when it ends. For example, the start point for calculating round-trip latency is when we start sending a message, and the end point is when the message is received. For the end-to-end one-way latency, we know the start point is when real event happens, and the end point is when the same event happens on the remote presence on the display. The latency can be determined accurately by calculate the time difference between the begin point and the end point. Provide a reliable value for us to decide when to start or stop masking the effects of delays.

Subjective experiments were done to perform very informal user studies on how the system respond to simple human movements and how humans react to latencies added to interactions through the system. The values are subjectively decided based on a few people's opinion. In some of the subjective experiments, the added delay increased by 50ms or 100 ms each time. Not all possible delay values were covered. The informal user study was a weak indication of what kind of delays the system can expect should result in masking actions. The type of interaction is also very simple handshake type of

interaction. The main purpose is to have a basic understanding of the impact of delays on actors, and to understand if each masking approach is able to mask the effects of delays for handshake type of interaction.

The objective latency experiments found out how much latency the MultiStage system added on actors view of each other. The subjective latency values give a hint of how much latency can be noticed by human. And how much latency can be tolerated by human for handshake type of interaction. By comparing latency values found in the objective and subjective experiments, we are able to determine when to apply approaches to mask the effects of delays for handshake type of interaction.

8.2 Platform

Figure 8.1 shows the topology of the MultiStage system running the experiments. Computers on Global Side, Stage 1 and Stage 2 were 2011 Mac minis at 2.7 GHz Intel Core i7 and with 8 GB 1333 MHz DDR3 memory. Computers on Stage 3 were Mac minis at 2.5 GHz Intel Core i5 and with 4GB memory. Each stage had three computers: two computers each equipped with two cameras each running LSM and LSA, and one with a large display running the Remote Presence system. The global side had two computers: one for the GSM and GSA, and one for the DSDS system. Each stage and the global side had a 1 Gbit/s network switch. All switches were connected to a switch with access to the Internet. For all experiments, all local side stages were on the same 1 Gbit/s switched Ethernet LAN inside the Department of Computer Science at UiT: The Arctic University of Norway. The global side DSDS computer was either on the same LAN as the stages, or located on a Planetlab [61] computer (Dell PowerEdge 1950, Linux) at the University of Oslo, 1500 km away. In this case, all data sent between stages went from Tromsø to Oslo and back again. This separated the stages across the Internet.

8.3 Objective Experiments

Using the Python psutil module [13], we measured the CPU utilization, amount of physical memory in use, and incoming and outgoing network traffic for all computers in use (See Chapter 7.2).

Factors in the experiments were the number of stages (1 to 3), the resolution of the images from the cameras (3D point cloud, 1000 to 5000 points per image), the number of cameras per stage (0 to 4), and the location and network type used by the DSDS subsystem distributing data between stages (LAN in Tromsø versus WAN in Oslo). The reason to send maximum 5000 points per image is because UDP protocol has an upper

Global Side, Stage 1 and Stage 2: 2.7 GHz Intel Core i7 CPU, 8GB RAM, Mac OS Lion

Stage 3: 2.5GHz Intel Core i5 CPU, 4GB RAM

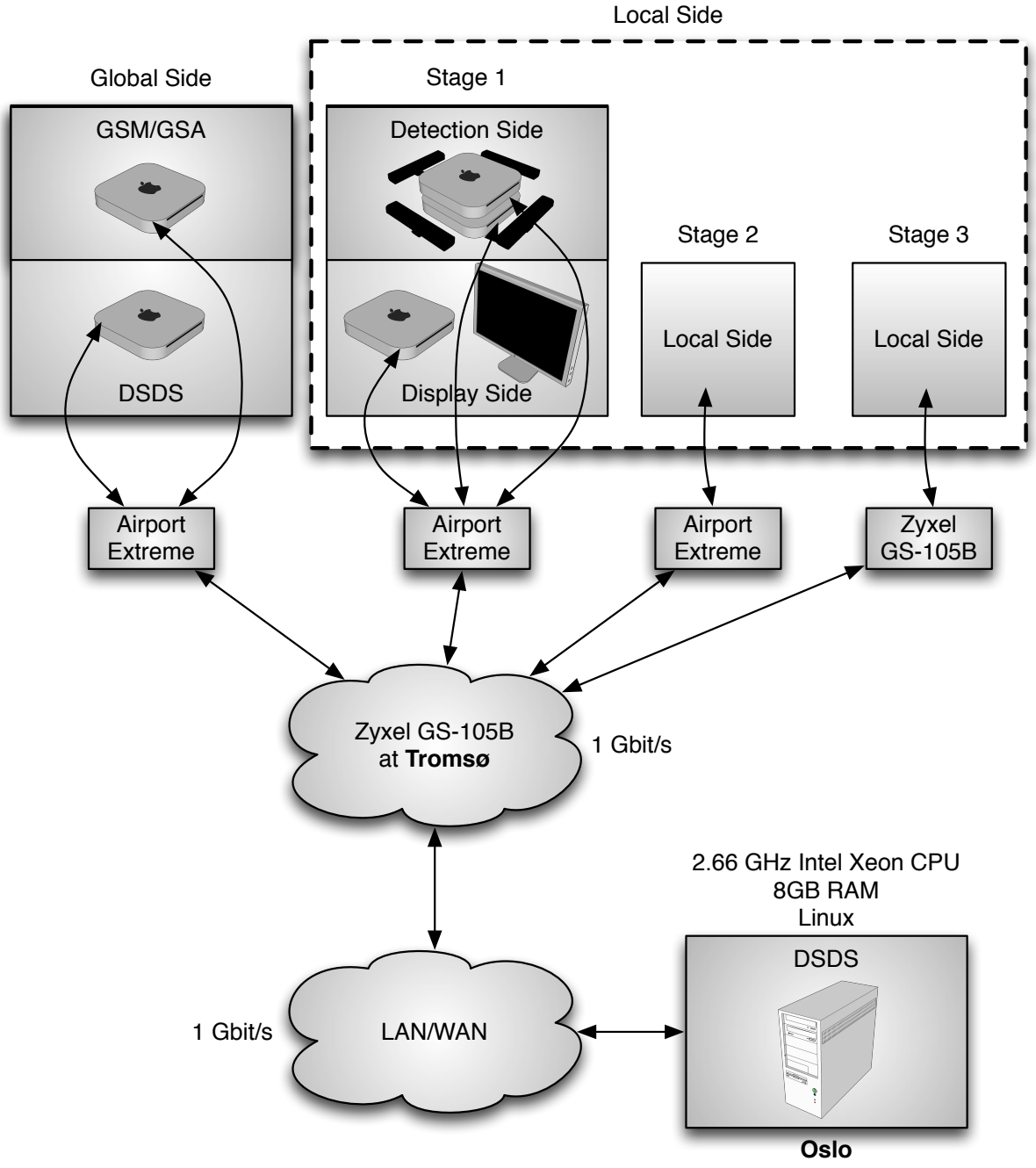


Figure 8.1: Topology of system running the experiments.

limit for each data packet. In 5000 points per image case, the size of each data packet is close to the UDP upper limit.

8.3.1 Resource Usage Metrics

The reason for the experiment is to understand the performance and resource constraints of the MultiStage system. And to understand how well the MultiStage system scale when number of stages increases.

The experiment were configured to the extreme case, 3 stages, where all four cameras were running at each stage pushing the maximum number of images (30 fps and 5000 points per image) to DSDS. Each stage sends out four data streams (each data stream is generated by one camera on the stage) to DSDS, this result in a total of 12 incoming data streams at DSDS. All three stages subscribe to all the streams from DSDS, this result in a total of 36 outgoing data streams at DSDS.

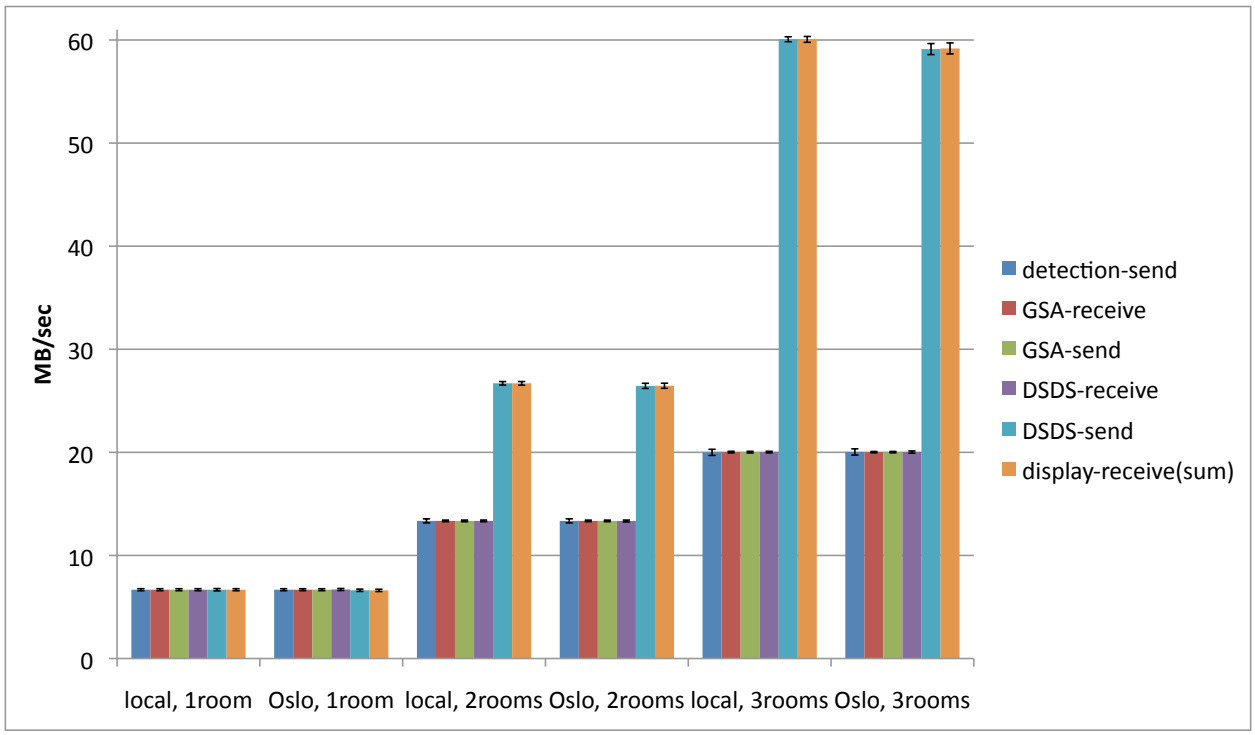


Figure 8.2: Incoming and outgoing network bandwidth usage with one, two, and three stages through a LAN and through the Internet. Each stage has four running cameras, the resolution of captured images are 5000 points per image.

Figure 8.2 shows the result of the network bandwidth usage in the experiment. There is practically no loss of data in the experiments with the DSDS on the same LAN as the stages. When we separate the stages with a WAN by locating the DSDS on a computer in Oslo 1500km away, we see just an insignificant increase in data not getting across to

all stages. The implication is that we can expect that the system typically will have bandwidth available even when the stages are separated by the Internet.

Figure 8.3 shows the result of CPU load and used network bandwidth. The CPU usage in all cases is either very low or low. This extreme case is set up to maximise the CPU load and network traffic of each computer in such a multi-stage configuration. But even in this case, the highest CPU load on the display side computer is less than 25 percent. And there is still available network bandwidth on a Gigabit Ethernet. The implication is that the system is not resource limited.

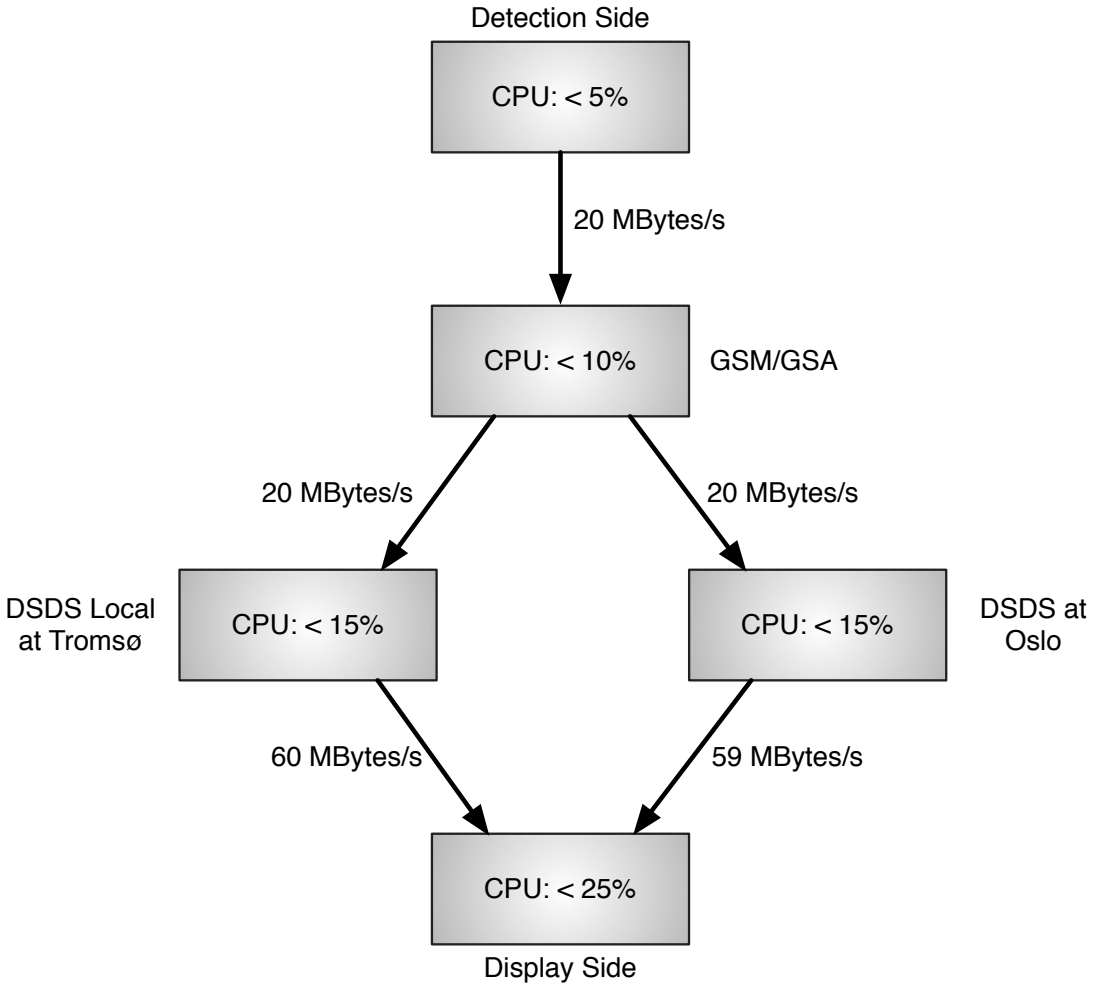


Figure 8.3: The CPU utilization and used network bandwidth usage in the case of three stages, four cameras running on each stage, and the image from the camera being 5000 points per image.

8.3.2 Latency Metrics

To better understand how much latency the MultiStage system added to interactions between actors. And compare with the latency measured in subjective experiments to

determine when to enable the approaches to mask the effects of delays. The following experiments are conducted:

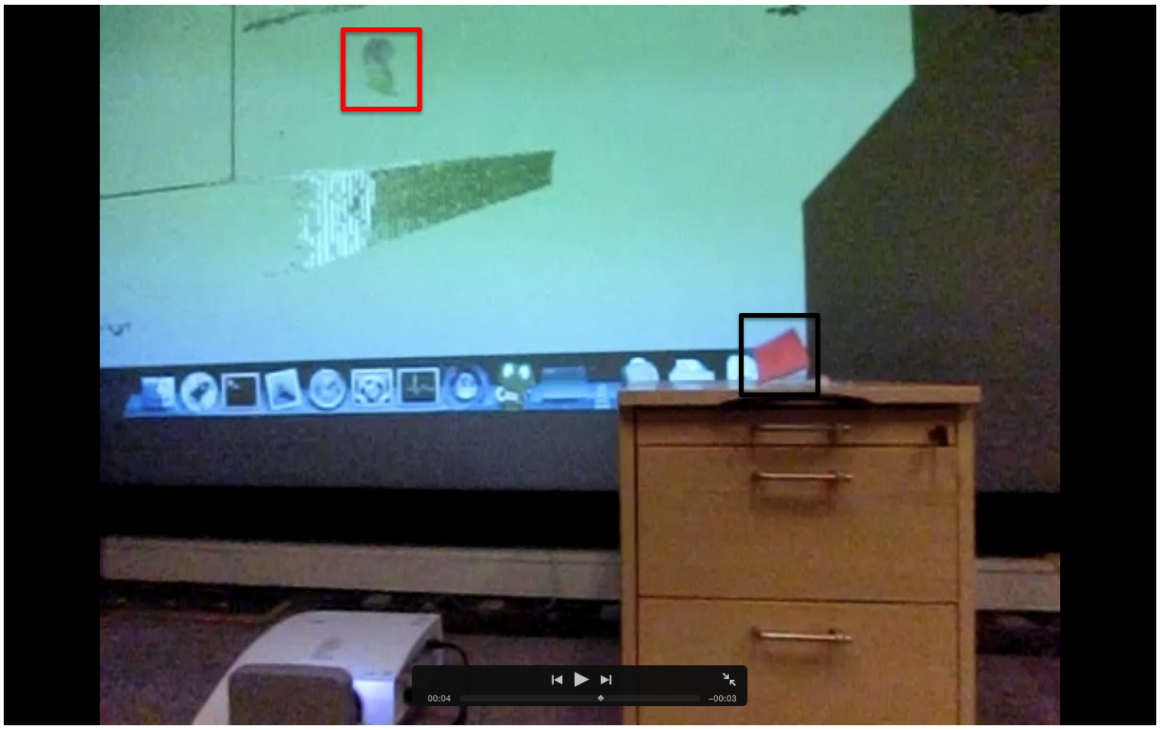
1. Global-to-local round-trip latency: The network transmission latency when send a message until the message is received.

2. System end-to-end one-way latency: When an event happens until it is displayed on the display.

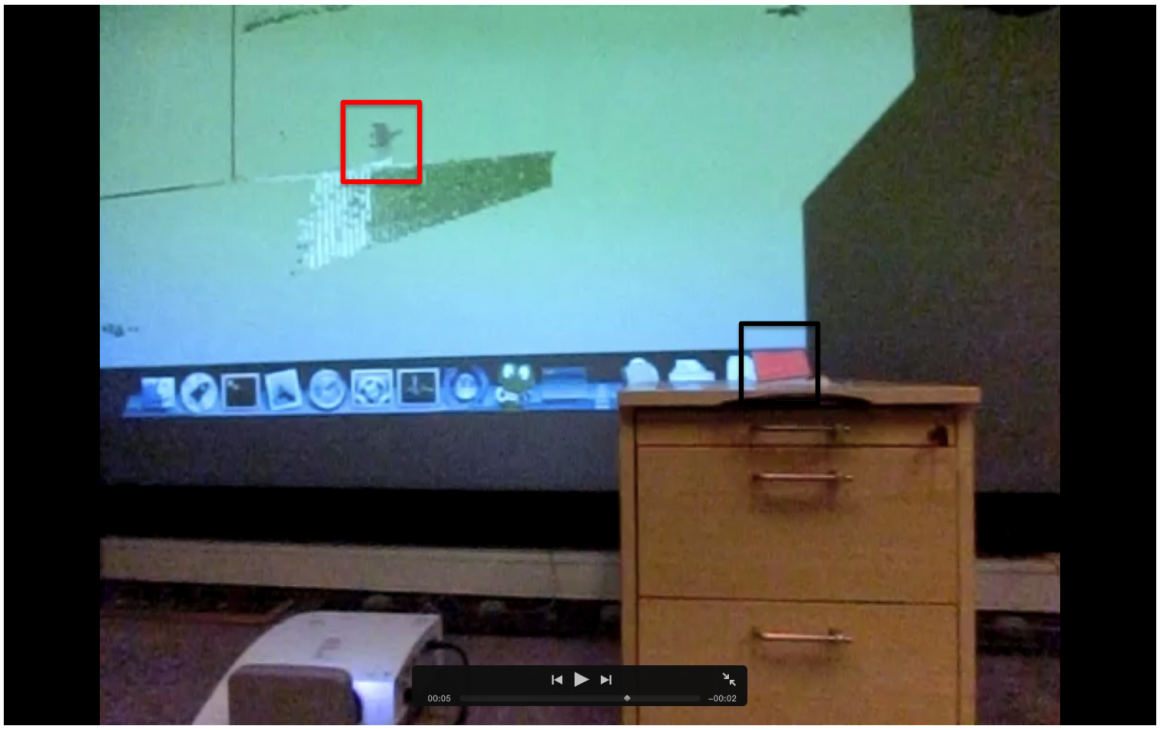
3. Actor-to-actor round-trip latency: Twice the system end-to-end one-way latency. When first actor initial an action, until he sees another actor's reaction is displayed on the display.

1. **Global-to-local round-trip latency**: The time going from the global side DSDS computer to a local side stage computer and back (described in Section 7.2). The reason to conduct this experiment is to check the general network transmission latency. We measured this by recording the time between when we send a message from DSDS to a stage, and when a reply message comes back to DSDS. We kept sending this message for a period of about 5 minutes, the latency was measured every 1 second. The Global-to-local round-trip latency is the average result of the measured values. When all stages and the global side were on the same LAN, the round-trip latencies were between 1 and 2 ms. When the DSDS system was on a computer in Oslo, the round-trip latencies were around 32 ms. This matches well with measurements reported by PingER [62] for Europe.

2. **System end-to-end one-way latency**: The time between a physical event happening on a stage being picked up by the cameras, and a visualization of the actor being displayed on the same stage. The reason to conduct this experiment is to find out the system latency for MultiStage. We used a video camera (CASIO EX-ZR200) with a high frame rate (240 FPS) to record several videos of an object and the remote presence of the object shown on a display behind the object. The videos were recorded directly after each other. Figure 8.4 shows the image example of the measurement. The black box indicates the original object. The red box indicates the remote presence of the object. Figure 8.4a shows the original object hits the desk and the remote presence of the object is still flying. Figure 8.4b shows the remote presence of the object hits the desk after some frames. We then counted frames to see how many frames it took from when the object hits the desk to when the visualization caught up. The video example of the measurement can be viewed from [63]. On a LAN, the end-to-end latency was between 90 and 125 ms. With DSDS at the computer in Oslo, the end-to-end latency was between 100 and 158



(a) The object hits the desk, but the remote presence of the object still flying on the air.



(b) The remote presence of the object hits the desk.

Figure 8.4: The measurements of system end-to-end one-way latency. The black box indicates the original object. The red box indicates the remote presence of the object.

ms. The variation in measured latency is because of several factors, including the distributed architecture of the prototype and the frame rate of the projector, video camera (240 FPS) and the Kinects (30 FPS), and other traffic on the LAN and WAN. Informal evaluation seems to indicate that users will tolerate visual interaction latencies in the order of a few hundred ms.

3. **Actor-to-actor round-trip latency**: The delay that actors experience between when they do an action and when they see the remote presence of another actor reacting. This is the typical latency when actors interact together. The typical latency between actors is twice the system end-to-end latency. Using the measured results from the system end-to-end one-way latency, the actor-to-actor round-trip latency is between 180 and 250 ms when DSDS computer is located at Tromsø, and the latency is between 200 and 316 ms when DSDS computer is located at Oslo.

8.4 Subjective Experiments

Subjective experiments were done to find out if the MultiStage system could provide low enough latencies and react fast enough to do masking. Therefore the subjective experiments are lightweight and not meant to be a user study. The experiments are mainly conducted by letting one to two actors perform arm movements. The actors are myself, supervisors and my colleagues. The type of movements is mainly handshake type of movements. In most of the experiments, we used a camera to capture actors' movements and the corresponding remote presences on the display. Videos are recorded and played back to subjectively decide when the interaction will break. Values are subjectively decided based on several people's observation.

8.4.1 Latency Metrics

1. Human response latency: Find out how much time it takes for human react to an action as well as find out if the reaction time varies for rapid and slow type of arm movements.

2. Human noticeable latency: Determine a lower boundary of enable approaches to mask the effects of delays. Whenever the performance requires a fast action-reaction type of interaction, the system might need to start masking the effects of delays when human can subjective notice there is a delay between two actions.

3. Human tolerable latency: Determine a upper boundary of enable approaches to mask the effects of delays. Higher latency might be tolerated before apply the

approaches to mask the effects of delays.

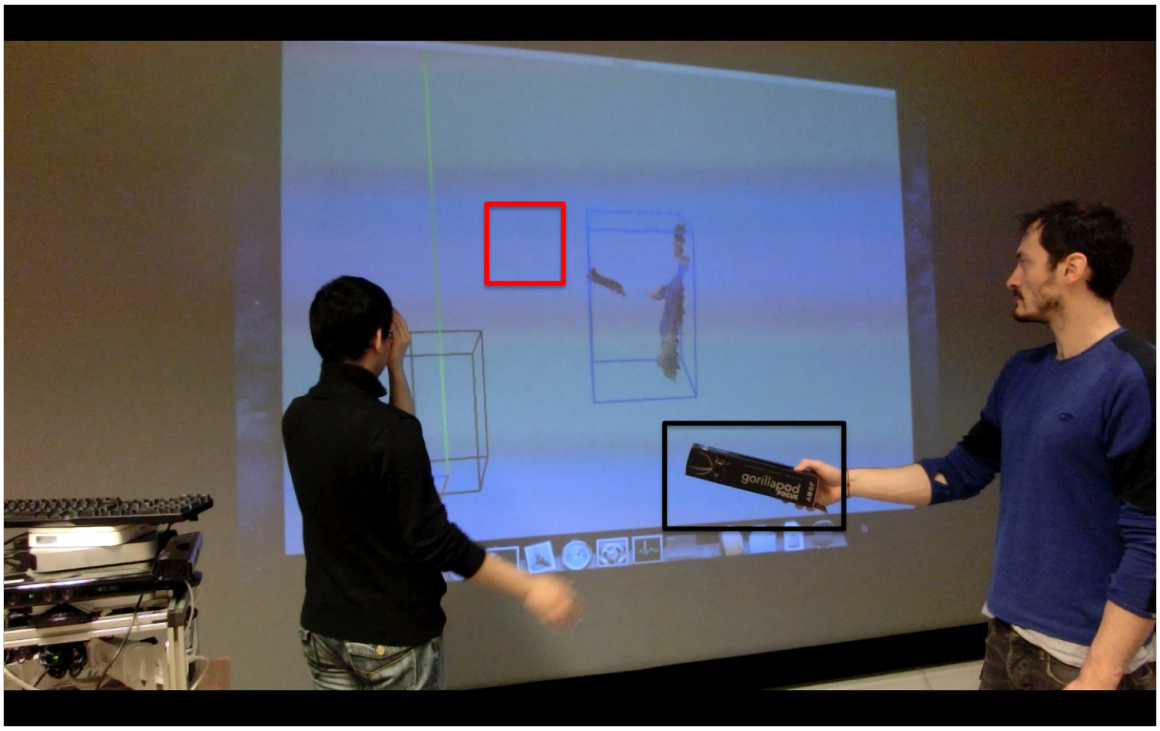
4. When to start and stop masking (Act-By-Wire): Determine when to start and stop the Act-By-Wire approach and make a smooth switch between pre-recorded and real video.

5. Cost of masking: To understand the CPU utilization when masking approaches are enabled.

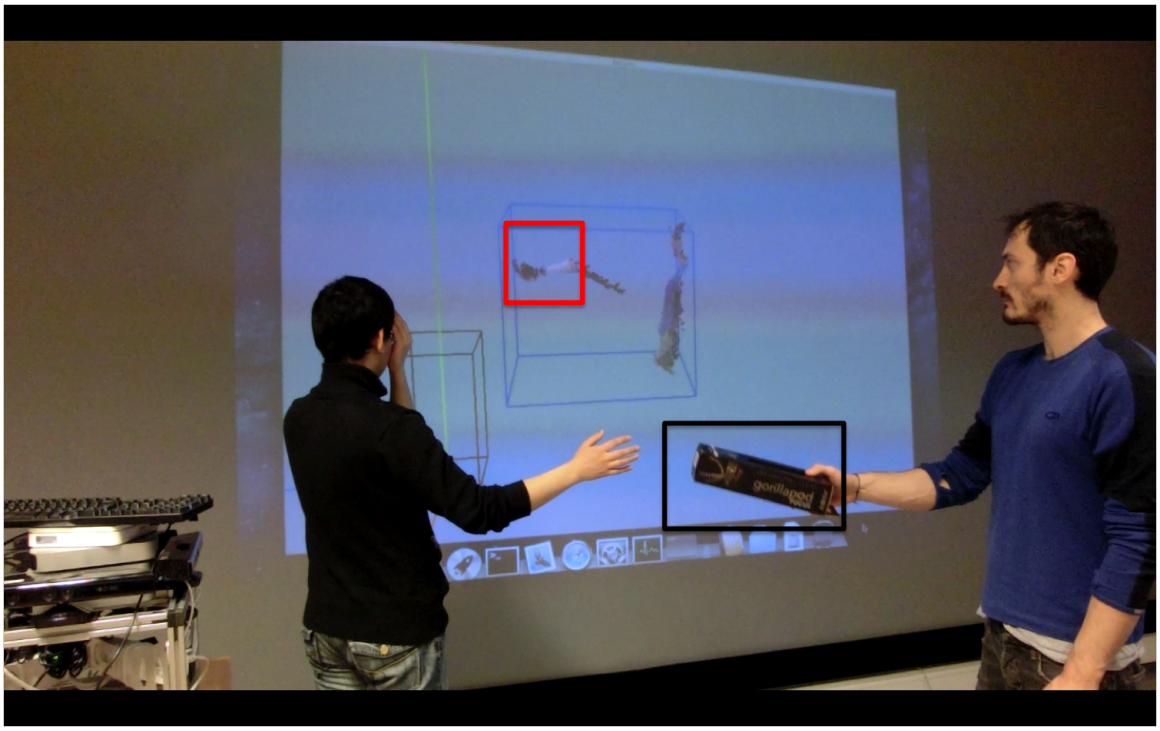
6. Maximum system end-to-end one-way latencies for approaches to mask the effects of delays: Find out the maximum system latency achieved by the masking approaches at least partially successful at masking the effects of delays. The experiments include: (a) Satisfactory Synchrony between all remote presences at every stage. (b) Satisfactory Synchrony between all actors at every stage. (c) Satisfactory Synchrony between all actors and all remote presences at every stage.

Human response latency: The time it takes for a human actor to reacts to another actor's action. We used a high frame rate camera (240 FPS) to record two actors' actions, and counted frames from when one actor initiated an action until the other actor responded to the action. The actions were repeated several times directly after each other. The actions used were rapid and slow moving arm movements. Different videos for rapid and slow movements were captured. This is because we want to find out if the human response latency varies for rapid or slow type of arm movements. The human response latency is about 345 ms. We did not find that the latency varied significantly with the speed of an action.

Human noticeable latency: The length of time at which a human actor will notice an action is delayed. We simultaneously observed an actor and the corresponding remote presence at Tromsø. When the actor moved an arm, the remote presence moved an arm. In software, we artificially added a delay to the remote presence until we noticed that the remote presence lagged behind the actor. The added delay is from 0 to 150 ms, increased by 25 ms each time. For each added delay, we used a camera record video for the actor and the corresponding remote presence on the display. The videos were captured directly after each other. The arm movements were repeated several times in the video. When the added delay was more than 100 ms, we did notice a difference in the movement between the actor and the remote presence. The system end-to-end one-way latency is about 90-125 ms when DSDS computer is located at Tromsø. This means the total noticeable latency is the sum of the system end-to-end one-way latency and the added delay. It is about 190-225 ms.



(a) Actor starts an action, the remote presence of another actor's action not displayed on the screen.



(b) The remote presence of another actor displayed on the screen later.

Figure 8.5: The experiments of human tolerable latency. The black box indicates actor who starts an action. The red box indicates another actor who reacts to the action.

Human tolerable latency: The actor can tolerate before the illusion of being on the same stage with other actors breaks. We observed an actor shaking hands with another actor on the same stage. We then moved one of the actors to a remote stage, and repeated the shaking of hands. We now observed an actor shaking hands with the remote presence of the other actor. The delay between the two actors was artificially increased until we subjectively decided that the handshake was not happening as fast as it did when the actors were physically on the same stage. We tried both rapid hand movement and slow hand movement. This is because we want to determine the tolerable latency for different types of hand movements. For fast hand movement, the added delay is from 0 to 500 ms, increased by 50 ms each time. For slow hand movement, the added delay is from 0 to 1000 ms, increased by 100 ms each time. For each added delay, we used a camera record video for the actors and the corresponding remote presences on the display. The videos were captured directly after each other. The hand movement was repeated several times in each video. Figure 8.5 shows the image example of the experiment. The black box indicates actor who starts an action. The red box indicates another actor who reacts to the action. Figure 8.5a shows an actor starts an action, but the remote presence of another actor's action still not displayed. Figure 8.5b shows the remote presence of another actor's action was finally displayed. The video example of the measurement can be viewed from [64]. We subjectively decided that for a rapid hand movement, it is not tolerable when 150-200 ms delay was added. The total actor-to-actor round-trip latency is in this case about 350-400 ms (+ 200 ms actor-to-actor round-trip latency). For slow hand movement, it is not tolerable when 600 ms delay was added. The actor-to-actor round-trip latency is then about 800 ms.

For a handshake type of interaction, longer delays bordered on creating the feeling that the remote actor was being obnoxious by delaying just a bit too long before responding to a handshake. However, this was not experienced unless we artificially added delays. This indicates that the prototype is able to maintain the illusion of being on the same stage for a handshake type of interaction. However, we observe that the typical actor-to-actor round-trip latency in Europe is around 300 ms or more. Consequently, when actors interact fast and rapidly, the system can expect to have to mask the effects of the delays.

When to start masking (Act-By-Wire, display pre-recorded data streams): We simultaneously observed an actor moving an arm, and the corresponding remote presence (figure 8.6). In software we artificially added delay to every image. The artificially added delay is 200 ms, 250 ms, and 300 ms. The reason we select these delays is because we found out human tolerable latency for rapid hand movement is about 350-400 ms. And the system end-to-end one way latency is about 100 ms. The artificial delay is the difference between human tolerable latency and the system latency. For each added delay,

we used a camera record video for the actor and the corresponding remote presence. The videos were captured directly after each other. And we subjectively decided when 250 ms delay was added, we observed a noticeable difference on the display.

For the above experiment, we also measured the **total delay** between when this image has been timestamped (the system add timestamp after it captures the image) and when the Collaboration system receives this image. We keep running the system for 5 minutes, and the total delay is measured every 1 second. We then calculate the average value of total delay, and it is about 280 ms (when 250 ms artificial delay was added). This means if the delay always more than 280 ms in performance, the Collaboration system needs to mask the effects of delays. And if the delay always less than 280 ms in performance, the Collaboration system does not need to mask the effects of delays.

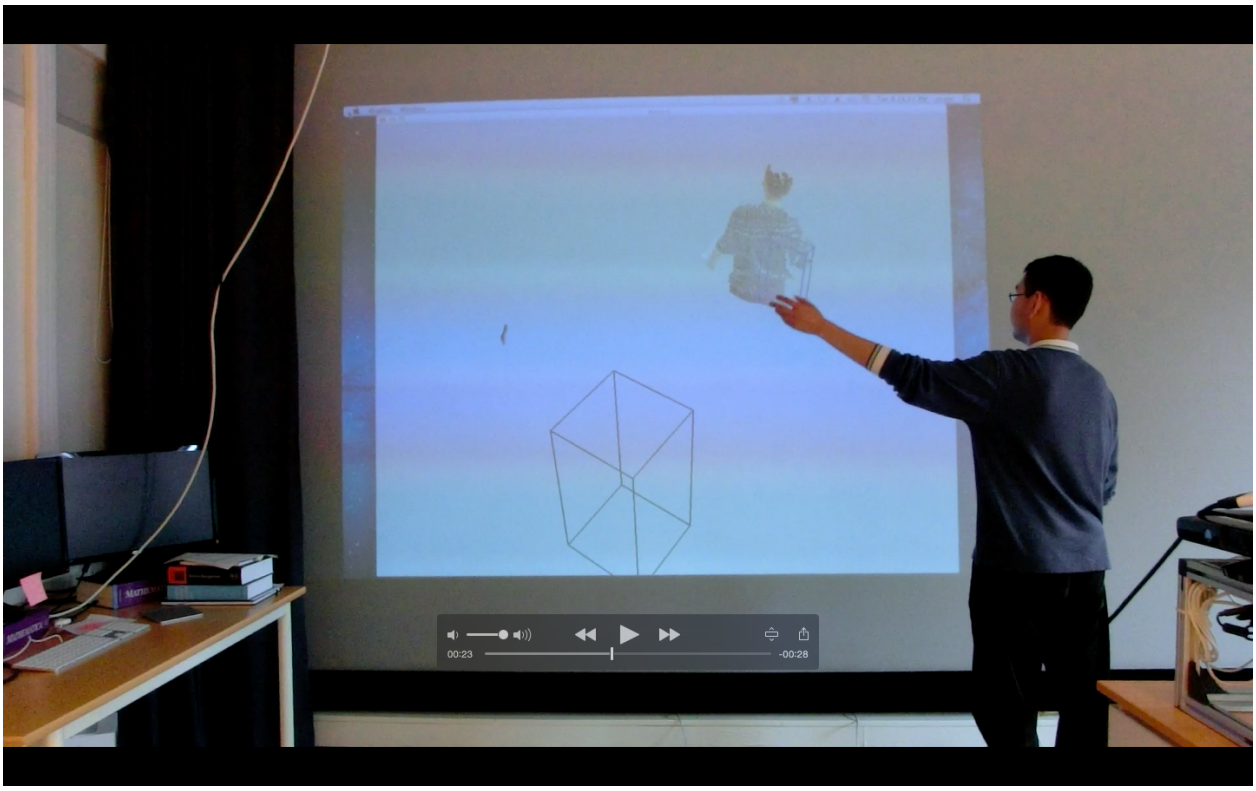


Figure 8.6: Find out the when to start Act-By-Wire approach.

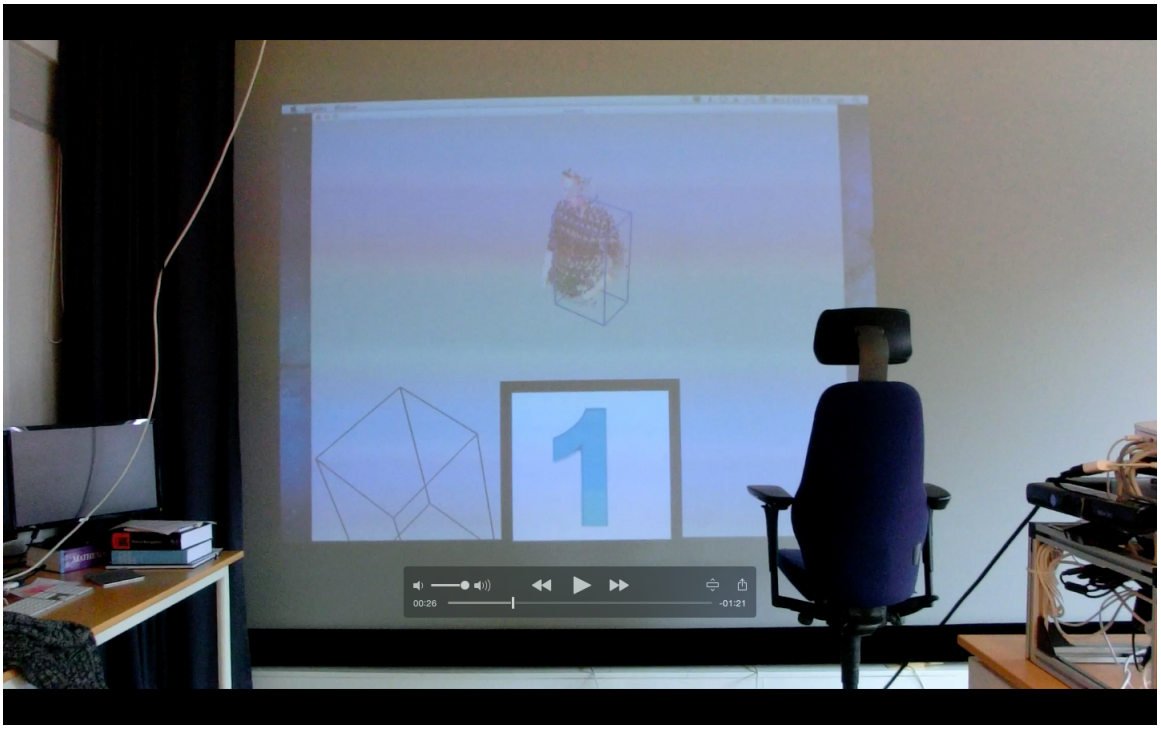
But there will be jitter together with latency, what about the case when total delay varies above or below 280 ms during performance? We let the actor keep doing arm movements as shown in figure 8.6. We observed the actor and the corresponding remote presence. In software, we artificially added a random delay to every image used to create the remote presence. We tried different combinations of delays and number of images delayed. The number of delayed images is increased from 10% to 90%, increased by 10% each time. For each increase, we used a camera record video for the actor and the

corresponding remote presence on the display. The videos were captured directly after each other. We found that when more than 50% of the received images during a period of 3 second were delayed by 280 ms or more, there is a subjectively clearly visible lag in the remote presence versus the actor. We therefore determine that when 50% of the images arrive 280 ms late during the last 3 second, this is the threshold for when to start masking. This is a threshold that can be customized for different usage scenarios.

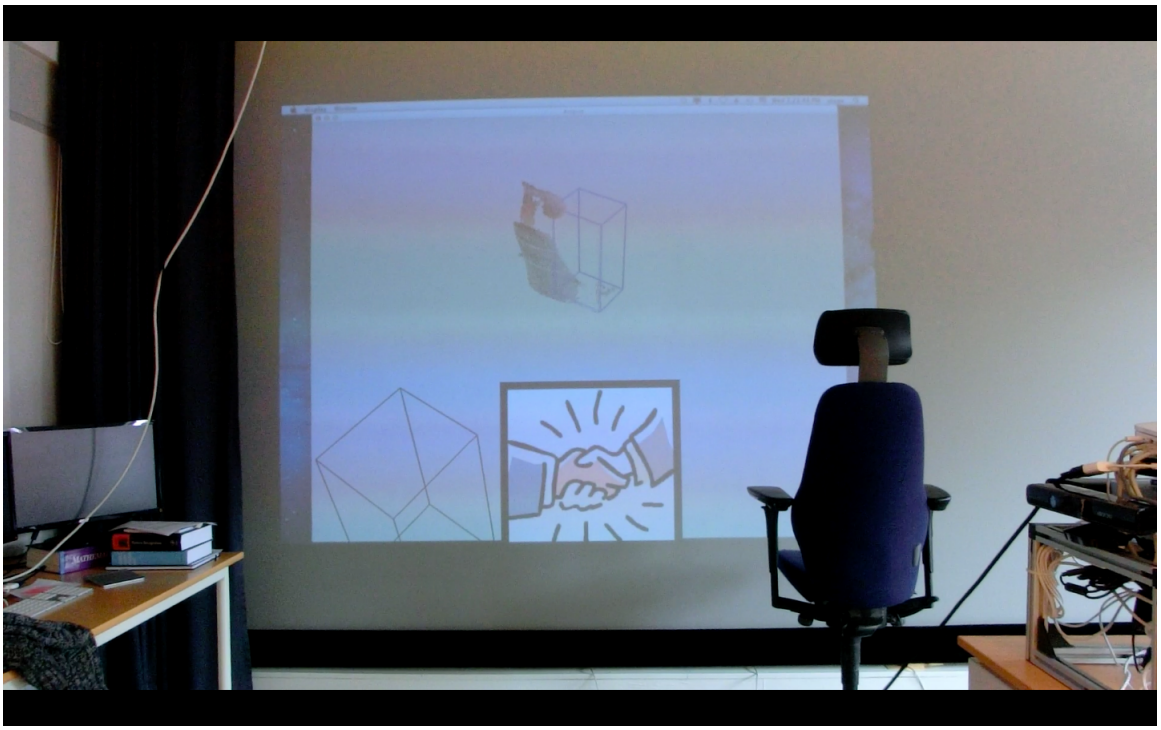
When to stop masking (Act-By-Wire, stop display pre-recorded data streams and show live data streams): When masking is active, we need to establish a threshold for when to stop masking. We observed a chair and the remote presence of the chair (figure 8.7). The reason we use a chair instead of an actor is because no movements are needed in this experiment. To find out when to stop Act-By-Wire approach, we only need to observe the switch back and forth between pre-recorded video (figure 8.7a) and real video (figure 8.7b). We artificially create a situation where more than 50% of the images used to create a remote presence arrive too late (more than 280 ms). Consequently masking is done by the system. For the experiment we used the Act-By-Wire prerecorded video masking approach. In software, we set when less than 45% images arrive too late, the pre-recorded video will be switched back to real video. We gradually decreased this percentage by 5% from 45% to 30%. For each decrease, we used a camera record video for the chair and the remote presence on the display. The videos were captured directly after each other. We observed the switching back and forth between the live streaming of the remote presence and the prerecorded stream. When 35-40% of the images arrive late in the past 3 second, the switch from the pre-recorded to the live streaming results in a transition without the observer noticing obvious effects of the delay. A higher percentage leads to a switch sooner, but the transition can be too fast and result in a blending in of the live streaming video with noticeable delays. A lower percentage results in the prerecorded video playing for too long, and this can become noticeable in itself. The video example of switch back and forth of pre-recorded video can be viewed at [65].

The goal for the above two experiments is to find a balance between when to start masking and when to stop. This can be different for different user activities and needs. In summary, we checked for late images during the last 3 second. A shorter period will lead to less delay in starting masking when needed, and a longer period will result in more delay. For shorter periods, a higher threshold for stopping the masking will reduce the likelihood of switching back and forth. For longer periods, a lower threshold for stopping the masking will increase the likelihood of switching back to the live streaming.

Cost of Masking: The CPU utilization at a remote presence computer without and with the technique to mask the effects of delays active was measured. Two cameras were used to send images for two remote presences to a single remote presence computer. The



(a) When more than 50% images arrive late, MultiStage will use pre-recorded video.



(b) When less than 45% images arrive late in past 3 second, MultiStage will switch back to real video.

Figure 8.7: Find out when to stop Act-By-Wire approach.

CPU utilization without masking was about 22%. When masking was done for both remote presences using two prerecorded videos, the CPU utilization was basically the same, 22%. When masking was done using two human skeletons, the CPU utilization at the remote presence computer went down to 9%.

We explain this by observing that a significant part of the CPU load was consumed to display videos, making the masking itself insignificant. The very simple human skeleton approach is clearly less CPU demanding. We explain this by the simplicity of the model and that the model uses the display much less than the videos do.

Approaches to masking the effects of delays	Satisfactory synchrony between all remote presences at every stage	Satisfactory synchrony between all actors at every stage	Satisfactory synchrony between all actors and all remote presences at every stage
Act-By-Actor	< 190-325ms	< 190-325ms	< 190-325ms
Act-By-Director	< 390-525ms	Any	< 390-525ms
Live Stage	Any (only at live stage)	< 390-525ms	Any (only at live stage)
Delay Local Remote Presence	Any	Any	< 390-525ms
Delay Locally All Remote Presences Waiting for the Slowest	Any	Any	< 390-525ms
Act-By-Wire (blend in pre-recorded remote presence)	Any	Any	Any
Act-By-Wire (blend in on the fly created remote presence)	Any	Any	Any

Figure 8.8: Approaches to masking the effects of delays. The delay values are the maximum system end-to-end one-way latencies for when an approach will be at least partially successful at masking the effects of delays.

Maximum system end-to-end one-way latencies for approaches to mask the effects of delays to achieve satisfactory synchrony between actors and remote presences: Figure 8.8 shows the maximum system end-to-end one-way latency at which each masking approach is in principle at least partially successful at masking the effects of delays.

The maximum system end-to-end one-way latency was measured by having two actor stands in front of two cameras at the same stage, and following actor script to do handshake. As shown in figure 8.9 to 8.12, different techniques to mask the effects of delays are enabled. The remote presences of two actors are displayed on the same screen. We artificially added delay to the remote presence on the right side of the display. We

gradually increase the delay by 100 ms from 0 ms to 500 ms (for some experiments, the added delay was up to 700 ms). For each approach, we also added 1000 ms delay as the extreme experiment case. For each added delay, we used a camera record video for the actors and the corresponding remote presences on the display. The videos were captured directly after each other. The handshake was repeated several times in each video. Then we subjectively decided the temporary causal synchrony would break after we see a noticeable difference between the two remote presences. The system end-to-end one-way latency when DSDS locally at Tromsø is 90-125 ms. The maximum system end-to-end one-way latency will be the sum of the system end-to-end one-way latency and the added delay.

1. Act-By-Actor: when 100-200 ms delay was added, we will see a noticeable difference, and the maximum end-to-end one-way latency is 190-325 ms.

2. Act-By-Director: When 300-400 ms delay was added, we will see a noticeable difference, and the maximum end-to-end one-way latency is 390-525 ms. Because the performance will start at the same time, all actors will always be synchronized. In figure 8.9, 1000 ms delay was added to the remote presence on the right side of the display. Although the remote presences are not synchronized, but all actors will be synchronized because they follow the same actor script. The video example of Act-By-Director approach can be viewed at [66].

3. Live Stage: When 300-400 ms delay was added, we will see a noticeable difference, and the maximum end-to-end one-way latency is 390-525 ms. The secondary stages will start earlier so that when the actor at live stage start performance, all the remote presence will start at the same time. Therefore, all remote presences and actors will be synchronized at the live stage. Figure 8.10 shows the live stage approach, actor on the left side located on the live stage and actor on the right side is located on the secondary stage. To simulate network transmission latency, 1000 ms delay was added to the remote presence on the right side of the display. The secondary stage will start performance 1000 ms earlier than the live stage. The display shows the performance of the remote presences on the live stage. Figure 8.10a shows the actor on the secondary stage will start performance 1000 ms earlier. Figure 8.10b shows the actor on the live stage start performance. All actors and all remote presences on the live stage will be synchronized. The video example of Live Stage approach can be viewed at [67].

4. Local Delay: When 300-400 ms delay was added, we will see a noticeable difference, and the maximum end-to-end one-way latency is 390-525 ms. Because the perfor-

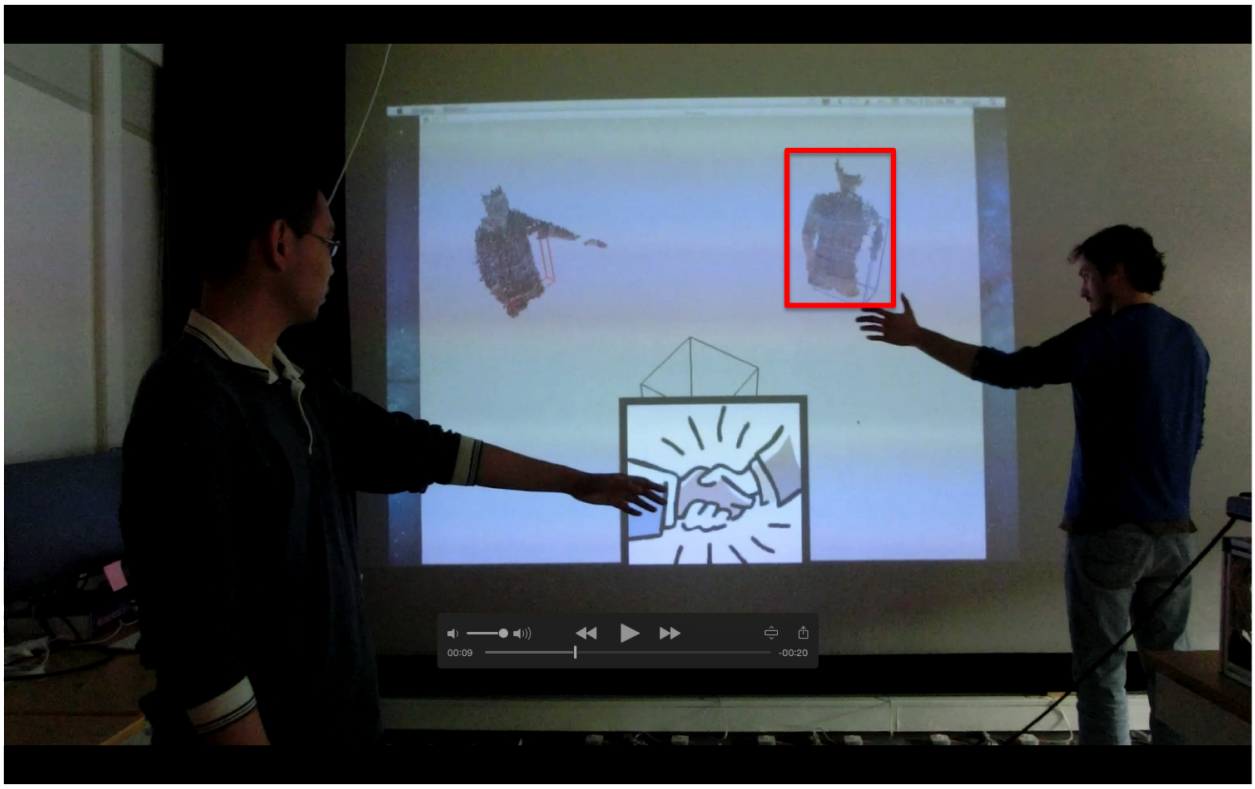
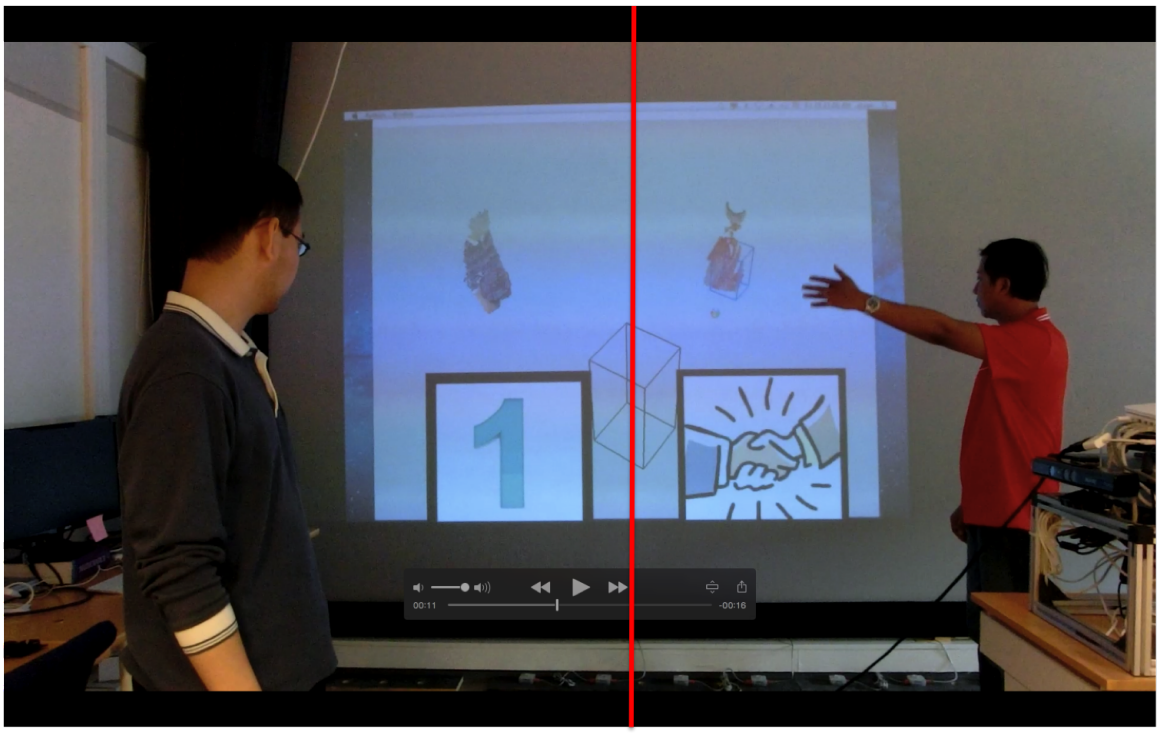


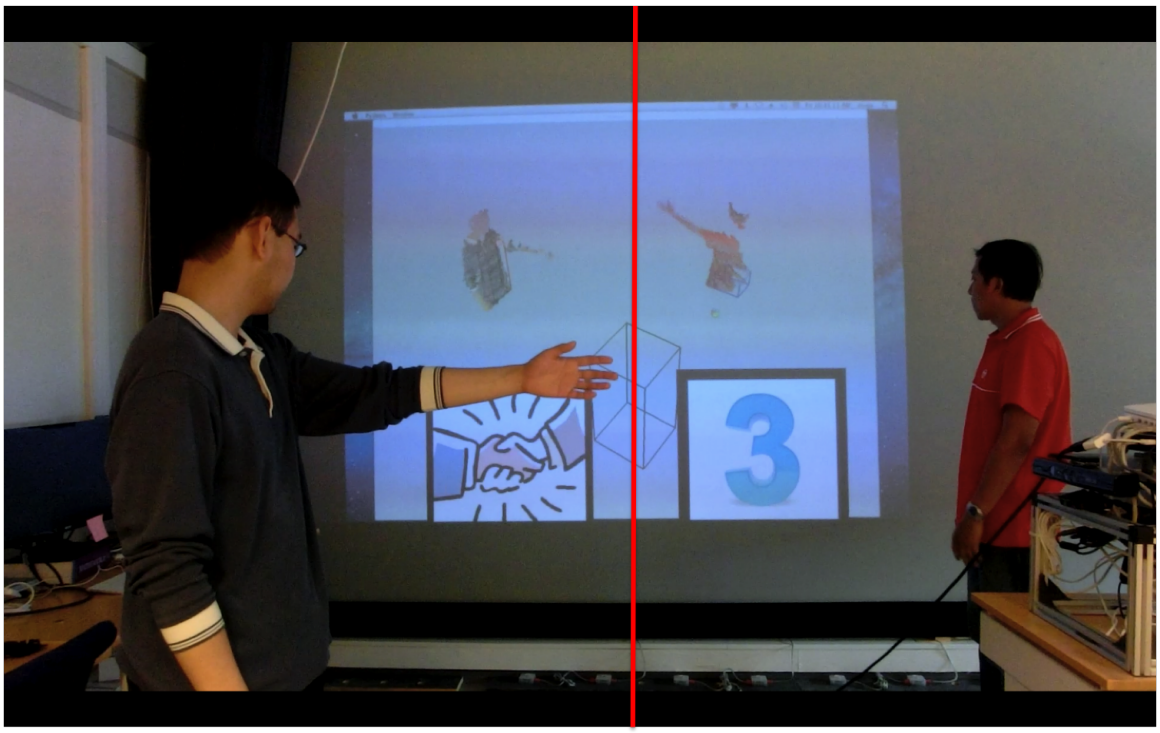
Figure 8.9: Act-By-Director approach: 1000 ms delay was added to the remote presence on the right side of the display. Although the remote presences were not synchronized, but all actors will be synchronized because they follow the same actor script.

mance will start at the same time, all actors will always be synchronized. Because remote presence will be delayed to wait until all remote presences are available, all remote presences will be synchronized. Figure 8.11 shows the local delay approach, to simulate network transmission latency, we artificially add 1000 ms delay to the remote presence on the right side. This approach will wait for all remote presences ready and the remote presences will be displayed at the same time. Figure 8.11a shows two actors start performance at the same time. All actors at every stage will be synchronized. Because of the added delay, the remote presences will not start move hands. Figure 8.11b shows after 1000 ms, the remote presences will start performance. This is because the local side on the left was equally delayed. All remote presences will be synchronized. But the actors and the remote presences will be out of sync for 1000 ms. The video example of Local-Delay approach can be viewed at [68].

5. Act-By-Wire: All stages will start performance at the same time. All actors will be synchronized. This approach will replace the delayed videos with pre-recorded videos. Therefore, all remote presences and all actors will also be synchronized.

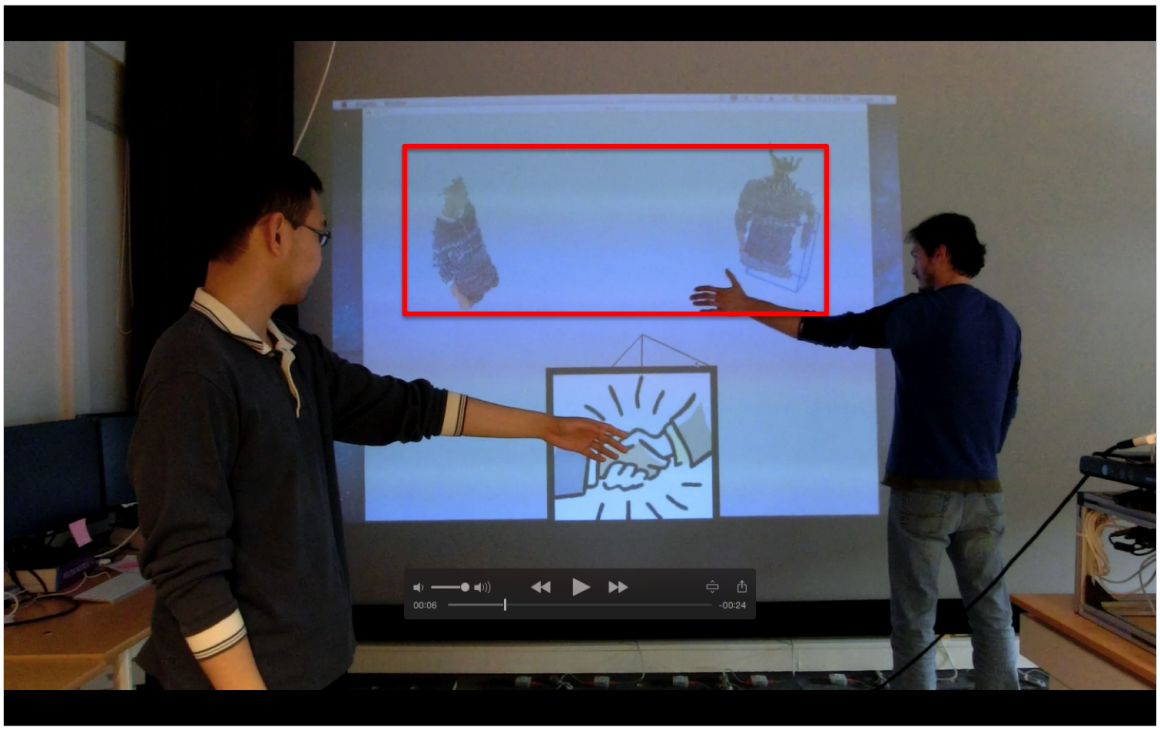


(a) The actor on the secondary stage will start performance 1000 ms earlier.

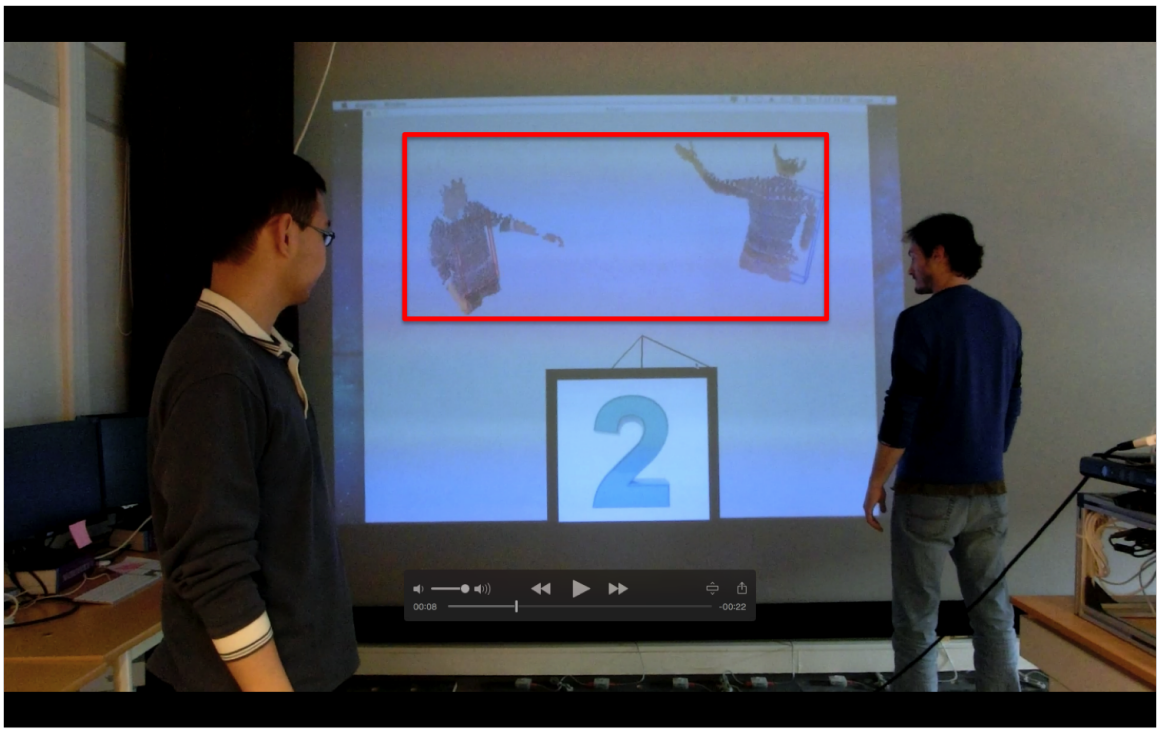


(b) The actor on the live stage starts performance. All actors and all remote presences on the live stage will be synchronized.

Figure 8.10: Live Stage approach: Actor on the left side is located on the live stage and actor on the right side is located on the secondary stage. 1000 ms delay was artificially added (simulate network latency) to the remote presence on the right side of the display. The secondary stage will start performance 1000 ms earlier than the live stage. The display shows the performance of the remote presences on the live stage.

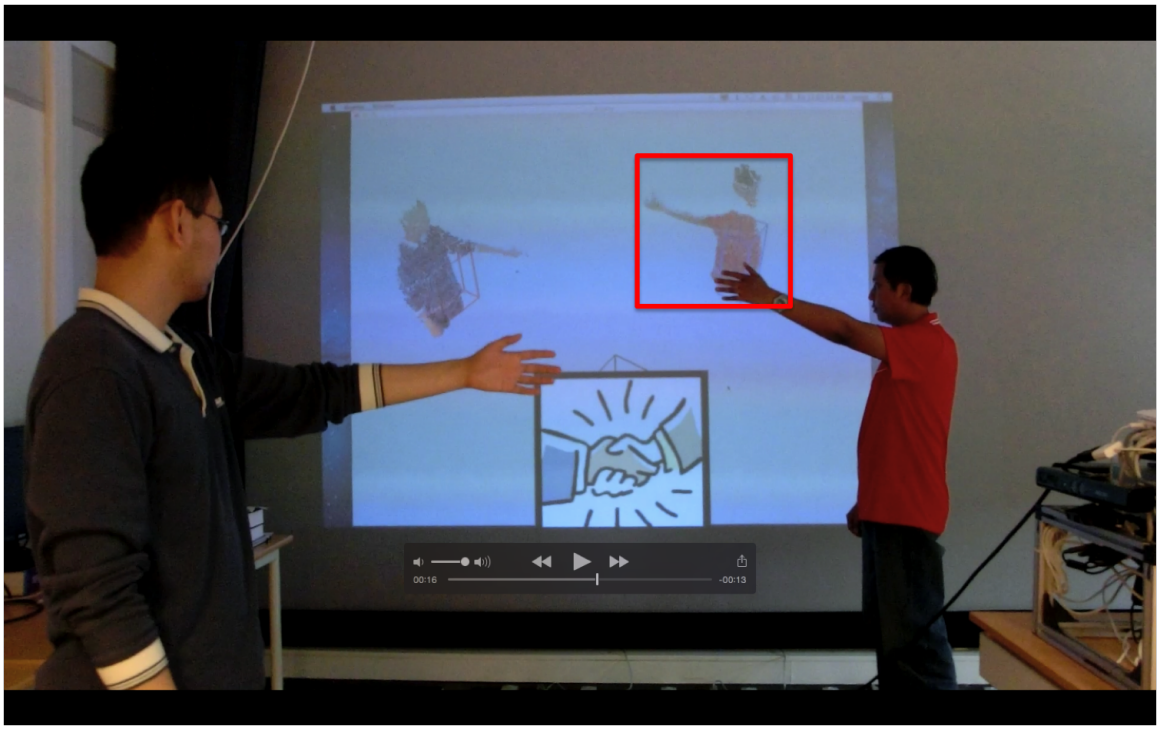


(a) Two actors start performance at the same time. All actors at every stage will be synchronized. Because of the added delay, the remote presences will not start move hands.

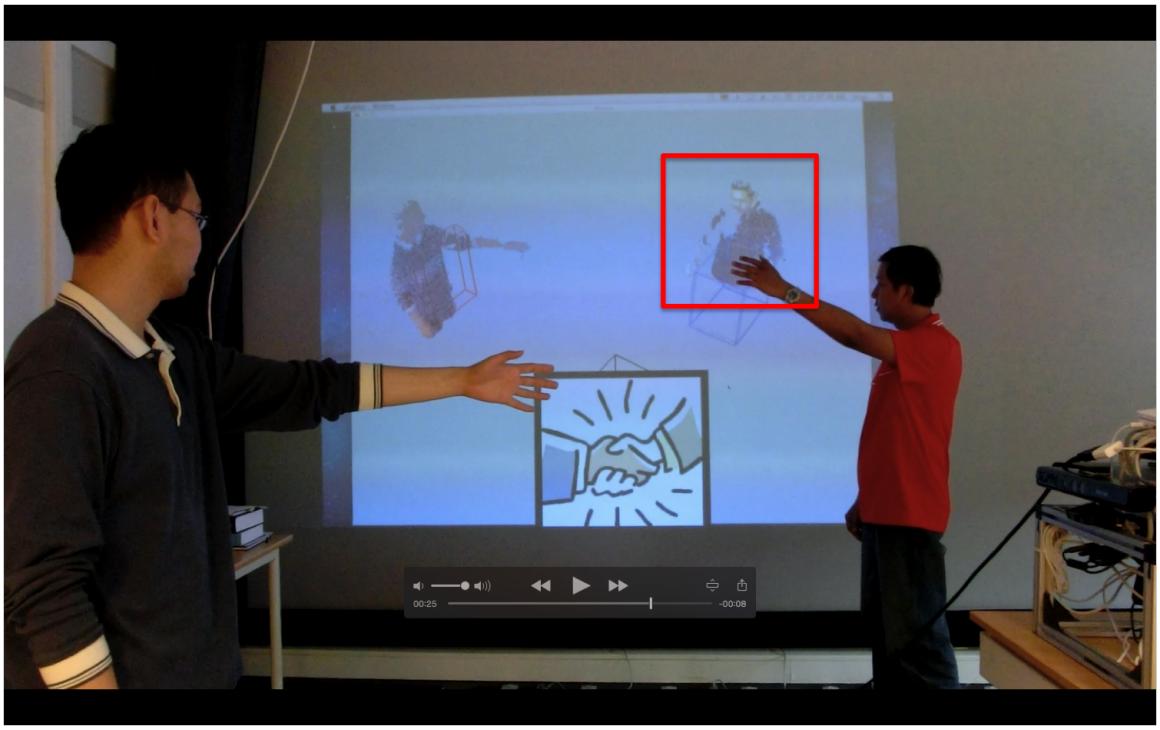


(b) After 1000 ms, the remote presences will start performance. All remote presences will be synchronized. But the actors and the remote presences will be out of sync for 1000 ms.

Figure 8.11: Local Delay approach: We artificially added 1000 ms delay (simulate network latency) to the remote presence on the right side. The local side on the left will be equally delayed for 1000 ms to wait for the data from right side arrives. This approach will wait for all remote presences ready and the remote presences will be displayed at the same time.



(a) The added delay is 200 ms, the pre-recorded video is not displayed.



(b) The added delay is 300 ms, the pre-recorded video is displayed on the red box.

Figure 8.12: The experiments to find maximum system end-to-end one-way latency for Act-By-Wire approach. The remote presence of actor on the right side will be artificially delayed. The red box indicates the remote presence of the actor will be replaced by pre-recorded videos when latency is higher than a pre-defined threshold.

Figure 8.12 shows the experiments to find maximum system end-to-end one-way latency for Act-By-Wire approach. The remote presence of actor on the right side will be artificially delayed. Figure 8.12a shows when 200 ms delay was added, the pre-recorded video is not displayed. It is still in the temporary causal synchrony. Figure 8.12b shows when 300 ms delay was added, the pre-recorded video is displayed on the red box. The approach achieves temporary causal synchrony by replace the original video with a pre-recorded video. The video example of Act-By-Wire approach can be viewed at [69] and [70].

8.5 Related Works

Compare latency values in MultiStage experiments and latency values presented in related works: Table 8.1 compares the latency values in MultiStage experiments to the latency values presented in related literature in Chapter 6.

1. Compare to DIP [6], [33] and [7]: DIP is a multi-site interaction and collaboration system for interactive musical performances. Musicians at two different locations interact with each other using media streams.

   (a) DIP found out for digital video equipment, it is hard to achieve latencies within few tens of milliseconds because the camera already needs close to 100 ms to compress picture stream. The end-to-end delay for DIP is 120-130 ms. The MultiStage system end-to-end latency is about 90-125 ms.

   (b) DIP found out a round-trip video delay of more than 230 ms makes synchronization hard for the users. In MultiStage, the human noticeable latency is about 190-225 ms. And the actor-to-actor round-trip latency is already higher than 230 ms. This means masking approaches need to be frequently applied.

   (c) DIP found out the tolerable latency for slow-paced music is much higher than for fast-paced music. This is similar in MultiStage system, where the tolerable latency for slow hand movement is much higher than rapid hand movement.

   (d) DIP found out for audio, an artificial delay of 50 ms to the remote room's audio stream was tolerable. MultiStage mainly focus on video type of interactions. The tolerable latency is 350-400 ms for rapid hand movement, and 800 ms for slow hand movement.

   (e) In DIP, with the same latency added in both rooms it became possible to easily play together with a delay of up to 65 ms. In MultiStage, the artificially delay for Local Delay approach is 300-400 ms for video.

System / Latency	Round-trip latency	System end-to-end latency	Actor-to-actor latency	Human response latency	Human noticeable latency	Human tolerable latency	Act-By-Actor	Act-By-Director	Added Local delay	Act-By-Wire
Multi-Stage	32ms	90-125ms	200-316ms	345ms	190-225ms	rapid: 350-400ms slow: 800ms	190-325ms	390-525ms	300-400ms	Any
DIP (2 sites)		120-130ms			video: 230ms	audio: 50ms			audio: 65ms	
network games	<150-200ms					250ms				
Halo						excellent: 50ms good: 150ms				
Car-racing game						50ms				
Keyboard interaction					150ms					
Mouse interaction					195ms					
Influence on network latency	<75ms									
Latency study				Exp: 700-750ms Unexp:1250ms	200ms					
RPG						500 ms				
Ref [51]		100-120ms 220-260ms				verbal: 150ms				
Tele-Haptic						120 ms			50ms, 100ms	

Table 8.1: Compare the latency values in MultiStage system to the latency values presented in related literature.

2. Several papers evaluate the impact of delays in network games. In [10] and [30], and [32], most network games rely on network end-to-end latency with in the range of 150-200 ms. According to [44], to play Halo (First-person shooter game), under 50 ms of latency is needed to have an "excellent" game experience. And a "good" game experience needs latency within 150 ms. In [45], up to 50 ms delay is not critical for car-racing game. In a graphical user interface, 150 ms delay cannot be noticed for keyboard interaction and 195 ms delay for mouse interaction. In [46], players tolerate up to 250 ms reaction time for a real-time multiplayer game. In Influences of Network Latency and Packet Loss on Consistency in Networked Racing Games [48], it found out additional delay of up to 75 ms would largely increase the inconsistency rate of causality. In [50], the threshold for first-person shooter games and racing game is 100ms, threshold for Role-playing game (RPG) and sport game is 500 ms, and threshold for Real-time strategy (RTS) game is 1000 ms.

   Network games, such as first-person shooter game and racing game are generally latency critical, where less than 100 ms delay are required. This is close to the latency value for human noticeable latency in MultiStage. For RPG and RTS game, higher latency value can be tolerated. This is similar to the human tolerable latency for slow hand movement in MultiStage.

3. In a study of latency [31], it found out for haptic feedback, the effects of performance will be noticeable at around 200 ms delay. Visual feedback on arm movements ranges from 150 to 250ms. These values are close to the human noticeable latency in our experiment.

   This paper found out the human response time for drivers in braking situation is 700-750 ms for expected brake, and 1250 ms for unexpected brake. The human response experiment in MultiStage only evaluated expected movements. But this is sufficient for MultiStage, because the actor's interaction will be expected or even pre-defined movement.

   In [71], it puts the presences of participants into a stressful virtual environment (virtual world) and measured the effect of latency. All participants are placed into the same environment with stereo visuals, sound, realistically moving things in the virtual world, and passive haptics fixtures. Half participants experienced 50 ms end-to-end latency, and half experienced 90 ms end-to-end latency. The result show the heart rate for participants in 50 ms condition was higher than participants in 90 ms condition. This is because virtual world produced more presence for the better (low latency) virtual world than the less realistic one. Skin Conductance, Reported Presence, and Reported Fear were not differing too much for the two

experiment cases. The Simulator Sickness score was higher than the previous study. They believe this is because of higher frame rate and relatively low latency. Users complained when system end-to-end latency was above 120 ms. The authors believe even latency is below 100 ms, it is still important parameters to understand the effectiveness of virtual environment.

In MultiStage, the interactions are mainly actor-to-actor interactions. The tolerable latency is much higher than the haptic actions.

4. In [11], introduced a tele-haptic system applies group synchronization control in collaborative haptic play with building blocks. It uses LL technique for group synchronization. Subjective assessment was performed to evaluate in the experiment. In the user study experiment, two users were asked to do a collaborative haptic play to build 22 blocks. After experiment, users gave a subjective assessment score: 1. Very annoying. 2. Annoying. 3. Slightly annoying. 4. Perceptible, but annoying. 5. Imperceptible. Total users are 30. When added 50 ms additional delays, the average score dropped from a little more than 4 to 3 with the standard deviation changed from 0 ms to 20 ms at an interval of 5 ms. When added additional delay 100 ms, the interaction dropped from a little more than 3 to 2 with standard deviations changed from 0 ms to 40 ms at an interval of 10 ms. However, with more than two participants seeing significantly different latencies, the fairness cannot be maintained for all computers.

   In MultiStage, the satisfactory latency for LL approach is 390-525 ms and the added delay is 300-400 ms. A more detailed experiment could be conducted with more users perform to user study, add standard deviation into the artificial delay, and let the user evaluate the performance afterwards. The masking approaches done by MultiStage also consider situation when there are more than two stages. However, the performance needs to be evaluated.

5. In [51], a comparison is made between the end-to-end latency of an immersive virtual environment and a videoconferencing system. The tolerable latency for verbal communication was found to be 150 ms for teleconferencing system. The end-to-end latency from when the person moved an arm until it was reflected through the system was measured to be 100-120 ms for the teleconferencing system, and 220-260 ms for the virtual environment when the avatar for the user had been preloaded.

   The end-to-end latency for teleconferencing system is close to end-to-end latency in MultiStage. The end-to-end latency for virtual environment system is higher then

end-to-end latency in MultiStage. This is because the virtual environment used a more complicated model and it takes more time to for system to finish processing. If more complicated processing added to MultiStage in the future, this will most likely to increase MultiStage system latency. And the noticeable, tolerable, and satisfactory latency values will be decreased.

6. Latency hiding techniques such as DR and LL are widely used in network games. For example, in [45], it combines DR and LL techniques. In the experiments they evaluated their algorithms, or evaluated how fairness and consistency are maintained. User studies on delays were not performed. In the experiment, they set a fixed lag to 300 ms, and network transmission delays to 100 ms, 300 ms, and 500 ms. Then they set fixed network transmission delay to 800 ms, and lags to 100 ms, 300 ms, and 500 ms. Then they compared between DR and DR with LL, and found out the latter can decrease more inconsistency and achieve better fairness.

   The latency experiments in MultiStage are informal user studies, the purpose is to find out the acceptable latency values to allow user interactions. However, it would be worth to evaluate MultiStage performance by conducting the above experiment.

8.6 Discussion

We have divided the MultiStage system into a local side and a global side. This allows for a clear separation of concerns between what can be taken care of locally at each stage, and what must be done system-wide to bind together stages and do system-wide actions.

The latencies between the subsystems are a few ms in the LAN and a function of the distance in the Internet case. A user can notice the difference between actions when about 200 ms delay was added. We did not find this latency to be significant in our experiment.

The actor-to-actor latency is always higher than what is tolerable for tight interactions. A user can notice the difference between actions when about 200 ms delay was added. This implies that we need to start approaches to mask the effects of delays for tight interaction. A user can tolerate up to 800 ms in looser interactions.

The experiments measured the subjective metrics. No formal user studies were performed. The determination of thresholds was done naively based on the opinion of a few persons observing actors and remote presences.

The experiments used simple movements by an actor, primarily hand and arm movements. The results can be expected to be different for other actions done by actors, such as body rotation, jumping, and dancing.

A handshake includes both loose and tight interaction. We want two users to shake hands at the same time, but at what point in time they put their hands down is not a big deal. It is not so simple to make it look like the real thing.

To blend in prerecorded video, we need to know how long each image has been delayed in order to determine how long a delay we must blend into the prerecorded video and also when we should stop the prerecorded video. In experiments, we did this by add artificial delay to the remote presence and subjectively decide the value until we saw an obvious movement difference between the real user and his remote presence.

In the experiments on when to start and stop masking, we artificially added delay to the remote presence and found a critical value 280 ms. That is the delay between when the image been captured until it has been received by the Collaboration system. The total system end-to-end latency will be larger than 280 ms because it takes time for the camera to do detection, for the Collaboration system to pass the image to the Remote Presence system, and projector also needs time to display the image. The critical value was decided based on the tolerable latency for rapid hand movement. For slow hand movement, where larger delay will be tolerable, the critical value will be larger.

We subjectively decided to let the Collaboration system measure how many packets arrive late in the past 3 second. However, this threshold can be changed to a larger or smaller value. In the experiments, we stop masking if 35-40% packets arrive late in the past 3 second. We also found out, a longer period will results in longer time switch back from pre-recorded data streams to live data streams, even when the movements of remote presences of local and remote actors are close enough in time. Therefore, for a longer the period the percentage can be adjust to a lower value to switch back live data streams faster. For a shorter period with the pre-determined percentage, it will increase the likelihood of switching back and forth between the pre-recoded data streams and live data streams. In this case, the percentage can be adjusted to a higher value to reduce the likelihood of switching back and forth.

8.7 Conclusions

MultiStage is a prototype created to better understand what the issues are. The issues include:

1. Bandwidth: Having stages across the Internet is a challenge for the system because traffic load, failures and outages are mostly unknown before they happen. We have documented that the system scales to at least three stages with a total of at least 12 incoming and 36 outgoing data streams.

With regards to bandwidth, the location of the distribution server is presently not critical. This may change if the data streams grow in size and number. However, if the global analyzer and distribution subsystems are located on computers on the same LAN as one or more of the stages, the Internet traffic is significantly reduced. This will penalize the other stages but could be useful for a performance with local audiences or where synchronized interactions are mostly among actors on the local stages.

2. Latency: Based on informal use of the system, we found that even 800 ms of delay while interacting using slow movements was in some cases tolerable. However, the general case seems to be that delays above 200 ms are noticeable when having remote presences based on vision and visualizations. We found that an actor-to-actor round-trip delay of above 200 ms is frequently the case, and masking is consequently frequently needed.

3. Masking approaches: The masking approaches we developed and did performance measurements on, demanded insignificantly more resources (CPU usage) and delays than not using them, and can even in the most complicated case when using Act-By-Wire approach, be switched in and out with insignificant delays and resources. However, masking approaches where significant processing is needed will give raise to much more CPU usage.

The subsystems and bindings between subsystems make for a complex actor collaboration system. While good programming practices will reduce the number of failures, a simpler system will provide for a higher probability of avoiding failures immediately before and during a performance.

The built-in online monitoring of the state of the individual components of the system is important to discover where problems happen, and to help in fixing them. During our experiments of MultiStage, the monitoring system has helped us to identify if all stages are running properly. Online performance monitoring is critical for discovering delays long enough so that the system can try to mask their effects.

Even if the system can do temporal causal synchrony and mask away the effects of delays, it is not yet clear how practical the system is in use. While we have not done formal user studies exploring the system capabilities with actors needing to tightly coordinate their movements, we have documented the performance limits of the MultiStage system. This provides for a sound prototype platform for experiments in a context of distributed performances with real actors.

Chapter 9

pVD - Personal Video Distribution

The pVD system uses the decoupled producer and consumer model. It is designed for video sharing between a single user's computers. Each user's computer is a producer that can publish videos to the pVD global side. Each user's computer is also a consumer, subscribing to videos from the pVD global side. Video playback can be moved from one subscribing computer to another computer. The global side of pVD is a centralized server handles video distribution for all of the user's computers. This chapter describes the architecture, design and implementation of pVD.

9.1 Introduction

Figure 9.1 shows that users today use multiple personal computers, including both mobile devices and larger displays. Many of these computers will have cameras that can be used to produce live video streams, and will have significant storage for videos. Live video from a camera connected to a computer can easily be watched on the same computer. This is also the case for videos stored on the computer. However, it is more cumbersome to do a smooth handover and watch video produced and stored at one of the user's computers at the others. It is also cumbersome to locate a video across computers because there is not a shared video name space. Consequently, different videos at different computers can have the same name.

Existing industry approaches typically rely on a third party to let a user watch cameras and videos across computers. At a minimum, a log-in to a subscription service is needed. As well as being dependent on third-party computers, an external network giving access to the Internet is also needed even when all video producing and consuming computers are local, for example, at a user's home. This increases the probability for failures as well as cost and bandwidth usage.

Security and privacy are also issues users are concerned about when relying on third-

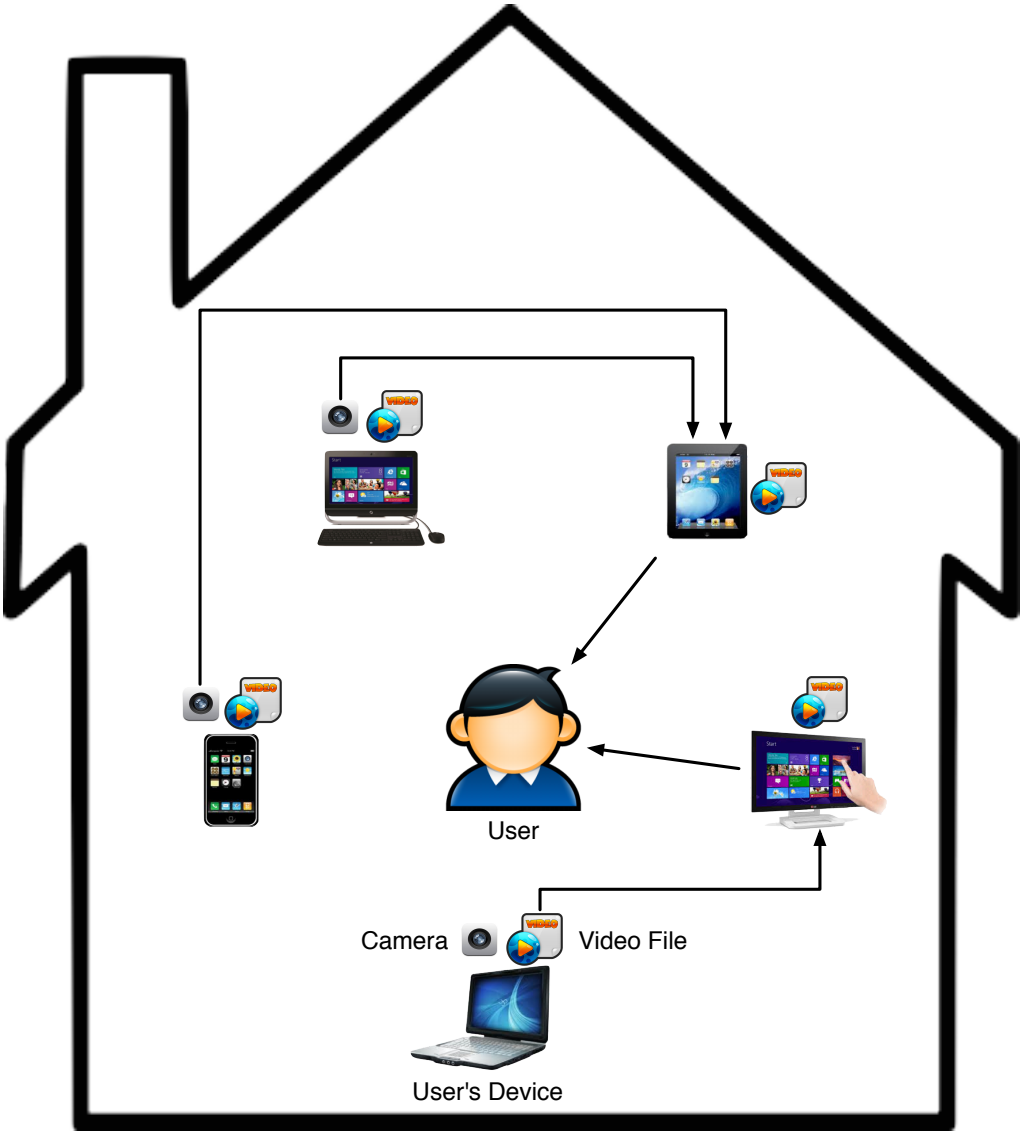


Figure 9.1: The complicated life of a user.

party services to store and service data [72], [73]. In [74] it is documented that people are more concerned about privacy on cell phones than laptops.

We report on the architecture, design and implementation of the pVD prototype, allowing computers belonging to a single user to subscribe to cameras and videos from each other, see figure 9.2. The system allows the live video from any camera to be viewed at any computer. When a stored video is played back or stored on one of the computers, it can be picked up by any of the computers. Playback can be started from where in the video the user last stopped watching it. The pVD system does not rely on a third-party service at all, using only a user's computers. When all computers are inside the same domain, for example, at home, no Internet access is needed.

The usage model assumes that a user has physical access to all the computers. To

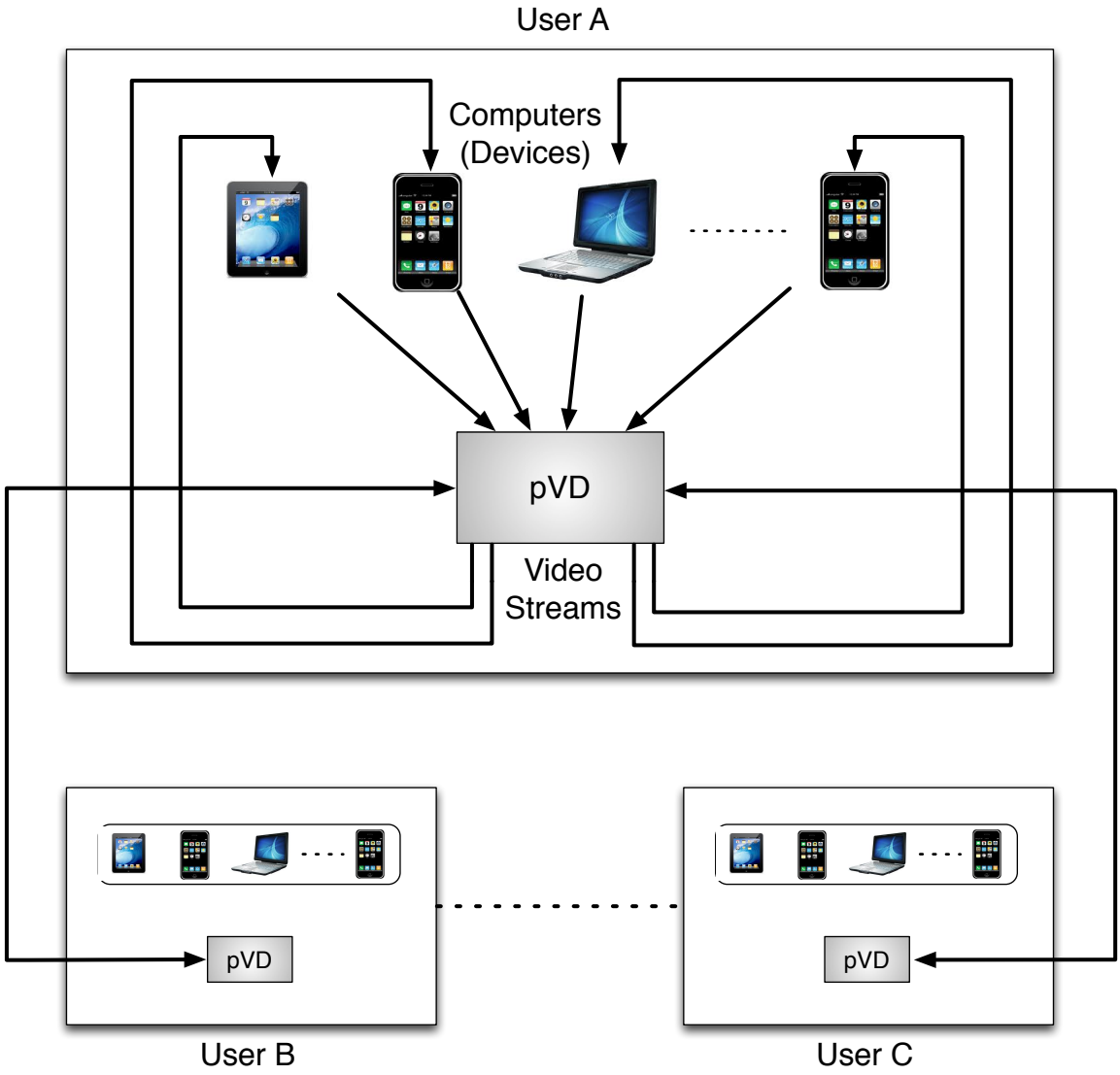


Figure 9.2: The idea of the Personal Video Distribution (pVD).

watch, for example, a smartphone's camera on a tablet, the user starts the pVD smartphone app and selects the camera as a video source. The app then starts streaming the video to the pVD system. On the tablet, the user starts the pVD app, inputs the smartphone's name and the name of the live video, and a subscription is sent to the pVD system. The pVD app on the tablet now waits for videos to arrive according to the subscriptions. The pVD system matches incoming video streams with subscriptions, and streams the video to the tablet. While a single video can easily be streamed in and out of some smart phones and tablets, streaming of multiple videos is more easily supported using more powerful computers such as laptops and PCs.

The prototype is presently functional for a single user with multiple computers. We intend to extend the prototype to support multiple users on a single pVD system, for

example, a family, in the future. We briefly describe how multiple users with separate pVD systems can share cameras and videos without relying on a third-party service.

Our contribution is to document a flexible and simple way to switch videos between a single user's computers by using only these computers, and without relying on a third-party (cloud) service at all. We document how to do this through the architecture, design, and implementation of a working prototype, and its performance characteristics. Several experiments have been conducted to measure how fast the pVD system responds to subscription requests, the CPU utilization of the part of the pVD system used by all the computers, and how pVD scales when the number of videos and computers increase.

9.2 Related Literature

There are many existing live video streaming services.

Content Distribution Network or Content Delivery Network (CDN) (e.g., Akamai [75] and Bootstrap CDN [76]) serves content including streaming media to end-users with high availability and high performance. P2P (e.g., Bittorrent [77]) is a decentralized network architecture where each client node is acting as both a resource supplier and consumer.

PPStream [78] and PPTV [79] are systems for video distribution over the Internet using a combination of client/server and P2P approaches for the distribution of videos. They maintain the state of a user on the user's computer, including which videos the user watched and where the user stopped the playback of a video. The user can later start a video from where it was stopped. Contrary to pVD, these systems are large scale, sharing many videos between many users, and they rely on every user having Internet access. Also, in pVD, a computer has the complete video so no P2P collaboration is needed. This reduces the complexity of the system.

YouTube is the world's largest video sharing website where people can upload, view, and share videos. Live streaming is possible through services such as YouTube Live Streaming Events and Google+Hangouts. These systems store and share many videos between many users, while pVD shares just a few videos between a single user's computers. Users are dependent upon YouTube and Google as third parties outside of the users' control.

LiveCast [80] and Qik [81] enable live video streaming from users' mobile and other devices to any users or friends connected to the Web. LiveCast is large-scale with many users. It is feature rich, and meant to be used across the Internet. Users are not dependent upon a third party except their own organization or company. Qik is also on a large-scale, enabling sharing between many users. The user must rely on Qik as a third party and store videos with Qik.

Universal Plug and Play (UPnP) [82] is a network protocol that allows devices to discover other devices and share data between them. The Digital Living Network Alliance (DLNA) [83] uses UPnP for media management and media sharing between devices. Windows, Mac, Linux, and Android also use the UPnP protocol to enable media sharing between devices. DLNA systems typically apply a media server and media players. Miracast [84] allows wireless streaming of videos from user's device to a big screen, such as television. User can also real-time screen sharing from one device to another. In contrast, in pVD, every computer is both a media server and a media player.

Apple AirPlay [85] allows limited wireless streaming between Apple computers. The computers must be on the same subnet. While a video stream stored locally on one computer can be sent to another local computer, all computers must log into and interact with a third party, an Apple iTunes account.

A mobile live video learning system used for large-scale learning is described in [86]. Students can either attend a course in person or watch live and stored video streams sent from a server to mobile devices. The system includes a central server, classrooms, and mobile devices. The server receives and records live streams from classrooms. The system does not maintain a user's video state to let play back resume at another computer.

Tele-TASK [87] is a tool for recording lectures and what happened at the presenter's computer, including presentation slides and software demonstrations. The presenter's computer desktop video and video of the presenter's are merged into a picture, encoded to MPEG-4 video, and saved to a file. Users with mobile devices can watch the lectures anywhere. Tele-TASK does not stream live videos and there's no live interaction between users.

CloudPP [88] is a Cloud-Based P2P Live Video Streaming Platform. It uses third-party cloud servers to construct a video delivery platform. The system dynamically allocates resources to save bandwidth and is able to let a very large number of clients receive live streams at the same time. CloudPP uses cloud services, resulting in privacy issues.

LiveShift [89] streams both live and stored videos. Live videos are streamed through a P2P network and peers store received videos for future playback. A user can view a stored video without local recording, and jump over boring parts to catch up the live video.

Eunsam Kim et al. [90], proposed an on-demand TV service architecture for networked Personal Video Recorders (PVRs). This design reduces interactive operation response time and saves network bandwidth. The architecture includes origin servers, cache servers, and Networked PVRs. The system serves both live videos and stored videos for playback.

The above three systems do not maintain a user's state, so the user cannot continue watching the same video from where it left off or switch to another device to continue watching the same video. They are also large-scale systems, requiring infrastructure.

Mobicast [91] is a mobile live video streaming system. The system includes a mobile client and a cloud service. Multiple users stream the same event from their devices. The streams can be stitched together, or the stream that has the best viewing angle is selected to provide a better collective viewing effect to viewers. If two users stream the same view, one of the streams can be stopped and later resumed to conserve battery power on the mobile device.

MobiSNA [92] is a mobile video social networking application. It allows users to share videos at anytime from anywhere, lets users easily find videos of interest to them, and the system also intends to enable users to manage videos in a personalized way for their own purposes. The system is based on client-server architecture. Each client runs on the mobile device with a camera, and it uploads or retrieves data from the server. The server side manages, stores and streams videos to the client side.

Video Cloud [93] proposed a cloud download scheme in which a user sends a request to the cloud, and the cloud downloads video from the Internet and caches the video. The user can get the video at any time and from any place.

MyVideos [94] is a prototype system used to manage home digital videos. The functionalities of the system include video segmentation, summarization, grouping, editing, video playback, and highlight.

The systems mentioned above share to a large extent some characteristics. They typically make a client dependent upon a third party outside of the client's control. They are intended for many clients. We expect them to be rather complicated because they need P2P and other approaches to maintain good performance when the number of clients grows. While they all allow a user to play back videos, these videos are in most cases not meant to be on the user's computers. The ability to switch a live camera feed between a user's computers is only available in a few of the systems.

A significant characteristic of pVD is that a single user's computers subscribe to video streams from each other. Multiple subscriptions can be set up. However, a producer of a stream and the stream's subscribers do not run their computers at the same time. When a stream starts streaming it can be picked up, and when no subscribers are running their computers, the stream will go to the pVD system where it is buffered until a subscriber becomes present. pVD can also do handover of video streams between computers, letting a video start playing again from where it was stopped at another computer. The state of all videos is stored at and handled by the pVD global side server, and not by the pVD local side computers.

9.3 Architecture

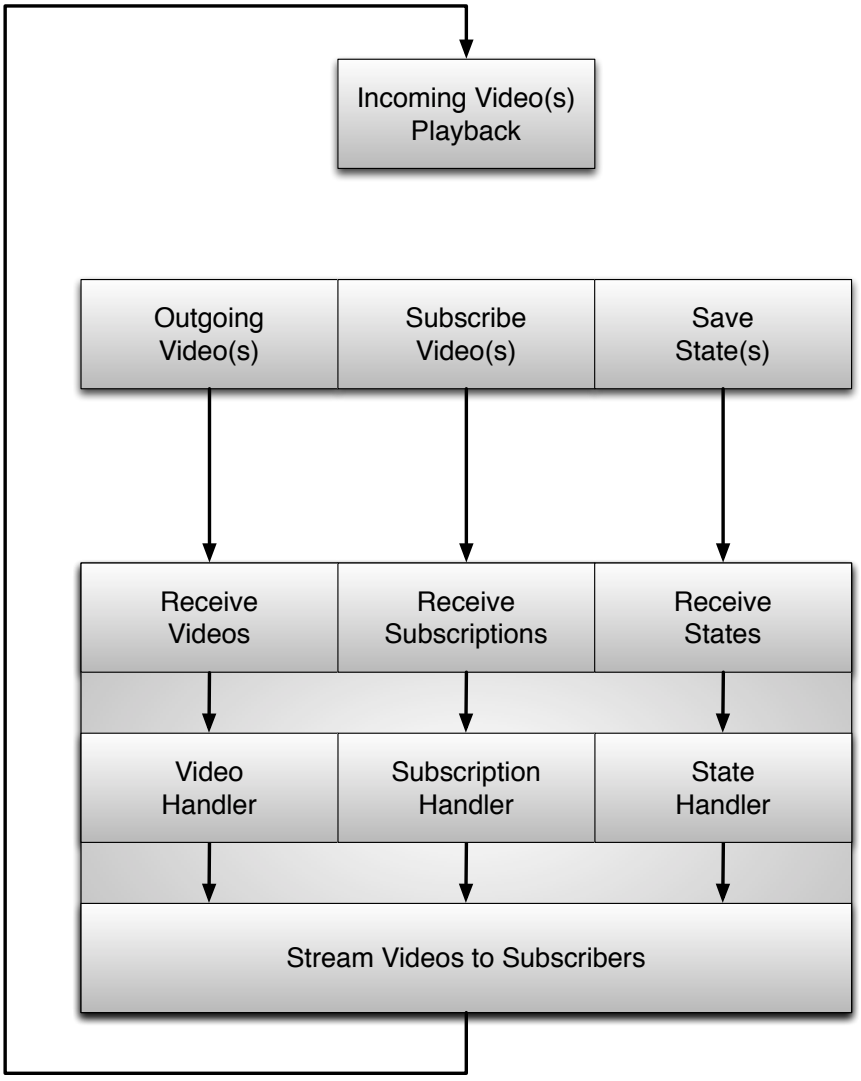


Figure 9.3: The architecture of pVD.

The architecture, shown in Figure 9.3, comprises functionality for handling incoming and outgoing videos between a single user's computers. A subscription model for videos is used. To display a camera feed or a video at a computer, the computer must subscribe to the video. pVD allows a computer to send subscriptions to it. pVD will receive all subscriptions and use them to manage the switching between computers.

A central part of the architecture is the distribution of live and stored videos to individual computers according to the subscriptions and the state of the videos. The functionality defined by the architecture includes streaming of outgoing videos, receiving incoming videos, buffering of video streams, and playback of videos.

Videos are handed over between computers by saving the current video playback state,

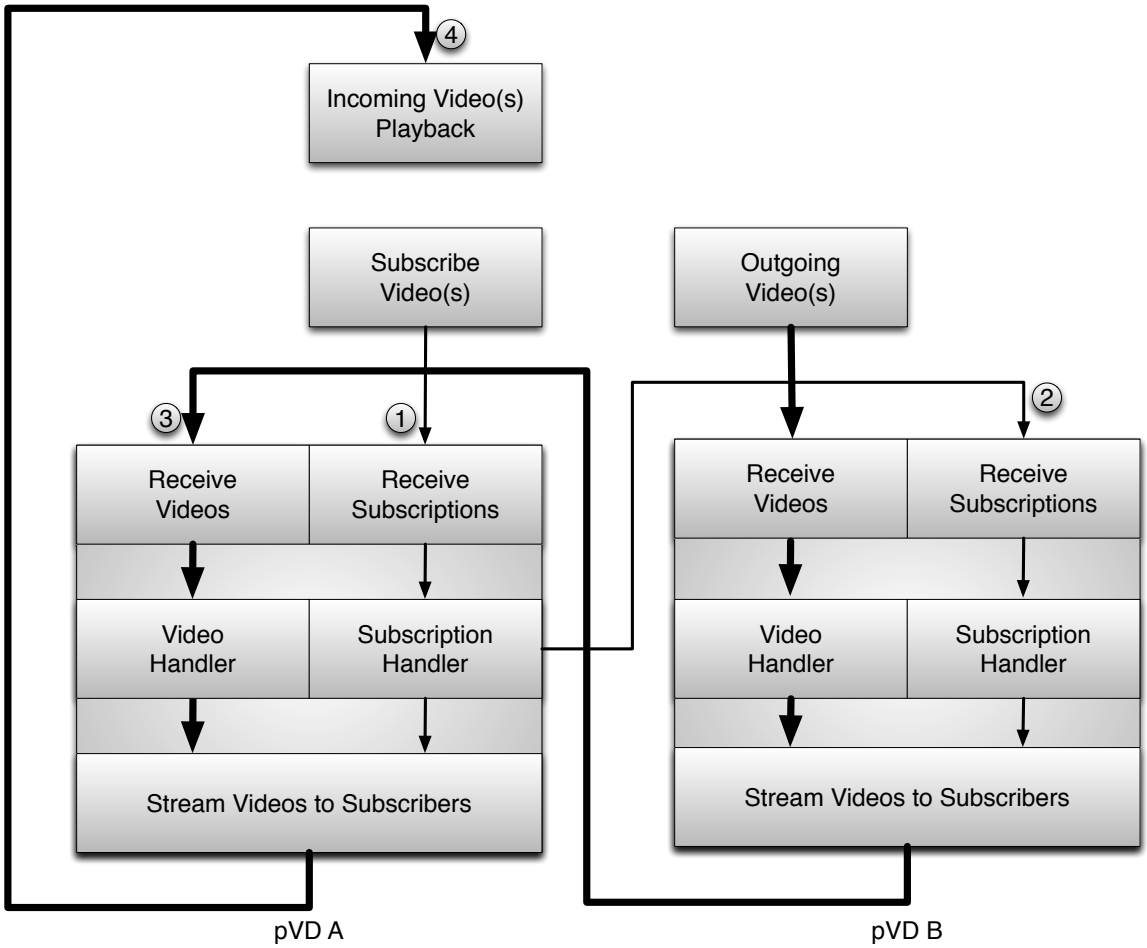


Figure 9.4: The communication between multiple pVDs.

including where in the video the user last stopped watching, to the pVD. When the video is streamed to a new computer, the state information is used to let a user continue watching the video on the new computer from where it left off at the other computer.

The architecture allows two users to provide videos from and to each other's computers as shown in Figure 9.4. User A will call user B through some channel (e.g., telephone or chat) and get the address (present prototype uses the IP address) to the global side pVD of user B (call it pVD B). User A will indicate this address when starting a subscription to a video on one of user B's computers. The subscription handler of pVD A uses the address to contact pVD B and registers the subscription with itself as the receiver. When user B starts streaming a video, pVD B will look at its subscription data and send the stream onwards to pVD A. In turn, pVD A will forward the video in a normal fashion to user A's computer.

To aid privacy, a user must, from the user interface of the computer with the video, explicitly acknowledge the streaming of the video to another user's computer each time streaming is started.

9.4 Design and Implementation

The design of the system is shown in Figure 9.5. The system is separated into a local and a global side. The local side executes on each device. It comprises a user interface that sends requests to start and stop subscriptions, starts playback of incoming live and stored videos from other computers, streams outgoing live videos from cameras, and sends stored videos. It also keeps track of the necessary state for handing over videos between computers.

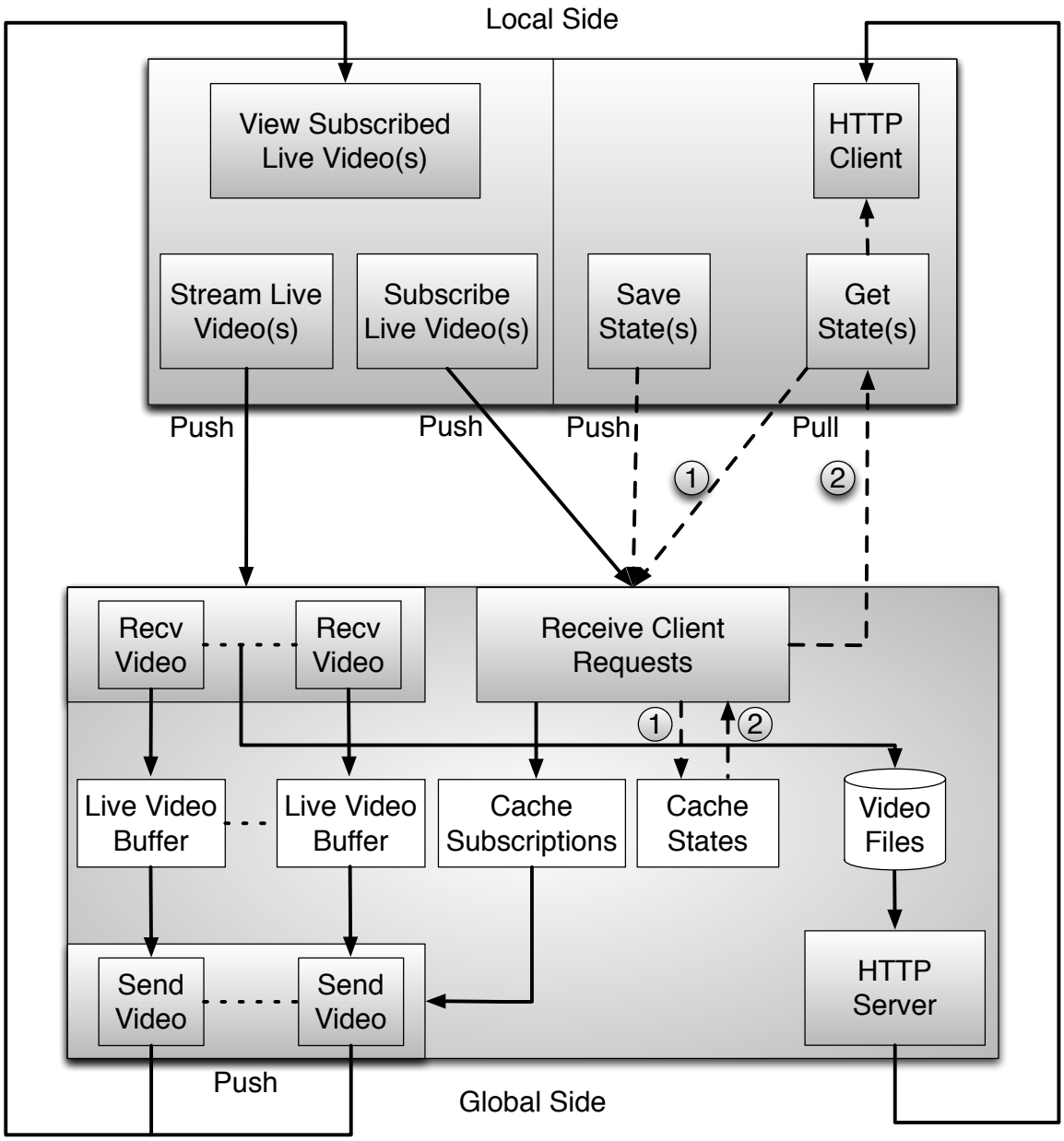


Figure 9.5: The design of pVD.

The global side executes on one of the user's computers, typically a PC or laptop,

with the necessary resources to serve or communicate with the other devices (bandwidth and sufficient storage and memory). It is assumed to be always on and accessible to the other computers.

The local side pushes videos, subscriptions, and state data to the global side. The global side receives incoming videos and data, and pushes out video streams to computers with subscriptions. The global side manages information about subscriptions and the state of live and stored videos.

The local side is concurrent to the degree supported by the operating system running on the computer. Some smartphone operating systems may have limited support for concurrency. The global side is designed as a concurrent system executing on a general-purpose operating system. This is done to make it simpler and more flexible, and to be able to benefit, with regards to performance, from multiple cores.

Each frame of a live video includes the sending computer ID, video ID, frame counter, and a timestamp for when the frame was captured. A subscription message indicates the user ID, computer ID of the viewer, and video ID. There are three state related messages to save, get, and remove where (frame number) in a given video a specific user and computer are at.

Video files are served by an HTTP server at the global side. The system was implemented using Python and Python OpenCV. It runs on Linux and Mac OS X.

9.5 Evaluation

To characterize the performance of pVD, a set of experiments was conducted using ten computers. Figure 9.6 shows the hardware configuration in the experiments. All computers were 2011-2012 Mac minis running at 2.5 or 2.7 GHz, and with 4 or 8 GB of memory. The global side computer was always connected to a Gigabit Ethernet, and local side computers were either connected to the same Gigabit Ethernet or to a Wi-Fi wireless network. There are 1-3 local side computers receiving videos and there are 2-6 computers sending out videos.

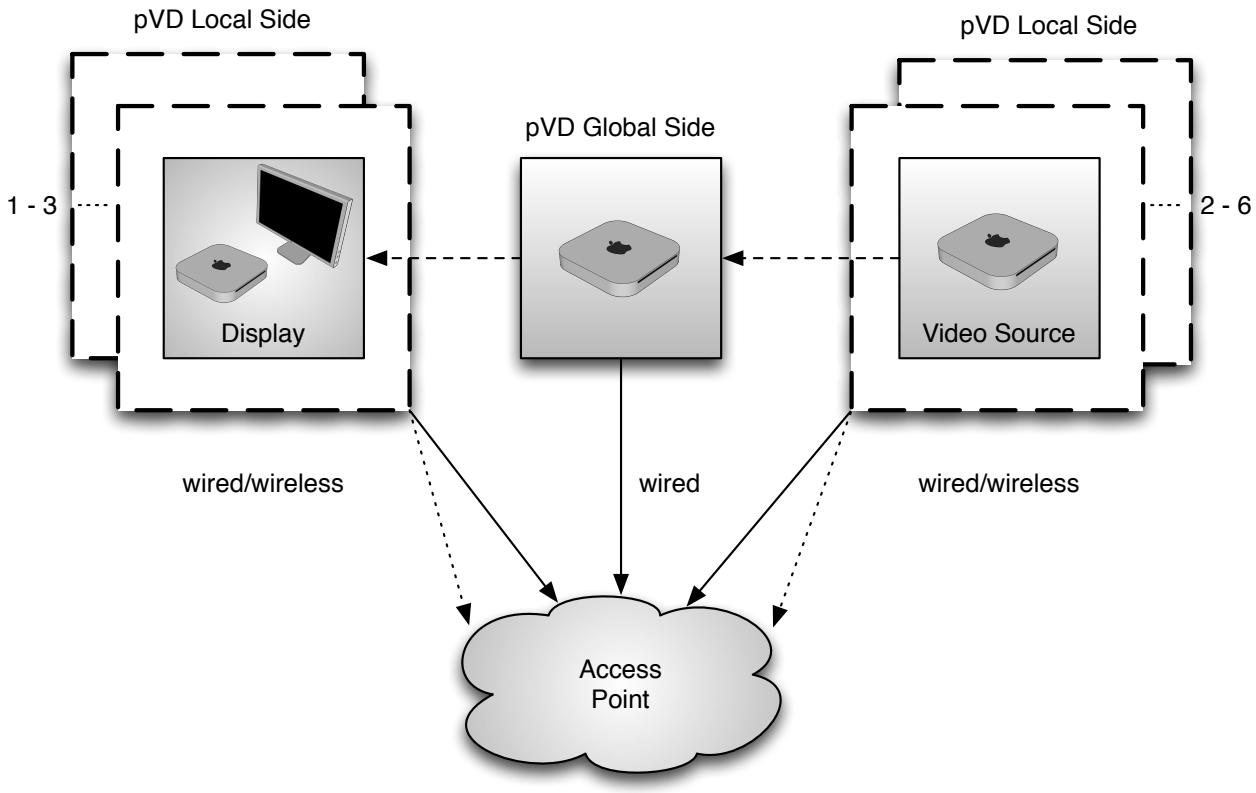


Figure 9.6: The hardware configuration

9.5.1 Experiments on wired Gigabit Ethernet

All computers were connected by wire to the same 1 Gbit/s Ethernet switch. Six computers were used to represent a user's computers having videos and cameras. These executed the local side. One computer was used to run the global side pVD. Three computers were used to represent local side viewers that subscribed to the produced video streams.

We measured the **subscribe round-trip latency**: the time it took from when the local pVD sent a request to start a subscription until it was received and processed by the global side pVD and an acknowledgement was received back at the local pVD. Figure 9.7 shows how we measured the subscription round-trip latency. The subscribing computer records the time when it requests a subscription, and the time when an acknowledgement arrives, and calculates the delay. We increased the number of computers from one to six. Each computer sent one or ten subscription requests. The subscription experiment uses TCP as the transport protocol.

We measured the **video end-to-end latency**: the delay from when something happens in front of the camera at one computer until it is visible on the display at a subscribing computer. To measure the video end-to-end latency, we set up one local pVD computer with a camera capturing a user, and another local pVD computer subscribing

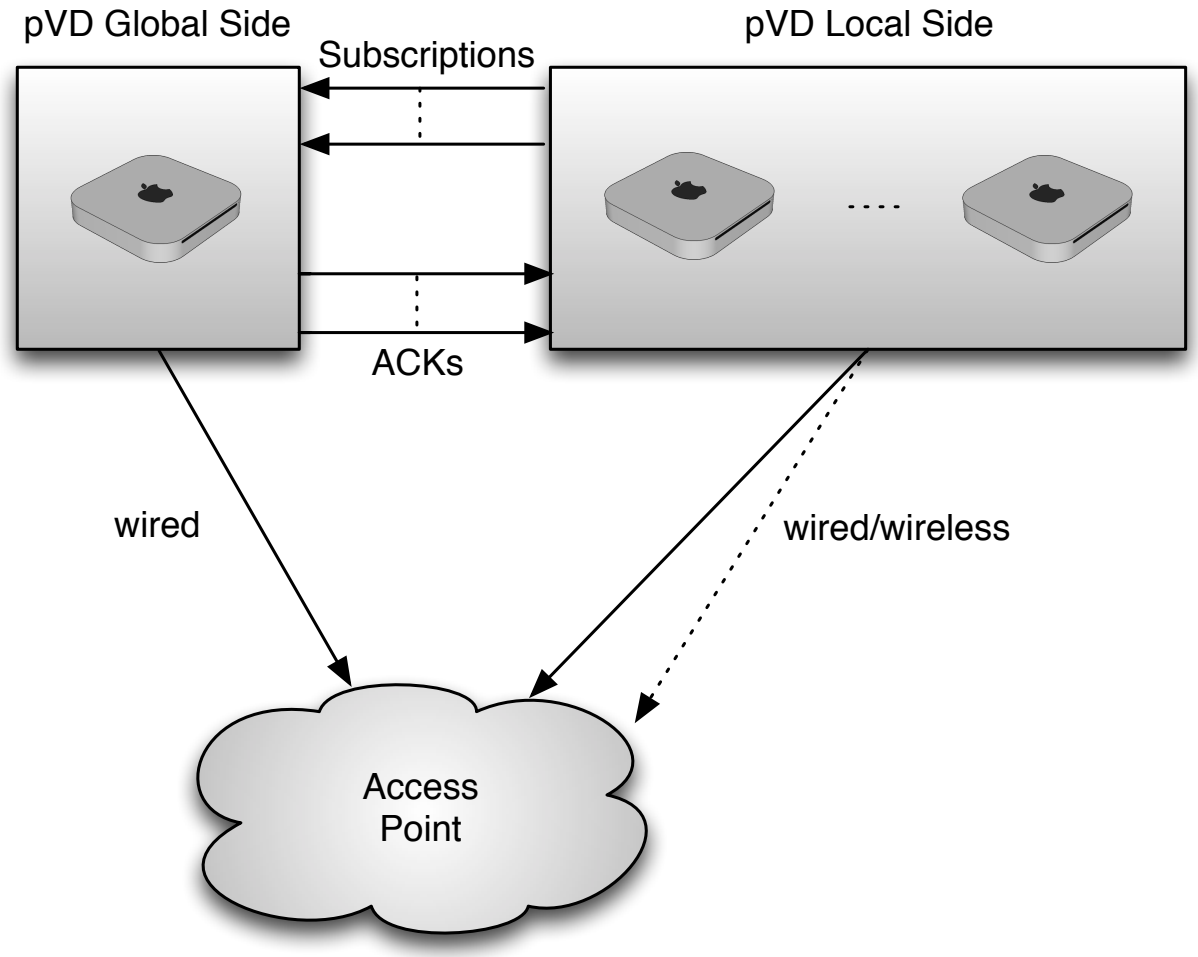


Figure 9.7: The subscription round-trip latency.

to the camera and displaying the camera output on a display. We arranged the user and the display such that a high frame rate video camera (CASIO EX-ZR200) could record both on the same video. We recorded several videos of the user and the display. We then counted frames to see how many frames it took from when the user initiated a movement until the movement became visible at the display.

We measured **resource usage** of the global pVD computer and the participating local pVD computers. Using the Python psutil module [13], we measured the CPU utilization at the global pVD computer, and the incoming and outgoing network traffic for both it and each of the other computers.

Videos were represented by point clouds from two Microsoft Kinect cameras per local pVD computer. These were sent using UDP messages, resulting in about 13.5 Mbit/s per camera, or about 26.7 Mbit/s of data from each local pVD computer. This is equivalent to about four High-Definition (HD) videos (4 to 8 Mbit/s) from each computer to the global pVD computer. In the results, we report the number of HD stream equivalents.

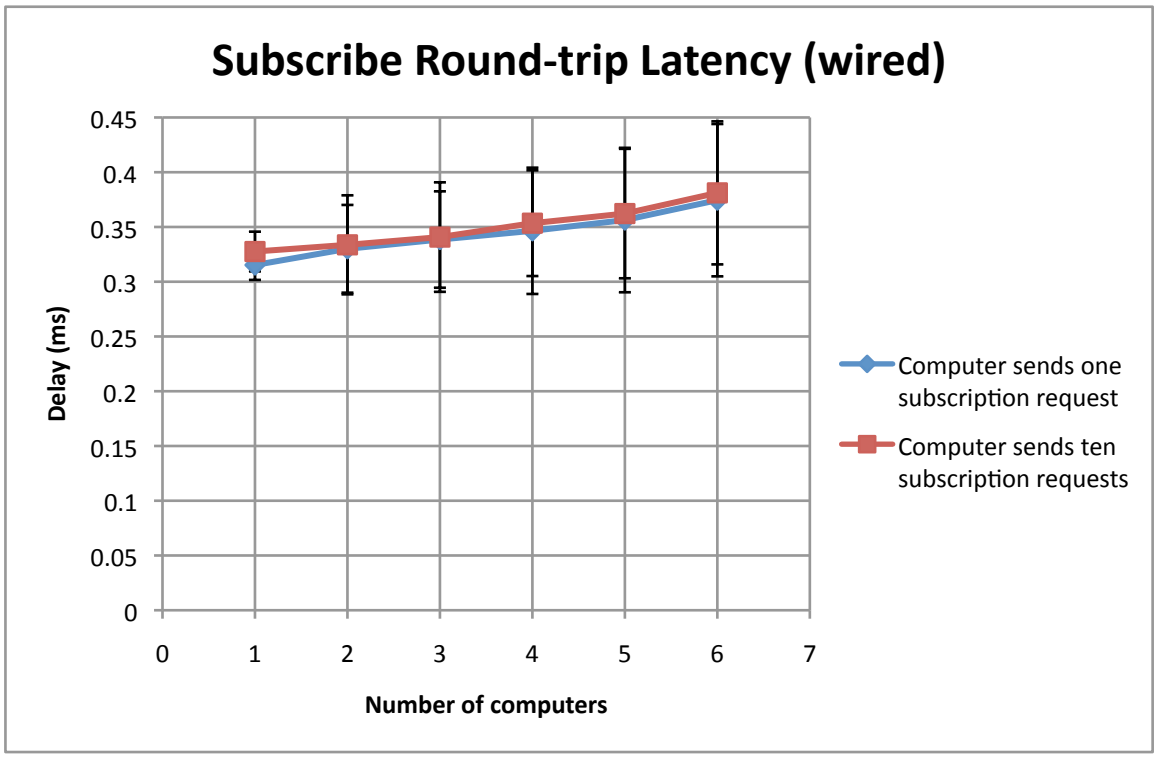


Figure 9.8: Subscribe round-trip latency when all computers are connected to a Gigabit wired Ethernet. There is one subscriber per local pVD computer. Each subscriber sends one request in the first experiment and ten requests in the second experiment.

To simulate local pVD viewers, we used 3 Mac minis as viewers, with each viewer receiving a copy of every stream sent to the global pVD computer. We gradually increased the number of camera computers from two to six, increasing the number of video streams to the global pVD computer and the number of outgoing streams from the global pVD computer.

Figure 9.8 shows the subscription round-trip latency. The round-trip latency is about 315 μs for one computer with one subscription request, and about 380 μs for six computers with ten subscription requests each. This is an insignificant increase. We conclude that the subscription mechanism in the global pVD scales well with the number of computers and videos we expect a user to have.

The video end-to-end latency was between 90 and 125 ms. We conclude that the video end-to-end latency is low enough to allow interactive use. In a study of latency [29], a 100 ms delay was noticeable by humans but found to be acceptable. More than 200 ms delay made interaction uncomfortable. The sum of the subscribe message latency and the end-to-end latency is less than 200 ms. While at the borderline, the pVD system is able to stream live video events with latencies making it useful for interaction.

Figure 9.9 shows the network traffic at the computers involved and the number of

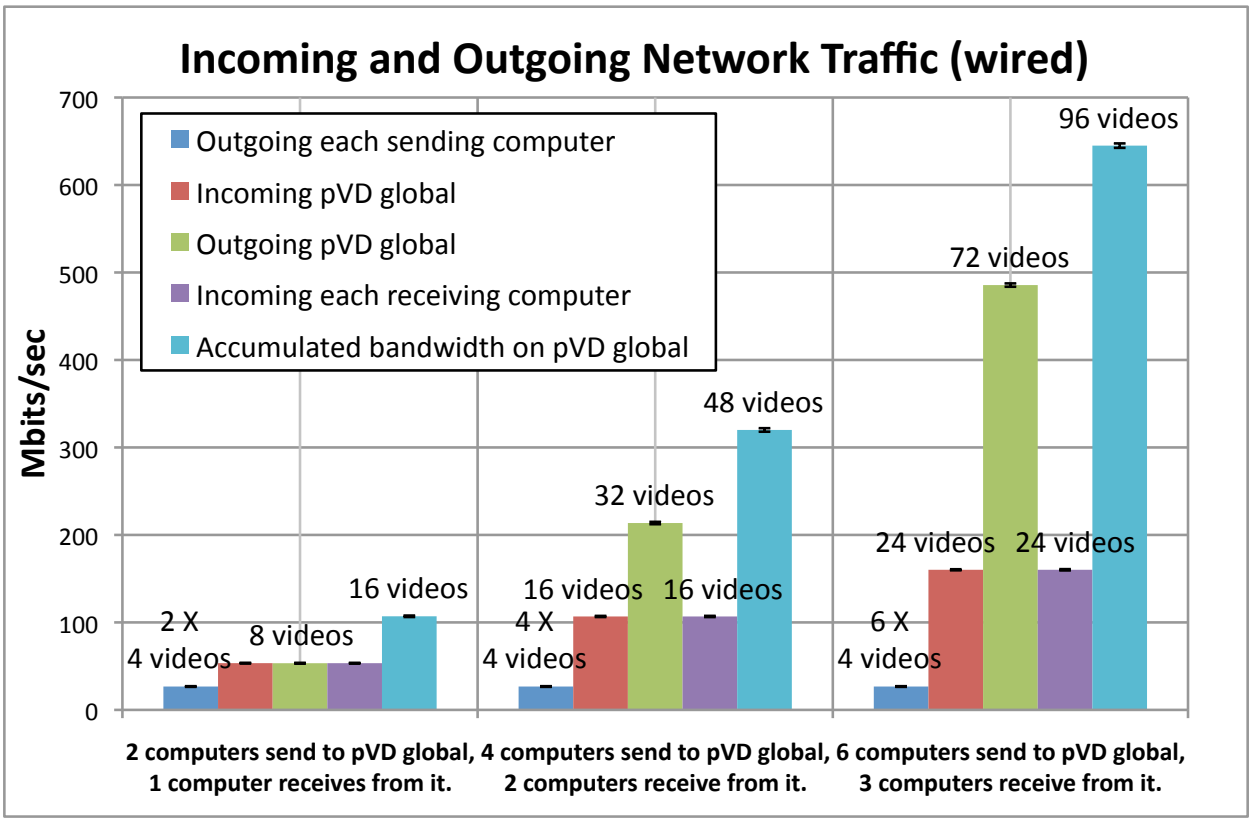


Figure 9.9: Incoming and outgoing network traffic using wired connection.

HD streams equivalent in each experiment. With two local pVD computers, the global pVD receives four HD streams (26.7 Mbit/s) from each computer resulting in a total of eight HD streams (53.4 Mbit/s) on the network simultaneously. The local pVD viewer computer receives all of the eight video streams from the pVD global computer, resulting in a max load of 16 HD videos in flight simultaneously on pVD global.

With six computers streaming to pVD global, we increased the number of pVD viewers to three. All of the viewers subscribe to every stream, so the global pVD sends out three times the incoming bandwidth (72 HD streams at 480 Mbit/s). The total number of videos in flight simultaneously is 96 on pVD global. This pushes the system beyond an expected normal usage, but we have not observed any significant packet loss.

In summary, the accumulated bandwidth on pVD global with two senders and one viewer is 107 Mbit/s, with four senders and two viewers is 320 Mbit/s, and with six senders and three viewers is 645 Mbit/s.

Figure 9.10 shows the CPU utilization on pVD global increases from 3.88 to 12.26% when it receives 8 to 24 videos and simultaneously sends 8 to 72 videos. The CPU utilization of each computer increased when the number of sent and received videos increased.

On a Gigabit network, the system can support in total 96 streams in the experiment.

The CPU utilization is also less than 15% in this case. This is much more than the normal usage. We conclude that the results show that the pVD global computer can easily be supported on even a low-end computer, and still have resources (such as CPU or bandwidth) available for other applications and systems.

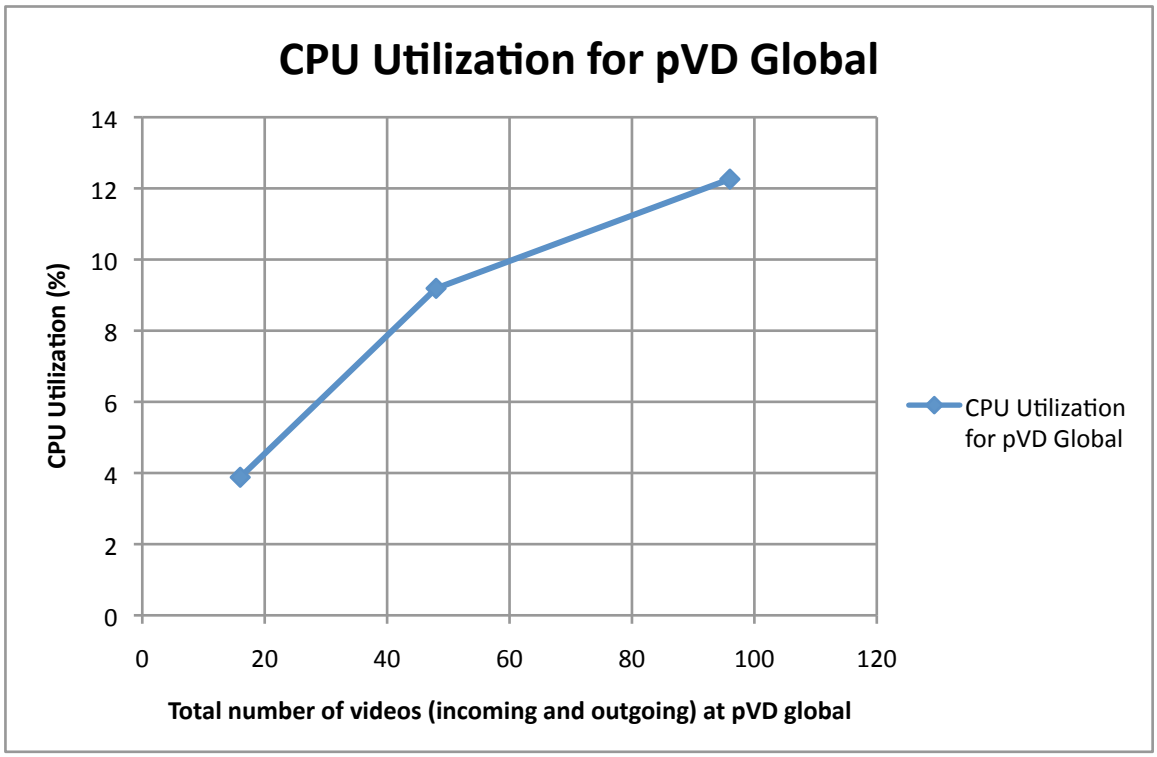


Figure 9.10: CPU utilization for pVD global.

9.5.2 Experiments on wireless network

To characterize the impact a wireless network has upon pVD, we configured a system where the pVD global computer is connected by a wired Gigabit Ethernet to an Apple AirPort Extreme 802.11n (4th Generation) WiFi access point, and where the other computers use the Wi-Fi network.

Figure 9.11 shows the subscription round-trip latency when local side computers are connected to the Wi-Fi network. The round-trip latency is about 1.7 ms for one computer with one subscription request, and about 5.7 ms for six computers with ten subscription requests each. The standard deviation increases a lot when more computers are involved.

Figure 9.12 combines the subscription round-trip latency when local side computers are on a Gigabit Ethernet and on a Wi-Fi network. The latency is less than 1 ms when all computers are connected to a Gigabit Ethernet, and it is less than 10 ms when the local side computers are on a Wi-Fi network. It is an insignificant increase. How ever the

standard deviation when all computers are in the wired network can be ignored compare to the standard deviation when all local side computers are connected to Wi-Fi.

The video end-to-end latency was between 90 and 125 ms.

Figure 9.13 shows the incoming and outgoing network traffic and the number of HD stream equivalents in each experiment. Local side computers are connected to Wi-Fi network. With two computers streaming to and one computer receiving from, the pVD global computer, eight videos were sent to, and fully received at, the receiving computer. The accumulated bandwidth at the pVD global computer was 107 Mbit/s. The receiving computer received 53.4Mbit/s. When a second receiver was added, for a total of two receivers, each received only 44Mbit/s instead of 53.4Mbit/s. We believe the reduced bandwidth can be removed by a more modern wireless network with better performance and resistance to interference from other nearby wireless networks. However, the experiment shows that it is possible to wirelessly stream at least eight HD videos to the pVD global computer and to wirelessly receive at least eight HD videos from the pVD global computer.

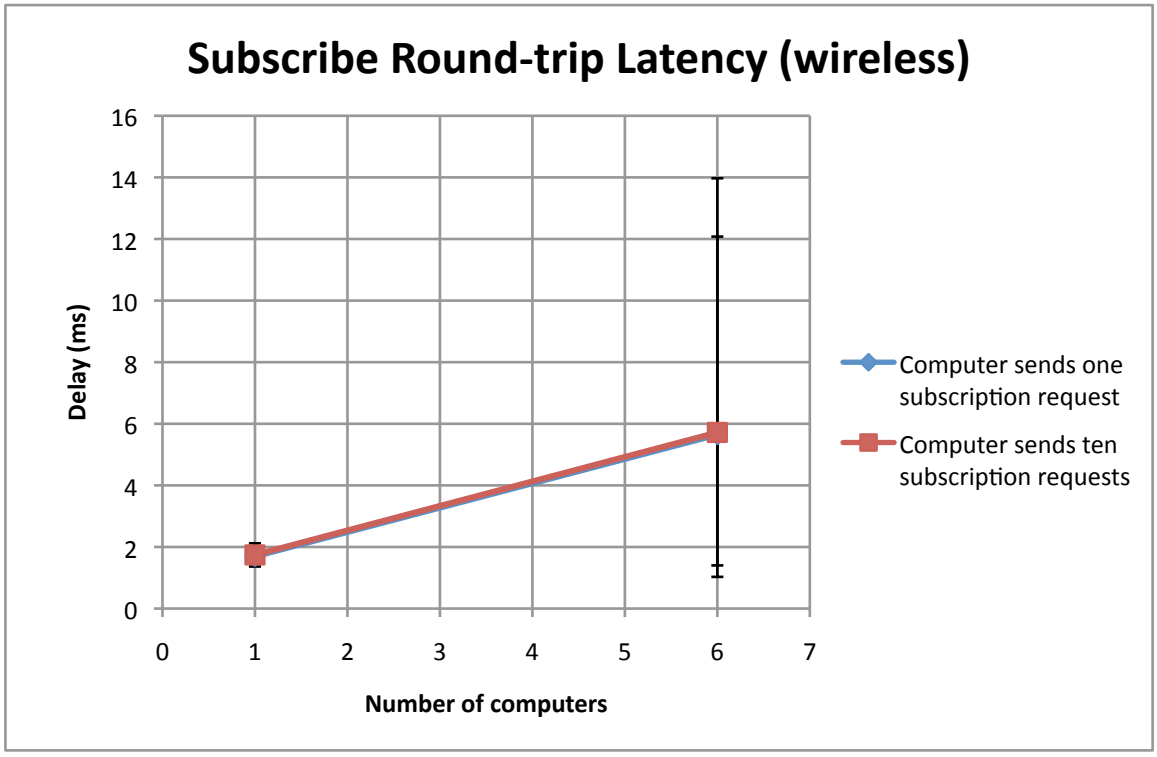


Figure 9.11: Subscribe round-trip latency when local side computers connected to a wireless network. There is one subscriber per local pVD computer. Each subscriber sends one request in the first experiment and ten requests in the second experiment.

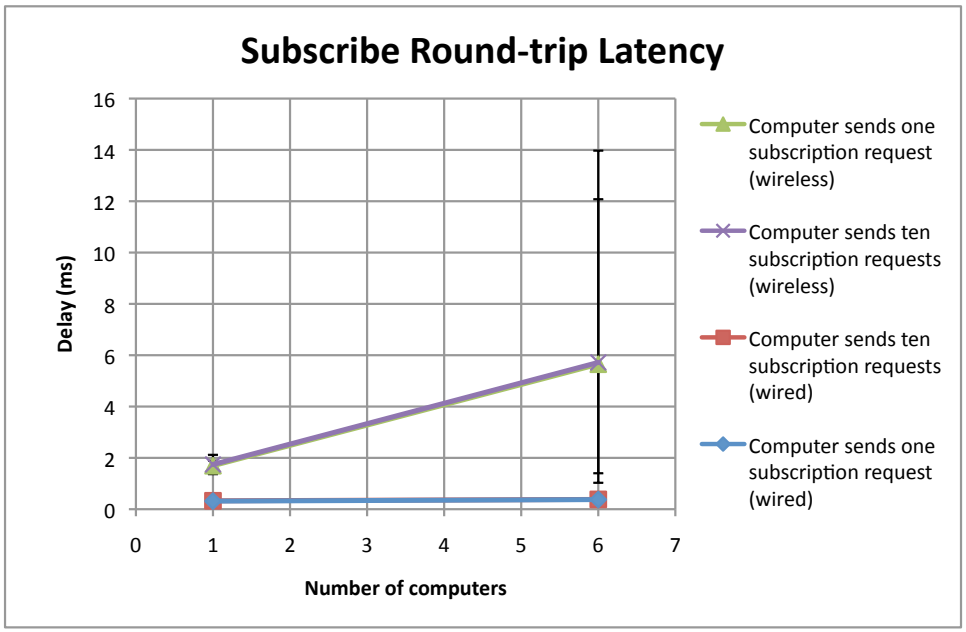


Figure 9.12: Subscribe round-trip latency on wireless and wired network. There is one subscriber per local pVD computer. Each subscriber sends one request in the first experiment and ten requests in the second experiment.

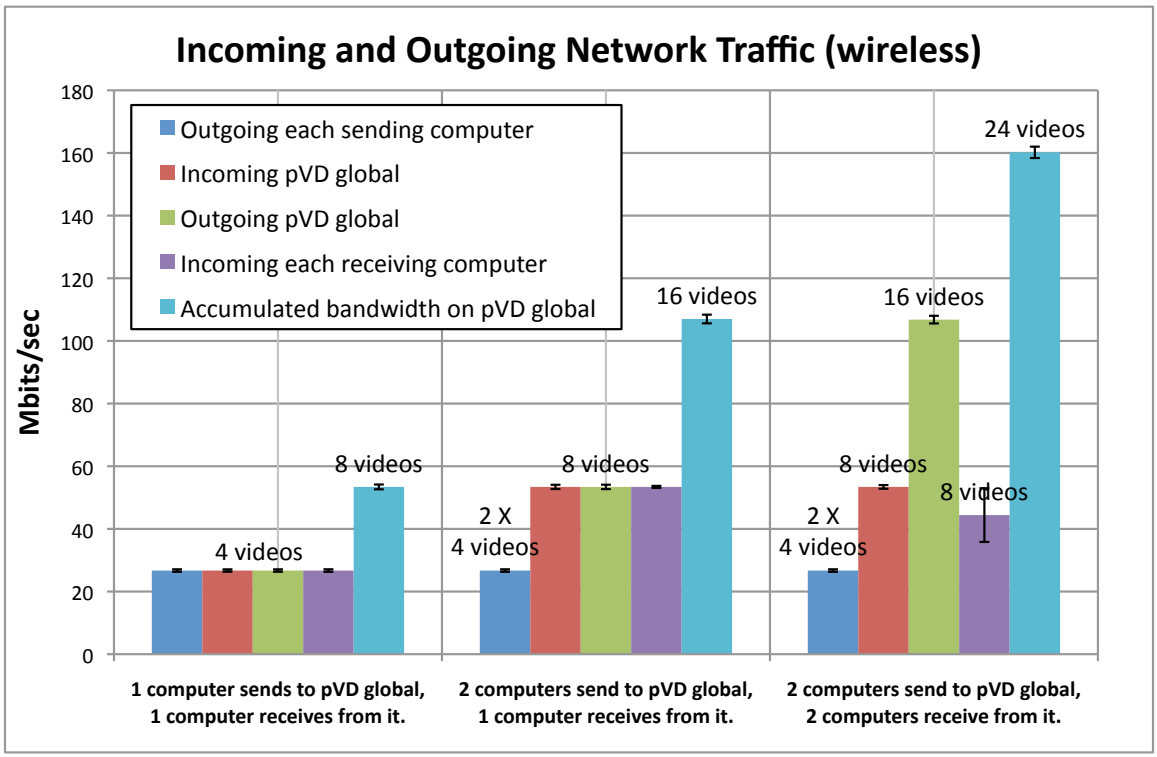


Figure 9.13: Incoming and outgoing network traffic using wireless connection.

9.5.3 Comparison on wired and wireless network

Figure 9.14 shows the resource usage on both wired and wireless network. On a wired Gigabit Internet, pVD supports 24 incoming and 72 outgoing videos. On a wireless network, pVD supports in total 8 incoming and 8 outgoing videos.

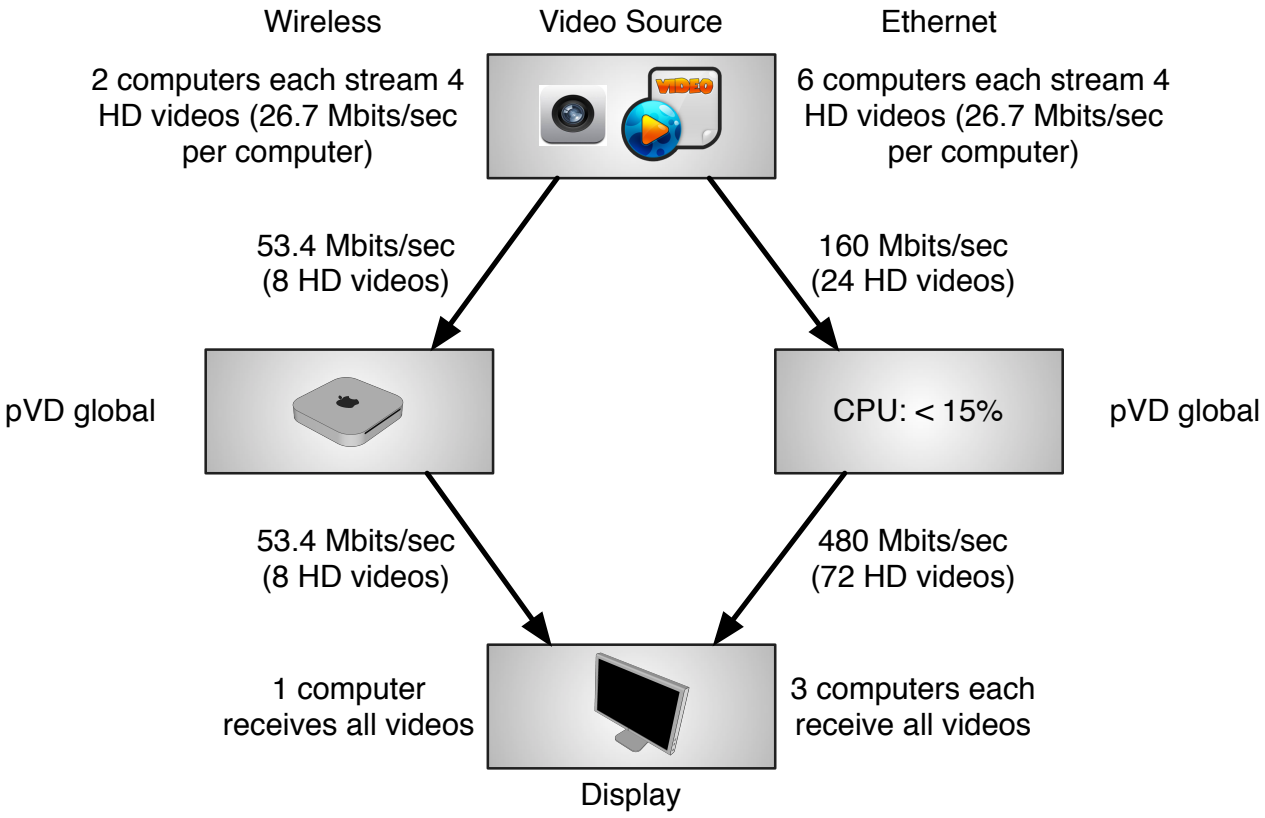


Figure 9.14: The resource usage on both wired and wireless network.

9.6 Discussion

pVD is based around a manual approach to controlling both video switching and privacy. A user must have access to all computers serving and consuming the videos. A user interacts directly with the pVD user interface on the sending and receiving computers. A user is the glue to bind together computers. When videos are sent between users, a sending user must manually accept a one-time streaming of a video to another user. The sending user can at any time halt the streaming. This provides for some control of privacy for the sending user. To strengthen the privacy we could have added techniques such as time-outs for video streams, halting them automatically when the time-out occurs. However, this adds complexity, and we wanted to keep the pVD system as rudimentary as we could within reason. The pVD system overall uses a simple and robust approach

customized for a single user with a handful of computers. However, it does not extend and scale to many computers and to many users, and was not meant to do so.

pVD can do handover of video streams between computers, letting a video start playing again from where it was stopped at another computer. The state of all videos is stored at and handled by the pVD global side server, and not by the pVD local side computers. Because pVD is meant for a single user with just a handful of computers, there are no performance issues in doing this centralized. It also aids in doing handover of videos between computers by having the state of the videos at one place. However, if the pVD global loses the state of the videos, the user must recreate it. We do not expect this to be an issue because of the usage domain with just a single user and a handful of computers.

The approach to sharing live and stored videos between multiple users will not scale to many users. It is intended to let a few users share live and stored videos in a case-by-case fashion. Users need to talk to each other to exchange enough information to connect. This can be automated and made more efficient, but we wanted to keep it basic and simple, and to involve the users in the sharing to reduce unintended sharing. While the multiple user approach demands the interaction of users, we still believe it is useful for simple ad hoc interaction and sharing between family and friends.

We have deliberately used a Gigabit wired Ethernet for some of the experiments where the goal was to measure the performance behavior of pVD global. A typical usage scenario is to have the pVD global computer connected by wire to a wireless access point while a user's computers share videos with each other through a wireless network. A single 8 Mbit/s HD video stream will in this setup at the worst consume about 16 Mbit/s of the wireless network. Wireless networks typically range from 54 Mbit/s to 300 Mbit/s. A wireless network should in practice be able to support a pVD configuration with a handful of computers and video streams. For the intended usage domain this is enough. Emerging wireless networks such as the 802.11ac technology [95] can achieve 1300 Mbps and should together with future computers allow for even better performance of pVD.

9.7 Conclusions

pVD is a system for simple ad hoc sharing of live (camera) and stored video streams between a single user's computers.

The pVD system uses subscriptions to bind together a video and camera on one computer with the other computers. The user must have physical access to each computer to set up a subscription, and to start the streaming and receiving of a video. This is a requirement of the prototype implementation. When a sender and a receiver run

simultaneously, videos will start to flow according to the subscriptions. The system saves video state and allows the user to continue watching a video on another device, picking up from the same position in the video.

The system avoids issues with relying on third parties and access rights by being designed for very small scale use and to be run only on a user's personal computers. The system can also be kept simple by not having to scale to support a large number of videos, computers, and users.

Network bandwidth is not a critical issue for the pVD prototype system and the system does not occupy too many system resources, such as CPU. When all computers are on a gigabit wired network, the system can support many simultaneous streams using a standard PC desktop for the pVD global side. The system also supports a handful of simultaneous streams on a wireless network. The system responds fast enough to a user's requests and the latency for streaming live events will be noticed but accepted by the user.

Chapter 10

Discussion

The prototype of the MultiStage system uses a centralized global side for the distribution of data streams to several local sides. The local side subscribes to data streams from the global side DSDS. Another possible approach is to distribute DSDS to every local side. The advantage is that if one DSDS fails, the other DSDS will still be running. Consider the bandwidth, if we have N local sides and each local side sends out M streams. Each local side subscribes to all streams from DSDS. For the centralized approach, the global side will receive $N \times M$ streams from WAN and send $N \times M \times N$ streams to WAN. For the distributed approach, each local side DSDS will receive $(N-1) \times M$ streams from WAN and send out $M \times (N-1)$ streams to WAN. Because no centralized DSDS is used, we now need a distributed approach to measure latency and clock difference.

The back to back configuration of cameras avoids having the infrared dot cloud used by the cameras (to achieve depth information) interfering with each other. While more Kinect cameras can be used, care must be taken to avoid interference. A more advanced sensor suite will help avoid this problem.

The local side LSM can generate 3D point cloud and 2D RGB image. For the 2D RGB image, LSA can do motion detection and just send control command to the global distribution side. In this way, bandwidth is saved compare with sending images directly to the remote side. By doing both local and global state analysis, it will reduce CPU load at the global side computer compare with doing everything globally.

Communication protocols: The MultiStage system uses both UDP and TCP protocols. The UDP protocol is used for sending and receiving data streams. The reason is we want data streams to be delivered as soon as possible to achieve lowest delay and we can tolerate few packets loss. TCP uses a mechanism that it will retransfer the packet if the other side loss the packet. In this case, TCP protocol will increase the transmission delay. However, MultiStage system uses TCP protocol for sending the subscription messages. This makes sure DSDS will receive all local side subscriptions.

NTP: The system currently uses NTP protocol to synchronize the latency between different local sides. However, when local sides are located in different places, the clock difference of computers can be tens of milliseconds. From Section 8.3.2, for tight interactions, more than 200 ms will make interaction awkward. An improved time synchronization approach can be explored to achieve better results.

Latency: From Section 8.3.2, the actor-to-actor latency is always higher than what is tolerable for tight interactions. We need to apply different approaches to mask the effects of delays for this type of interactions. The tolerable latency for looser interactions is up to 800 ms delay. This means we do not need to apply the masking approaches for this type of interactions.

The masking the effects of delays approaches:

1. The Act-By-Actor approach is sufficient for slow movements but when interactions become tighter, an actor's reaction delay can be noticed.

2. The Act-By-Director approach can tolerate up to 500 ms delay. The Live Stage approach further adjusts the start time at all secondary stages and lets them start the performance earlier. One live stage will be synchronized. The Local Delay approach delays the local remote presence to wait for the remote presence of remote actors to arrive. The remote presence and actors will be visibly out of sync when delays are more than 500 ms for handshake type of interactions.

3. The Act-By-Wire approach synchronizes all stages and actors. When latency goes high, either a prerecorded video will blend in or a model of an actor with predetermined actions will be used. In both cases, the audience cannot see the real performance of actors on remote stages.

Will the audience notice the delay of the masking the effects of delays approaches: For the Act-By-Director approach, latencies are measured before the play, and audience will not notice the delay because this approach applied in advance. For the Live stage approach, live stage can be switched during intermission of a performance. For the Local Delay approach, audience may notice actor and the remote presence of the actor out of sync if delay is too high. The Act-By-Wire approach has a period to check the latency and decide whether to use the masking approach. Whether the audience notices the short delay depends on how long the period is. In the current implementation, it is set to 3 second. More studies are needed to determine a better period.

Informal User study: This is a system's dissertation and thus no formal user studies were conducted. We mainly used handshake type of interaction in the experiments about human response latency, human tolerable latency and maximum system end-to-end one-way latency for masking the effects of delays approaches. The handshake type

of interaction can be both loose and tight interaction. Even if masking approaches are applied, it is not easy to make the handshake look like the real thing.

For the System Performance and State Monitoring system, local monitor processes at local side computer send measurements to global monitor process at DSDS. Local monitor just needs to know the location of global monitor. Global monitor knows the location of the Administration Interface and send measurements to the interface. The measurement about clock difference assumes the time when stage process receives the message from DSDS is half of the round-trip latency. However, the actual latency may not exactly the same as the assumed value. More possibilities can be explored to get more accurate latency and clock difference.

Figure 8.2 shows network traffic usage for one to three stage configurations. In the extreme case with four cameras running (30 fps and 5000 points per image) at each stage (three stages), there are 12 incoming data streams at DSDS. All three stages subscribe to data streams from DSDS. This results in 36 outgoing streams at DSDS. The total consumed network bandwidth is about 640 Mbits/s. The required capacity can be provided by a Gigabit Ethernet. But if we add one more stage, DSDS will have 16 incoming data streams and 64 outgoing data streams. The total consumed network bandwidth is about 1066.7 Mbits/s. The network can become a bottleneck if more stages are added. The problem can be solved by lower the quality of the image, so that each data stream occupy less network bandwidth. The problem can also be solved by using the distributed approach, DSDS is running at each local side. This results in 12 incoming and 12 outgoing streams on each local side DSDS. The total consumed network bandwidth reduces a lot compare to the centralized approach.

pVD: The sum of the subscribe message latency and the end-to-end latency is less than 200 ms. The subscribe latency grows linearly with the number of computers. This is acceptable because pVD is designed for a single user's computers. The pVD system is able to stream live video events with latencies making it useful for some types of interaction (a computer's camera can be seen interactively fast by other computer, etc.)

Chapter 11

Contributions

Here is a brief introduction about the contributions of the work. This is similar to the contributions section in Chapter 1, but with focus on where the contributions have been made, especially after given the idea, architecture, design and implementation of the system.

11.1 Principles

1. **One Actor: One Data Stream**: The one to one mapping between each actor and each camera allows for great flexibility in treating each actor individually when looking for gestures, and where each actor's remote presence is manifested and located in relation to the other actors on the stages. The advantage of having a one-to-one relationship between actors and cameras is that it takes very little processing to create individual streams for the actors. This helps to reduce the delay between when an actor moves and when it is manifested in the remote presence at remote stages. The disadvantage is that when the number of actors increases, so must the number of cameras (See Chapter 3).

2. The models for remote interaction between actors have been identified (see Chapter 6).

   (a) **Act-By-Actor**: the actors react to the remote presences of remote actors as if the latter were i. the actual actors; ii. Physically present on the same stage. This may suitable for slow type of interaction, such as a handshake. But for rapid interaction, it is hard for actors to interact when latency is beyond a certain threshold (350ms for rapid hand movement).

   (b) **Act-By-Director**: Each stage simulates the behavior of a director. It tells actors when to do actions and what the actions are. All actors follow the local

director. Assuming the scripts used by the directors are made correctly, even if the actors act on command, it will seem to an audience as if they interact freely with each other. And in a manner they would do if they were physically on the same stage. The principle is used by the **live** stage and **delay remote presence** approaches.

(c) **Act-By-Wire**: Remote presences are manipulated to mask the effects of delays. Manipulations include just-in-time blending in of prerecorded data streams of remote presences of actors, and just-in-time blending in of on-demand computed remote presences.

Different approaches to masking the effects of delays should be expected and need to be based on what the actors are doing. For the slow movements such as handshake type of interaction, longer delay can be tolerated. Act-By-Actor approach can be used in this type of interaction. But for rapid movements such as dancing, more advanced approaches such as Act-By-Director or Act-By-Wire approach should be applied.

3. Two different types to masking the effects of delays (see Chapter 6). **Act-By-Director** approach **coordinates** actors and let them interact at the same time. **Act-By-Wire** approach monitors the delays of each incoming data stream, and **substitutes** delayed data with locally computed data, or with data already available at each stage.

4. **Amplified Interaction**: In section 2.7, we gave a brief introduction about the idea of amplified interaction. State Monitoring and Analysis system (see Chapter 3) has functions to monitor actors' actions and analyze the actions to find out gestures. The Remote Presence system (see Chapter 5) will amplify actor's interaction by creating the animation of the gesture.

5. **Local vs Global Gestures**: The gestures performed by local actors are local gestures. Global gestures are performed together by actors on different stages. In Chapter 3, we described LSM and LSA to do local state monitoring and analysis. GSM and GSA will perform global state monitoring and analysis.

6. **Reliability by Receiver Autonomy**: Receiver side makes decision on how to use the data streams. For example, data streams maybe replaced with pre-recorded data streams if the Act-By-Wire approach is enabled (see Chapter 5).

7. **Flexibility by Receiver Autonomy**: Each individual data stream represents one actor on one stage. Each individual data stream can be mapped onto any display

in any stage, making for a very flexible combination of background, objects, and users (see Chapter 5).

11.2 Models

1. **Decoupled Producer and Consumer with Subscription**: MultiStage uses this model to handle the handover of data streams. Local sides publish data streams to global side DSDS. Each local side also subscribes to data streams from global side DSDS (Chapter 4).

   Another use of this model is called pVD (Chapter 9). It is designed for video sharing for a single user's computers. The system avoids any third parties, and relies only on the user's personal computers. The architecture is comprised of functionality for sending videos, subscribing to videos, and maintaining the video playback state. pVD handles incoming and outgoing videos between a single user's computers. To display a camera feed or a video at a computer, it must subscribe to videos. pVD allows a computer to send subscriptions to it. pVD will receive all subscriptions and use them to manage the switching between computers. The state of the video is also stored and allows the user to continue watching it from another computer.

2. **Masking the Effects of Delays through Coordination**: Act-By-Director is based on this model. Actors are coordinated by a script defining when to do an action and what is the action (Chapter 6.2).

3. **Masking the Effects of Delays through Substitution**: Act-By-Wire is based on this model. When data has been delayed too much, system will blend in pre-recorded data or blend in on-demand computed data (Chapter 6.2).

4. **Interactive System State**: The MultiStage system also used this model to monitor the internal state and doing live performance measurements of latency and bandwidth (Chapter 7.2).

5. **Actor-Local Sensing**: The system assumes that just a single actor is within the 3D field of view for each Kinect camera. All objects outside of this 3D space are ignored. (Chapter 3).

11.3 Artifacts

The prototype system for the MultiStage model include the following artifacts:

Sensor Suite: Each local side stage contains a sensor suite, which includes four Kinect cameras. The cameras arranged in a square cover almost 360-degree. Actors act around the cameras. This is described in details in Section 2.7.

Distributed State Detection: The sensor suite on each local side connects to computers. The computers have functions to detect actor's actions and generate data about actors. The captured data will be ready for distribution. This is described in details in Chapter 3.

Distributed State Analysis: The computers have functions to analyze for gestures from captured data. This is described in details in Chapter 3.

State Distribution: The Distribution system DSDS binds together all stages and manage the distribution of data streams to the stages. Local side stages publish data streams to DSDS and subscribe to data streams from DSDS. This is described in details in Chapter 4.

Remote presence System: The Remote Presence subscribes to data streams from DSDS. It creates the remote presence of a physical actor from the data streams. Actors on each stage can interact with the virtual presence of the actor. Audience can view the performance of actors and remote presences of actors. The remote presence also rendering the detected gestures. The detail is described in Chapter 5.

Human Interaction and Collaboration System: The Controllable Temporal Synchronization - Collaboration System provides approaches to mask the effects of delays. The approaches include: Act-By-Actor approach, Act-By-Director approach and Act-By-Wire approach. This is described in detail in Chapter 6. The Human Interaction System coordinates actors on different stages by running a script on each local side stage. Tell them when to start an action and what the action is. To enable the Act-By-Director and Act-By-Wire approach, the script must be also enabled. The detail about Human Interaction System is described in Section 7.4.

System Performance and State Monitoring System: The system has several local sides and a global side. The local side monitoring process running on each local side of MultiStage system. It detects MultiStage local side computers performance metrics. The global side monitoring process running on the global side of MultiStage system. It detects the MultiStage global side computers performance metrics. The local side process report monitored metrics to the global side process. The global side process gathers all monitored metrics. It also reports the metrics to the Administrator Interaction System. The detail about the system is described in Section 7.2.

pVD: The system does not aim for large-scale video distribution. It was designed to allow video distribution between any of a single user's devices, and it allows for a smooth handover of playback between computers. The system has several local sides and

a global side. The local side sends and views videos, and the global side coordinates the switching and distribution of videos, and maintains subscriptions and video state. The system allows videos from different computers to be viewed at other computers. It avoids any third parties, and relies only on the user's personal computers. The detail about video sharing for a single user's devices is described in Chapter 9.

11.4 Facts

As described in Chapter 8, the experiments are divided into two parts.

Resource Usage Experiment:

1. Each stage stream point cloud at maximum 5000 points needs about 53 Mbits/s bandwidth. We tried maximum three stages resulting in total 12 incoming data streams and 36 outgoing data streams at DSDS. The total consumed network bandwidth is about 640 Mbits/s. A Gigabit Ethernet is sufficient in this case.

2. The highest CPU load is less than 25 percent. The system is not resource limited and still has resource available for other applications.

Latency Experiments: Table 11.1 shows the measured latency in different experiments.

Latency	Value
Global-to-local round-trip latency	Local: 1-2 ms. Global: 32 ms
System end-to-end one-way latency	Local: 90-125 ms. Global: 100-158 ms
Actor-to-actor round-trip latency	Local: 180-250 ms. Global: 200-316 ms
Human response latency	345 ms
Human noticeable latency	190-225ms
Human tolerable latency	Rapid hand movement: 350-400 ms. Slow hand movement: 800 ms
When to start masking	When total delay is more than about 280 ms
Cost of Masking	Insignificant cost of CPU load

Table 11.1: Measured latency in different experiments. The total delay defined in when to start masking is the time between when an image has been timestamped (the system add timestamp after it captures the image) and when the Collaboration system receives this image.

Figure 8.8 shows the maximum system end-to-end one-way latency at which each masking approach is in principle at least partially successful at masking the effects of delays.

1. The Act-By-Actor approach has the lowest system end-to-end one-way latency before the temporary causal synchrony break.

2. The Act-By-Director approach has a script to synchronize all actors. When maximum system end-to-end one-way latency is about 390-525 ms, the temporary causal synchrony will break because of the remote presence about remote actor is out of synchronization with local actor.

   (a) Live Stage approach synchronizes all remote presences on one live stage. Because of other secondary stages will start performance earlier than live stage, the temporary causal synchrony between actors and remote presences on secondary stages will break when maximum system end-to-end one-way latency is about 390-525 ms.

   (b) Delay Remote Presence approach synchronizes all remote presences and all actors at every stage. But because the remote presences are delayed, the temporary causal synchrony between all actors and all remote presences at each stage will break when system end-to-end one-way latency is about 390-525 ms.

3. The Act-By-Wire approach synchronizes all actors and remote presences at every stage by blend in on the fly created remote presence when data streams about remote actors arrive too late. In the current implementation, this approach will blend in either pre-recorded video or command to control human skeleton's arm movement. The audience may notice the switch back and forth between live data stream and the pre-recorded data stream.

Chapter 12

Conclusion

The MultiStage system is a distributed system. Implementation of the MultiStage system has several local sides (at each stage) and one global side. A local side has several sensors to detect local actions and analyze for gestures. It also streams data streams to global side. The global side binds together all local sides. It analyzes global behaviors from received data streams and streams data streams back to the local sides according to local side subscriptions. Each local side then displays the received data streams and amplifies actors' interactions on its display. Each computer has a monitoring process. It periodically checks the state of all computers, including CPU utilization, memory usage, and bandwidth. It also checks the latency and clock difference between the global side computer and local side computers. The measured data can either be used to masking the effects of delays or to help the user find out errors faster.

When humans interact across distance, delays cannot be avoided. The MultiStage system has several approaches to mask the effects of delays. Consequently, when delays become too large, some of the effects of delays can be masked to create an illusion for the humans interacting, and for observers, that they are in the same room or on the same stage. However, the illusion created by masking has several limitations depending on which masking approach is used. This research mainly investigates two different types of masking. One type coordinates the interaction at suitable times to create a better illusion. The other frequently monitors the delays, and substitutes delayed data with data already available at each stage. Depending on the type of interaction, a suitable masking approach should be selected.

Experiments for the MultiStage system have been conducted and can be divided into two main types. One is a resource usage experiment and the other is an experiment to check latency and different approaches to mask the effects of delays. In the resource usage experiment, we measured the CPU utilization, memory, and network traffic for all computers. Each computer is connected to a wired Gigabit Ethernet. The results show

that the resource usage in all cases is either very low or low. The implication is that the system is not resource limited. The system scales to at least three stages with a total of at least 12 outgoing and 36 incoming data streams. With regards to bandwidth, the location of the distribution server is presently not critical. In the experiments evaluate approaches to mask the effects of delays. The approaches we developed and did performance measurements on, demanded insignificantly more resources than not using them, and can even in the most complicated case when using Act-By-Wire, be switched in and out with insignificant delays. Based on informal use of the system, we found that even a delay of 800 ms while interacting using slow movements in some cases was tolerable. However, the general case seems to be that delays above 200 ms are noticeable when having remote presences based on vision and visualizations. We found that an actor-to-actor round-trip delay of above 200 ms is frequently the case, and masking is consequently frequently needed.

The video streaming architecture can also be used in the home environment. This is mentioned in Chapter 9. A pVD system is introduced in this chapter. The pVD system is designed for simple ad hoc sharing of live (camera) and stored video streams between a single user's computers. The pVD system uses subscriptions to bind together a video and camera on one computer with the other computers. The system also saves video state and allows the user to continue watching a video on another device, picking up from the same position in the video. In one of the experiments, we simulated the real-case scenario. Each user's computers are connected to a Wi-Fi access point and the global side is connected to a Gigabit Ethernet. In this experiment, the system also supports a handful of simultaneous streams on a wireless network and responds fast enough to a user's requests. The latency for streaming live events will be noticed, but accepted by the user.

Chapter 13

Future Research

Some possible changes and improvements for future research for the MultiStage system are presented in this chapter.

The global side of MultiStage system currently is a centralized server. We could also use a distributed approach to design and implement the global side and make a comparison between the two different implementations. The global side would then be integrated into each local side. This will result in less incoming and outgoing data streams through the Internet at DSDS. More local side stages can be added. The problem will be each local side DSDS needs to know the location of all other DSDS. Compare to the centralized approach, all local sides just need to know the location of the centralized DSDS.

The user studies we have done, only based on a few people's opinion, and the interaction used was only a handshake type of interaction. We have not done formal user study experiments exploring the system capabilities with actors needing to tightly coordinate their movements. A formal user study for different approaches to mask the effects of delays is required to find out how satisfied actors and audience are with different masking approaches and in what situation different masking approaches can be applied. More complicated interactions, such as rapid movements and dancing can be applied in the future user study.

For the Act-By-Wire approach, instead of using a static prerecorded video, a model is computed to create the remote presence. A wide range of possibilities is in principle available. These include blurring the movements of an actor such that delays are not so obvious and predicting what an actor is going to do. The current prototype just uses a simple human skeleton with arm movements. This approach can be further developed to add more complicated movements and predictions. More approaches to mask the effects of delays can also be explored.

The system uses NTP protocol to synchronize the clock between computers. When

each local side is located at different locations that are far away from each other, how accurate the clock is between different locations needs to be measured.

The actor script tells actor when to do an action and what this action is. It is currently implemented as a single script for each stage. But for more complicated movements, one script for each actor will be supported letting each actor on same stage do different actions. Even if they perform same action, it can be at different time.

Future research on the use of sensors include:

1. The only sensor our MultiStage system has now is Kinect cameras. In the future, more sensors could be added to the system. For example, we could add microphones to capture sounds.

2. Actors can get sensors on body. This will provide information more than on-stage sensors. For example, on body sensor could detect the heart rate of the actor and computer can decide to change the pace of the action. The on body sensor could also detect small movements and eye movements about actors.

3. We can also let MultiStage system support different mobile devices. People can connect to the system with their mobile device, to receive data stream from or send data stream to the MultiStage system.

Chapter 14

Appendix A - Published Papers

14.1 MultiStage: Acting across Distance

This paper was published in Second International Conference on Information Technologies for Performing Arts, Media Access and Entertainment, ECLAP 2013, Porto, Portugal, April 8-10, 2013.

MultiStage: Acting across Distance

Fei Su, Giacomo Tartari, John Markus Bjørndalen,
Phuong Hoai Ha, and Otto J. Anshus

Department of Computer Science
University of Tromsø, Norway
{fei.su,giacomo.tartari}@uit.no,
{jmb,phuong,otto}@cs.uit.no

Abstract. We report on a prototype system helping actors on a stage to interact and perform with actors on other stages as if they were on the same stage. At each stage four 3D cameras tiled back to back for an almost 360 degree view, continuously record actors. The system processes the recorded data on-the-fly to discover actions by actors that it should react to, and it streams data about actors and their actions to remote stages where each actor is represented by a remote presence, a visualization of the actor. When the remote presences lag behind too much because of network and processing delays, the system applies various techniques to hide this, including switching rapidly to a pre-recorded video or animations of individual actors. The system amplifies actors' actions by adding text and animations to the remote presences to better carry the meaning of actions across distance. The system currently scales across the Internet with good performance to three stages, and comprises in total 15 computers, 12 cameras, and several projectors.

Keywords: Temporal Synchronization; Remote Interaction; Computer Mediated Collaboration.

1 Introduction

We envision computer mediated collaborative performances where actors at physically remote locations, as illustrated in Figure 1, interact and coordinate their actions as if they are next to each other on the same stage or in the same room. Through various means, including audio, video and animations, each actor has a remote presence at one or several remote stages. We are interested in how to mask the effects of delays and distance.

In this paper we describe a system doing this for the visual side of a remote presence: MultiStage collects state, like video, about each stage through various sensors, like cameras and microphones, and analyses the observed state to identify information like actor gestures. State data and information is streamed between stages to maintain a remote presence for each actor, and to monitor and control the system.

Each stage has several incoming data streams that are used to create a presence of remote actors. Actors in a room watch and react to the remote presence

P. Nesi and R. Santucci (Eds.): ECLAP 2013, LNCS 7990, pp. 227–239, 2013.

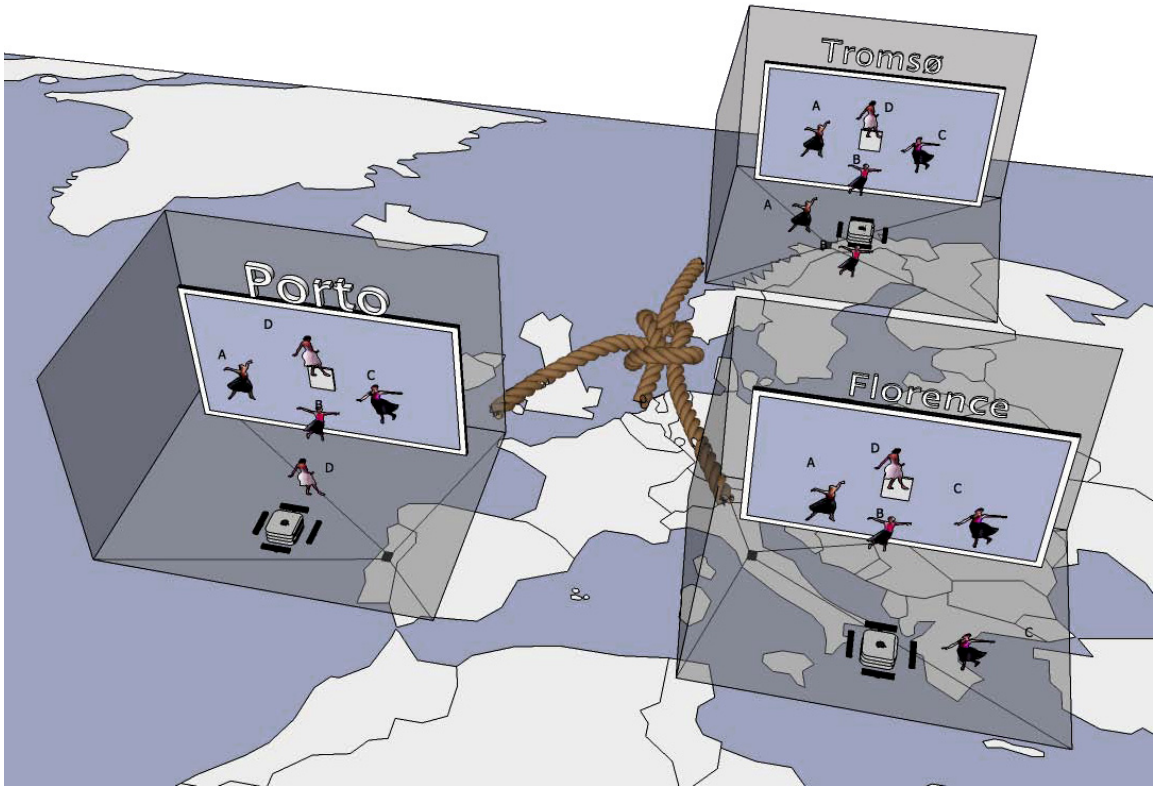


Fig. 1. Four dancers at three different stages dance together. Each stage is equipped with sensors to detect actors and a display to visualize the remote presence of all the performers. The rope and knot represent the global system binding together the stages.

of other actors. There can also be several third parties, audiences, just observing, and not directly participating. Audiences can be physically present at one of the stages, or be on the Internet. An audience local to a stage can watch the local physical events unfolding, and watch visualizations of both the local and remote events.

In principle, there will always be some delay from an event happens until it can be observed. Light alone needs 134ms to travel the length of Earth's equator. In practice, the total delay when observing a remote event includes delays coming from the sensors, transferal of data from sensors to computers, processing of the sensor input, network transmission, on-route processing, receiving and processing the received data, and preparing and visualizing the data locally. Even if the delays can be reduced, they can never be removed. Consequently, we have to live with the delays, and find ways of reducing the effect they have on the actors and the audiences. The effect of the delays can be reduced through different techniques including on-the-fly manipulation of the remote presence representation of actors. We must also mask the effect of distance. On a theater stage the actors use several techniques including costumes, makeup, and exaggerated movements to reach out to the audience. We propose to let a user instruct the system through gestures to add enhancements to the remote presence. For example, a given arm movement could be turned into a text bubble above the visualization of the user, or a glowing halo around the arm. We call this *amplified interaction*.

There are many commercial tele-conferencing and messaging systems where two or several persons interact through instant text, video and audio as well as file transfer. The latencies can be quite tolerable. However, teleconferencing systems are best when used in unstructured interaction without interactively fast synchronized movements of participants. Tele-conferencing systems are typically not flexible with regards to manipulating remote presences, and how they are arranged on, say, a display. They also lack functionalities for amplified interaction.

2 Related Literature

Several research systems for collaboration exist. In [1], [2] and [3] two room interaction systems are described with a focus on achieving audio synchrony. They compensate for the network latency by delaying local actions correspondingly, making both rooms experience the same delay. In [2] a series of experiments based on the DIP system is described with focus on the audio delay, and how the delay affects musician's cooperation. An artificial delay of 50ms to the remote room's audio stream was tolerable. With the same latency added at both rooms it became possible to play easily together with a delay of up to 65ms. This approach used by DIP for audio can also be used by MultiStage for video.

In [4] a remote camera system for teleconferencing supporting user cooperation between a local and a remote room is described. The system captures 360-degree images as well as supports pan/tilt/zoom of cameras. The audio and video can be recorded. Consumed network bandwidth is from 1.95 to 7.4MBytes/s.

In [5] a three-room distributed collaboration system is described, allowing three people to collaborate in a virtual environment. At each room there is a multi-touch table, camera, speaker, microphone, and two LCD monitors to display the two other rooms. The shadow of remote hand and arm gestures are captured by an infrared camera and displayed on the multi-touch table to show the remote person's behavior.

In [6] a remote presence system using a remote controlled android is described. The state of the android includes idle, speaking, listening, left-looking and right-looking. A teleoperator control android's behavior by choosing its state. They conclude that using an android gives a strong remote presence.

In [7] a system intended for informal meetings between rooms is described. The system merges the images from panorama cameras acquiring the background of a room, with a camera acquiring the users when they are close by the display. The system amplifies the remote presence of the users by allowing users to maintain eye contact during a conversation.

In [8] a multi-camera real-time 3D modeling system for tele-presence and remote collaboration is described. 3D models of users are computed from 2D images from multiple cameras, and the 3D models are streamed to remote rooms where users are visualized in a virtual 3D environment. Computing and visualizing collisions and reaction forces to virtual objects in the virtual space strengthen the remote presence. The system is built on top of a middleware that simplifies the use of a compute cluster to obtain 3D meshes and textures from the cameras.

In [9] a multi-modal corpus for research into human to human interaction through a virtual environment is presented. The virtual environment is defined as a virtual dance studio where a dance teacher can teach students choreographies. Both teacher and students are represented in the virtual studio by 3D avatars. The corpus consists of the recordings of the 3D avatars and outputs form other sensors, such as cameras, depth sensors, audio rigs and wearable inertial measurement devices. A dance instructor and a musician provided also some ground truth annotations for the corpus.

In [10] a study on hand gesture speed classification with the goal to improve the human-computer interaction is presented. The aim of the study is to train a virtual human to detect hand's movement in a noisy environment. The factors of the study are multiple body features like hand, wrist, elbow and shoulder, evaluated against different gesture speed such as slow, normal and fast.

3 Temporal Causal Synchrony between Actors

Some actions by actors are causally related. One actor does an action, and some time later another actor does an action because of the first action. A system must preserve the order of the actions when they are causally related.

Even if causality is preserved, there is a delay between an action and the corresponding reaction(s), and the system should ideally keep the delay low enough to make actors experience it as if they would when on the same physical stage. How large the delay is indicates how well actors are in temporal causal synchrony.

We define actors to be in *loose* temporal causal synchrony with each other when there are no special demands on delays. This is typically the case in unstructured interaction where it does not matter a great deal if actions by actors are slightly delayed or out of order with each other. This will typically be the case in teleconferencing with approaches like Skype.

However, for structured interaction with coordinated movements, as in synchronized dancing and in rapid action-reaction situations like, say, martial arts, correct causal ordering and short delays become critical to preserve the illusion that the actors are on the same stage. We define *interactive* temporal causal synchrony to be when actions by an actor is seen in causal order and as fast as actors are used to when being on the same stage.

Delays are unavoidable, and they can be large and even varying enough so that interactive temporal causal synchrony can not be achieved. In these cases we must mask the effects of the delays to create an illusion of synchrony. Some approaches are outlined in the following.

Actor Feedbacks: The actors reacts to the remote presence videos as if they were the actual other actors. Depending on how large the delays are and how much they vary, the interactions can become awkward. Only loose temporal causal synchrony can be expected to be achieved.

Shared Clock, Shared Performance Start-Time, Individual Actor Scripts: We synchronize the clocks of all computers, set a performance start-time

and begin a count-down at each stage. When the count-down finishes each actor starts acting according to a script defining what the actor should do and when the actor should do it (for instance, two actors doing handshake). Assuming that the scripts are made correctly, even if the actors don't actually interact it will seem to an audience as if they do. In this approach the scripts have been made with knowledge about the delays, and each script tell the actor when, modified by the delay, to do an action.

Shared Clock, Individual Performance Start-Time, Individual or Shared Actor Scripts: We synchronize the clocks, and select a start-time for the performance. We select one stage to be the live stage. The other stages are secondary stages. We measure the delay from the live stage to all secondary stages and modify the start time of each stage's count-down according to the delay between it and the live stage. When the count-downs finishes, all remote presences will move at the same time and in synchrony with the actors present at the live stage. The actors and the audience at the live stage will see the other stages as if they are in interactive temporal synchrony with the live stage. Actors and the audiences at the secondary stages will experience the effects of delays.

Act-by-wire: We synchronize the clocks, and start the stages at the same time. The computers are continuously monitoring and measuring several metrics including delays between stages. If one or several videos are arriving late because of delays, the computers do on-demand manipulations and animations of the live video or substitute the live video with a pre-recorded video. If there are scripts available telling the system what each actor was meant to do, the system can create animations mimicking the expected movements. If there are no scripts telling the system what to do, it can try to predict the movements of an actor based on the most recent movements, or it can blur what is going on such that the audience not so easily notices the delays. In all cases, the goal is to create an illusion of interactive temporal causal synchrony.

4 Amplified Actor Interaction and Gestures

On a theater stage, with a significant physical distance between actors and the audience, bold makeup, clothes, and exaggerated movements are used to better project to the audience what the actors are doing.

In remote interactive performances there is a distance not only to an audience, but also between the actors. Consequently, the actors need their appearance, movement and gestures to be amplified such that they become easier to see and understand both for the other users and for the audience. In this way we extend the range of human interaction to remote locations and enrich the communication between them. We term this amplified interaction.

To be able to detect what an actor is doing, we must surround him with an interaction space [11]. An interaction space detects human movements, and analyzes them looking for gestures. A gesture represents a pre-defined command to the system to execute code to do some functionality.

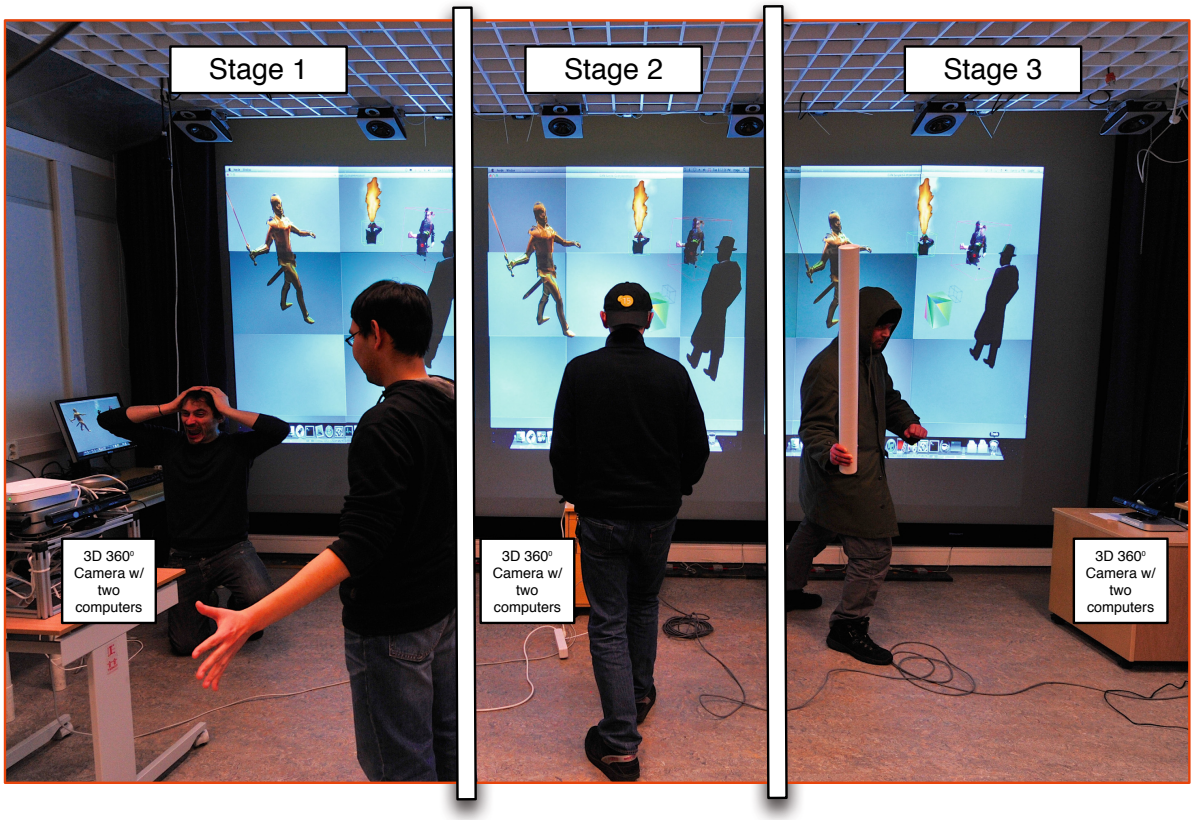


Fig. 2. To do experiments, MultiStage is set up with three stages and four actors in the same room. Each stage has its own camera rig. Each stage displays all actors. The global system binding together the stages are located either locally or on a remote computer across the Internet. Note: the flame animation has been enhanced in the figure for better visibility.

A gesture can be simple, like raising an arm, or complicated like doing two-arm movements. They can also be active like walking in a specific direction or passive as in standing still posturing. A collective (collaborative) gesture is a combination of the above kinds of gestures. Collective gestures can happen at the same stage, or be distributed, comprised of gestures from multiple stages. For example, when two actors at different stages, within some short timespan, raise their left arms above their head this can be interpreted as, say, a command to the system to animate a lightning between the two raised arms and display it on all the displays.

Based on the gestures we can create effects in the remote presence manifesting itself at remote rooms. A user's arm movement can in the remote presence be amplified by having a text bubble appear in the video, and by adding other visual effects to the representation of the user. The users remote presence can even be enhanced by executing a model of the user and using its output as the basis for the remote presence.

To experiment with the system, we set up three stages, named stage 1, 2 and 3, see figure 2, in a single room. There are two actors on stage 1, and one actor at each of the other two stages. Even if all three stages were co-located in

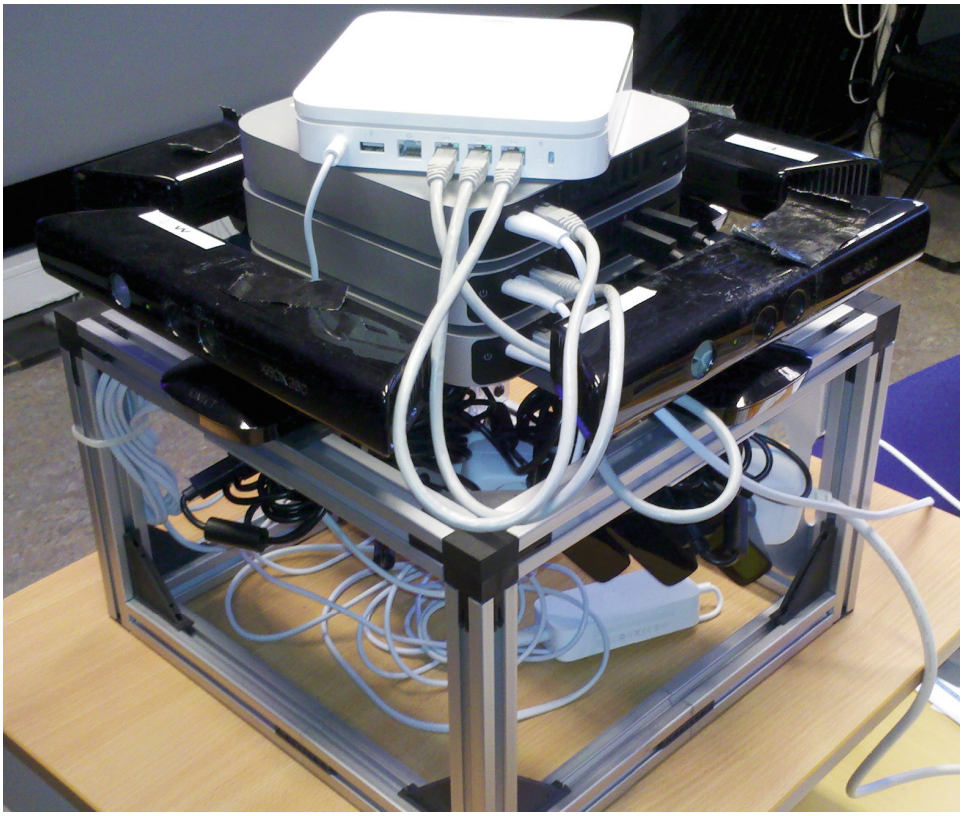


Fig. 3. The four 3D Kinect camera rig used at each stage for almost 360 degrees coverage

the same room they each occupied a different area of the room, and they each had their own interaction space and display. Each interaction space uses four Kinect 3D cameras, see figure 3. The cameras are arranged in a square with two computers receiving camera output and doing processing on the images. Four Kinects arranged in a square cover almost 360 degrees. We typically place the camera rig in the middle of a stage, and act around it. The room where the stages are located has a large 6m by 3m display wall. Each stage displays the remote presences of local and remote actors onto its assigned area of the display wall.

To simulate both the situation when all stages are on the same local network as well as when they are connected through a wide area network, the Internet, we locate the global side handling the distribution of data between the stages either locally at Tromsø or at a computer in Oslo or Copenhagen.

The images picked up by the cameras are analyzed and sent as data streams to all stages. This data represents the actors and to some degree what they are doing. The data is used to create a remote presence of each actors. This can take the form of a simple video, a manipulated video, or an animation of the actor as illustrated in the figure. Each stage has a display where the remote presence of each actors is displayed inside the same virtual stage.

On the virtual stage three of the actors have been amplified. At Stage 1 the kneeling actor with hands on his head is interpreted by the system as showing agitation, and the system has added an animated fire above his remote presence.

The other actor at Stage 1 does nothing the system recognizes, and a low resolution video of him is displayed at all stages. The actor at Stage 2 knows that if he keeps his hands in the pocket, has a hat on, and emulates walking, his remote presence will be that of an animated figure of a walking man with long dark coat and a hat. The actor at Stage 3 knows that if he has something looking like a sword in his right hand his remote presence will be that of a knight with a sword.

Presently, the prototype system cannot do all of the described functionality. The actual dynamic gesture and posture recognition is not yet in place. Consequently, the three amplified remote presences in Figure 2 were predetermined to be what they are.

5 Design and Implementation of Prototype

The design of the prototype, please see Figure 4, comprises several systems including the collaboration system, the human interaction system, the administrator interaction system, and an internal state & performance monitoring system.

The MultiStage system has a local side and a global side. The local side primarily focuses on what is happening locally on a stage. The systems implementing the local side executes on computers local to a stage. These systems include:

(i) the local detection system doing local state monitoring (LSM) recording what the cameras see, and doing on-the-fly local analysis (LSA) of the data to find interesting objects and events in the videos. The data is streamed to the global side for further analysis and distribution to the other stages.

(ii) the remote presence system subscribing to data streams from the other stages, and creating a remote presence of remote actors. Presently the primary remote presence technique is to visualize remote actors on a very large display per stage. In the future we may add physical devices like robots to the remote presence.

(iii) the human interaction system inform actors on when they should start actions, like moving arms, according to a given script. It will also in the future enable an actor to give gesture input to control a remote physical presence, like a robot and manipulating how the actor is displayed.

(iv) the temporal causal synchrony system applies the techniques discussed previously in this paper to reduce the effects of delays.

The global side is the glue binding the stages together, taking care of distribution of data between stages, and doing analytics needing data from multiple stages. The global side includes these systems:

(i) the administrator (or director) interaction system lets an administrator/director manage the systems, and setting start times for performances.

(ii) the global state detection system doing global state monitoring (GSM) collecting data from all the stages, and making it available for on-the-fly global state analysis (GSA) to detect distributed state like collective gestures and col-

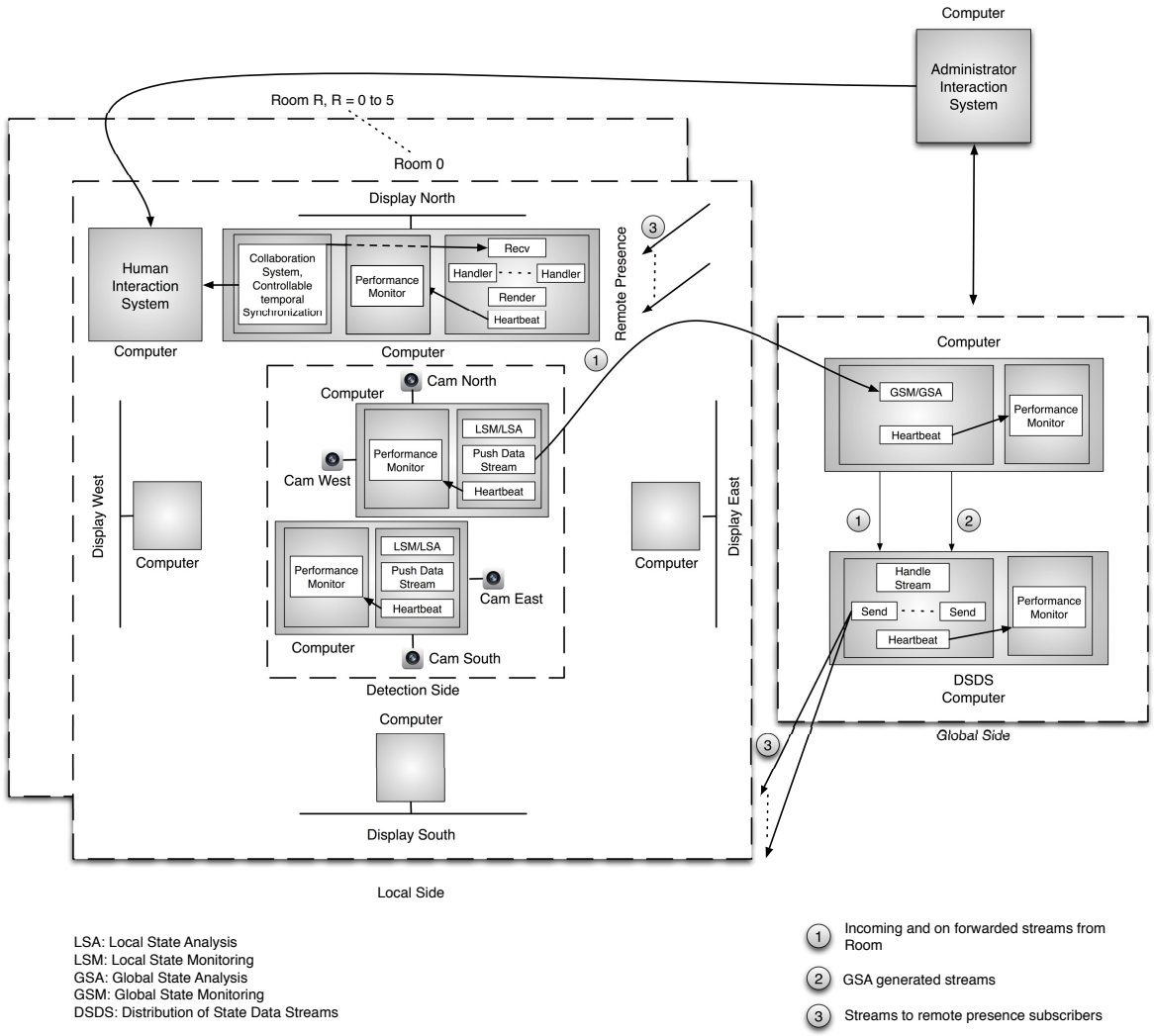


Fig. 4. The Design of MultiStage showing the systems at each stage and the global systems binding stages together

lisions when actors at different physical stages occupy the same volume on the virtual stage.

(iii) the distribution of state data streams (DSDS) system managing subscriptions from stages for data streams, and doing the actual transmitting of data to the remote presence computers locally to the stages.

Both the local and global side executes the internal state and performance monitoring system doing live performance measurements of several metrics including latency and bandwidth. These are made available to the global sides administrator interaction system. The performance measurements are also made available to the temporal causal synchrony system.

The systems were implemented on the operating systems Linux and Mac OS X and using several languages including C, Python and the Go programming language [12]. The animations and 3D models are rendered using the Horde3D graphical engine [13].

The prototype in Figure 2 can be configured to run on a variable number of computers. We typically have three to four computers per stage, two for the global side, and one computer for the administrator interaction system. With three stages the prototype comprises in total 12-15 computers. All computers can be connected through a combination of wireless network, switched gigabit Ethernet network and a wide area network (between Tromsø and Oslo (1500km)).

6 Evaluation

To characterize the performance of MultiStage a set of experiments were conducted. All computers used were modern Mac Minis at 2.7GHz. Each stage had three computers: two with two cameras each, and one with a large display. The global side had two computers: one for the global state monitoring and analysis, and one for the distribution of state data streams. Each stage and the global side had a network switch each. All switches were connected to a switch with access to the Internet.

For all experiments all stages were on the same 1Gbit/s switched Ethernet LAN inside the Department of Computer Science at the University of Tromsø. The DSDS, the system distributing data streams to the stages, was either on the same LAN as the stages, or located on a Planetlab [14] computer at the University of Oslo, 1500km away. In this case, all data sent between stages went from Tromsø to Oslo and back again. This separates the stages across the Internet.

Using the Python Psutil module [15], we measured the CPU utilization, amount of physical memory in use, and incoming and outgoing network traffic for all computers in use. We also measured three types of latencies: (i) the latency between the global side DSDS computer and the stages. We measured this by recording the time when we send a message from DSDS to a stage, and recording when a reply message comes back to DSDS; (ii) the end to end latency: the time it takes for a physical event happening on a stage to be picked up by the cameras and until a visualization of the actor is actually displayed on the same stage. We used a video camera with a high frame rate to record several videos of a user and the remote presence done on a display behind the user. We then counted frames to see how many frames it took from the user moved to the visualization caught up; (iii) the latency an actor can tolerate before the illusion of being on the same stage breaks. We subjectively decided this through two experiments. In the first we had an actor moving his arms while we observed him and his remote presence simultaneously. In software we artificially added a delay to the remote presence until we subjectively decided that the remote presence lagged too much behind to be mistaken for being on the same stage. In the second experiment an actor shaked hands with a remote actor. The delay between the actors was artificially increased until we subjectively decided that the handshake was not happening as fast as it would if the actors were physically on the same stage.

Factors in the experiments were the number of stages (1 to 3), the resolution of the images from the cameras (bounding box alone, 1000 to 5000 points per

image), the number of cameras per stage (0 to 4), and the location of the DSDS subsystem distributing data between stages (LAN in Tromsø vs. WAN to Oslo).

The results show that the resource usage in all cases are either very low or low. The implication is that the system is not resource limited. There is practically no loss of data in the experiments with the DSDS on the same LAN as the stages. When we separate the stages with a WAN by locating the DSDS on a computer in Oslo 1500km away, we see just an insignificant increase in data not getting across to all stages. The implication is that we can expect that the system typically will have satisfactory bandwidth available even when the stages are separated by the Internet.

When all stages and the global side were on the same LAN, the round-trip latencies were between 1-2ms. When the DSDS system was on a computer in Oslo the round-trip latencies were around 32ms. This matches well with measurements reported by PingER [16] for Europe.

On a LAN the end to end latency was between 90-125ms. With the DSDS at the computer in Oslo, the end to end latency was between 100-158ms. Two times the end to end latency, 200-316ms, is the delay that actors will experience from they do an action until they see a visualization of another actor reacting. We term this the **actor to actor latency**.

We subjectively decided that movements being delayed less than 100ms maintains the illusion of being on the same stage. However, the objective measurements show that an actor to actor latency is at typically 300ms. Consequently, the system should apply its techniques to mask the effects of the too long delay.

In the handshake experiment, we decided that an actor to actor latency of about 600ms was just acceptable and could be mistaken for how people shake hands when both are present in the same room. Longer delays bordered on creating a feeling that the remote actor was being obnoxious by delaying just a bit too long before responding to a hand shake. This indicates that the prototype is able to maintain the illusion of being on the same stage for hand-shake type of interactions.

The variation in latency we measured is because of several factors, including the distributed architecture of the prototype and the frame rate of the projector, video camera (240 fps) and the Kinects (30 fps), and other traffic on the LANs and WAN.

7 Conclusions

The subsystems and bindings between them makes for a complex actor collaboration system. While good programming practices will reduce the number of failures, a simpler system will provide for a higher probability of avoiding failures right before and during a performance. We will simplify based on the lessons learned from the prototype.

We believe that the built-in on-line monitoring of the state of the individual components of the system is important to discover where problems happen, and to help in fixing them. The on-line performance monitoring is critical for discovering delays long enough so that the system can try to mask their effect.

Having stages across the Internet is a challenge for the system because traffic load, failures and outages are mostly unknown before they happen. We have documented that the system scales to at least three stages with a total of at least 12 outgoing and 36 incoming data streams. Based on the performance measurements we conclude that the location of the data stream distribution server binding together the stages is not critical for the end to end latency of the system when it is used to do natural interaction, like handshakes, where delays of even 600ms is tolerable. However, when movements are meant to happen simultaneously and synchronized, the distribution server should be located where it provides for the lowest latencies. Data available in services like PingER [16] can help to choose a location to minimize latency between stages.

With regards to bandwidth, the location of the distribution server is presently not critical. This may change if the data streams grow in size and number. However, if the global analyzer and distribution sub-systems are located on computers on the same local area network as one or more of the stages, the Internet traffic is significantly reduced. This will penalize the other stages but could be useful for a performance with local audiences or where synchronized interactions are mostly among actors on the local stages.

Even if the system can do temporal synchrony and mask away delays, it is not yet clear how practical the system is in actual use. While we have not done formal user study experiments exploring the system capabilities with actors needing to tightly coordinate their movements, we have documented the performance limits of the MultiStage system. This provides for a sound prototype platform for experiments in a context of distributed performances with real actors.

Acknowledgment. We would like to thank Ken Arne Jensen for helping us build the camera rigs and make it look like something done by the creatures in Alien, Jon Ivar Kristiansen for help with the network and silently testing us by giving us a broken switch, and Maria Wulff-Hauglann for being brave enough to lend us shiny new Mac Minis so we could do a third stage.

This work was funded in part by the Norwegian Research Council, projects 187828, 159936/V30, 155550/420, and Tromsø Research Foundation (Tromsø Forskningsstiftelse).

References

1. Sawchuk, A., Chew, E., Zimmermann, R., Papadopoulos, C., Kyriakakis, C.: From remote media immersion to distributed immersive performance. In: Proceedings of the 2003 ACM SIGMM Workshop on Experiential Telepresence, pp. 110–120. ACM (2003)
2. Chew, E., Kyriakakis, C., Papadopoulos, C., Sawchuk, A., Zimmermann, R.: Distributed immersive performance: Enabling technologies for and analyses of remote performance and collaboration. In: NIME 2006 (2006)
3. Zimmermann, R., Chew, E., Ay, S., Pawar, M.: Distributed musical performances: Architecture and stream management. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP) 4(2), 14 (2008)

4. Sato, Y., Hashimoto, K., Shibata, Y.: A new remote camera work system for teleconference using a combination of omni-directional and network controlled cameras. In: 22nd International Conference on Advanced Information Networking and Applications, AINA 2008, pp. 502–508. IEEE (2008)
5. Tang, A., Pahud, M., Inkpen, K., Benko, H., Tang, J., Buxton, B.: Three's company: understanding communication channels in three-way distributed collaboration. In: Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, pp. 271–280. ACM (2010)
6. Sakamoto, D., Kanda, T., Ono, T., Ishiguro, H., Hagita, N.: Android as a telecommunication medium with a human-like presence. In: 2007 2nd ACM/IEEE International Conference on Human-Robot Interaction (HRI), pp. 193–200. ACM (2007)
7. Dou, M., Shi, Y., Frahm, J., Fuchs, H., Mauchly, B., Marathe, M.: Room-sized informal telepresence system. In: 2012 IEEE Virtual Reality Workshops (VR), pp. 15–18. IEEE (2012)
8. Petit, B., Lesage, J., Menier, C., Allard, J., Franco, J., Raffin, B., Boyer, E., Faure, F.: Multicamera real-time 3d modeling for telepresence and remote collaboration. International Journal of Digital Multimedia Broadcasting 2010 (2009)
9. Essid, S., Lin, X., Gowing, M., Kordelas, G., Aksay, A., Kelly, P., Fillon, T., Zhang, Q., Dielmann, A., Kitanovski, V., et al.: A multi-modal dance corpus for research into interaction between humans in virtual environments. Journal on Multimodal User Interfaces, 1–14 (2012)
10. Elgendi, M., Picon, F., Magnenat-Thalmann, N.: Real-time speed detection of hand gesture using kinect. In: Springer (ed.) Proceedings of the Autonomous Social Robots and Virtual Humans Workshop, 25th Annual Conference on Computer Animation and Social Agents (2012)
11. Stodle, D., Troyanskaya, O., Li, K., Anshus, O.: Tech-note: Device-free interaction spaces. In: IEEE Symposium on 3D User Interfaces, 3DUI 2009, pp. 39–42. IEEE (2009)
12. Go, `http://golang.org/`
13. Horde3d, `http://www.horde3d.org/`
14. Planetlab, `https://www.planet-lab.eu/`
15. Psutil, `http://code.google.com/p/psutil/`
16. PingER, `http://www-iepm.slac.stanford.edu/pinger/`

14.2 pVD - Personal Video Distribution

This paper was published in The Ninth International Conference on Wireless and Mobile Computing, Networking and Communications, WiMob 2013, Lyon, France, October 7-9, 2013.

2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)

pVD - Personal Video Distribution

Fei Su, John Markus Bjørndalen, Phuong Hoai Ha, Otto J. Anshus
Department of Computer Science
University of Tromsø, Norway
fei.su@uit.no, jmb@cs.uit.no, phuong@cs.uit.no, otto@cs.uit.no

***Abstract*—A user has several personal computers, including mobile phones, tablets, and laptops, and needs to watch live camera feeds from and videos stored at any of these computers at one or more of the others. Industry solutions designed for many users, computers, and videos can be complicated and slow to apply. The user must typically rely on a third party service or at least log in. The Personal Video Distribution (pVD) system supports sending and viewing live and stored videos between any of a single user's computers, and allows for a smooth hand-over of play back between computers. The system avoids any third parties, and relies only on the user's personal computers. We present the architecture, design and implementation of the pVD prototype. The architecture is comprised of functionality for sending videos, subscribing to videos, and maintaining the video play-back state. The design has a local side sending and viewing videos, and a global side coordinating the switching and distribution of videos, and maintaining subscriptions and video state. The prototype is primarily done in Python. A set of experiments was conducted to document the performance of the prototype. The results show that pVD global side has low CPU usage, and supports a handful of simultaneous exchanges of videos on a wireless network.**

Keywords*—*Personal Video Distribution; Video Playback; Mobile Video Streaming; Video Hand-off.

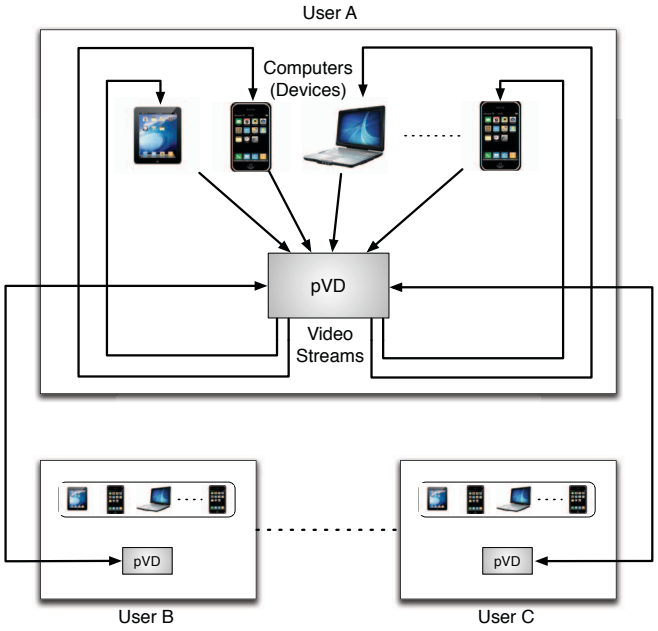


Fig. 1. The idea of the personal Video Distribution (pVD).

I. Introduction

Users today use multiple personal computers, including both mobile devices and larger displays. Many of these computers will have cameras that can be used to produce live video streams, and will have significant storage for videos. Live video from a camera connected to a computer can easily be watched on the same computer. This is also the case for videos stored on the computer. However, it is more cumbersome to do a smooth hand-over and watch video produced and stored at one of the user's computers at the others. It is also cumbersome to locate a video across computers because there is not a shared video name space. Consequently, videos at different computers can have the same name.

Existing industry approaches typically rely on a third party to let a user watch cameras and videos across computers. At the minimum, a log in to a subscription service is needed. As well as being dependent upon third party computers, an external network giving access to the Internet is also needed even when all video producing and consuming computers are local, say, at a user's home. This increases the probability for failures as well as cost and bandwidth usage.

Security and privacy are also issues users are concerned about when relying on third party services to store and service data [1], [2]. In [3] it is documented that people are more concerned about the privacy on mobile phones than laptops.

We report on the architecture, design and implementation of the Personal Video Distribution (pVD) prototype, allowing computers belonging to a single user to subscribe to cameras and videos from each other, see Figure 1. The system allows the live video from any camera to be viewed at any computer. When a stored video is played back or stored on one of the computers, it can be picked up by any of the computers. Play-back can be started from where in the video the user last stopped watching it. The pVD system does not rely on a third party service at all, using only a user's computers. When all computers are inside the same domain, say, at home, no Internet access is needed.

The usage model assumes that a user has physical access to all the computers. To watch, say, a smart phone's camera on a tablet, the user starts the pVD smart phone app and selects the camera as a video source. The app then starts streaming the video to the pVD system. On the tablet, the user starts the pVD app, inputs the smart phone's name and name of the live video, and a subscription is sent to the pVD system. The pVD app on the tablet now waits for videos to arrive according to the subscriptions. The pVD system matches incoming video streams with subscriptions, and streams the video to the tablet. While a single video can easily be streamed in and out of some smart phones and tablets, streaming of multiple videos are more easily supported using more powerful computers such

978-1-4799-0428-0/13/$31.00 ©2013 IEEE

as laptops and PCs.

The prototype is presently functional for a single user with multiple computers. We intend to extend the prototype to support multiple users on a single pVD system, e.g. a family, in the future. We briefly describe how multiple users with separate pVD systems can share cameras and videos without relying on a third party service.

Our contribution is to document a flexible and simple way to switch videos between a single user's computers by using only these computers, and without relying on a third party (cloud) service at all. We document how to do this through the architecture, design and implementation of an actual working prototype, and its performance characteristics. Several experiments have been conducted to measure how fast the pVD system responds to subscription requests, the CPU utilization of the part of the pVD system used by all the computers, and how pVD scales when the number of videos and computers increase.

II. Related Literature

There are many existing live video streaming services. PPStream [4] and PPTV [5] are systems for video distribution over the Internet using a combination of client/server and peer-to-peer approaches for distribution of videos. They maintain the state of a user on the user's computer, including which videos the user watched and where the user stopped the playback of a video. The user can later start a video from where it was stopped. Contrary to pVD, these systems are large-scale sharing many videos between with many users, and rely on every user having Internet access. In pVD, a computer also has the complete video so no peer-to-peer collaboration is needed. This reduces the complexity of the system.

YouTube is the world largest video sharing website from where people can upload, view and share videos. Live streaming is possible through services such as YouTube Live Streaming Events and Google+Hangouts. These systems are storing and sharing many videos between many users, while pVD shares just a few videos between a single user's computers. Users are dependent upon YouTube and Google as third parties outside of the users' control.

LiveCast [6] and Qik [7] enable live video streaming from users' mobile and other devices to any users or friends connected to the web. LiveCast is large scale with many users. It is feature rich, and meant to be used across Internet. Users are not dependent upon a third-party except their own organization or company. Qik is also on a large scale, enabling sharing between many users. The user must rely on Qik as a third party and store videos with Qik.

The Digital Living Network Alliance (DLNA) [9] uses Universal Plug and Play (UPnP) [8] for media management and media sharing between devices. Windows, Mac, Linux and Android also use the UPnP protocol to enable media sharing between devices. DLNA systems typically apply a media server and media players. In contrast, in pVD every computer is both a media server and a media player.

Apple AirPlay allows limited wireless streaming between Apple computers. The computers must be on the same subnet. While a video stream stored locally on one computer can be sent to another local computer, all computers must log into and interact with a third party, an Apple iTunes account.

A mobile live video learning system used for large-scale learning is described in [10]. Students can either attend a course in person or watch live and stored video streams sent from a server to mobile devices. The system does not maintain a user's video state to let play back resume at another computer.

Tele-TASK [11] is a tool for recording lectures and what happened at the presenter's computer, including presentation slides and software demonstrations. Users with mobile devices can watch the lectures everywhere.

CloudPP [12] is a Cloud-Based P2P Live Video Streaming Platform. It uses third-party cloud servers to construct a video delivery platform.

LiveShift [13] streams both live and stored videos. Live videos are streamed through a P2P network and peers store received videos for future playback. A user can view a stored video without local recording, and jump over boring parts to catch up the live video.

Eunsam Kim et al. [14], proposed an on-demand TV service architecture for a networked personal video recorder. This design reduces interactive operation response time and saves network bandwidth. The architecture includes origin servers, cache servers and Networked PVRs. The system serves both live videos and stored videos for playback.

Mobicast [15] is a mobile live video streaming system. Multiple users stream the same event from their devices. The streams can be stitched together, or the stream that has a best viewing angle is selected to provide a better collective viewing effect to viewers. If two users stream the same view, one of the streams can be stopped and later resumed to conserve battery power on the mobile device.

The systems mentioned above share to a large extent some characteristics. They are typically making a client dependent upon a third party outside of the client's control. They are intended for very many clients. We expect them to be rather complicated because they need P2P and other approaches to maintain good performance when the numbers of clients grow. While they all allow a user to playback videos, these videos are in most cases not meant to be on the user's computers. The ability to switch a live camera feed between a user's computers is only available in a few of the systems.

A significant characteristic of pVD is that a single user's computers subscribe to video streams from each other. Multiple subscriptions can be set up. However, a producer of a stream and the streams subscribers does not have to be running at the same time. When a stream starts streaming it can be picked up, and when no subscribers are running, the stream will go to the pVD system where it is buffered until a subscriber becomes present. pVD can also do hand-over of video streams between computers, letting a video start playing again from where it was stopped at another computer. The state of all videos are stored at and handled by the pVD global side server, and not by the pVD local side computers.

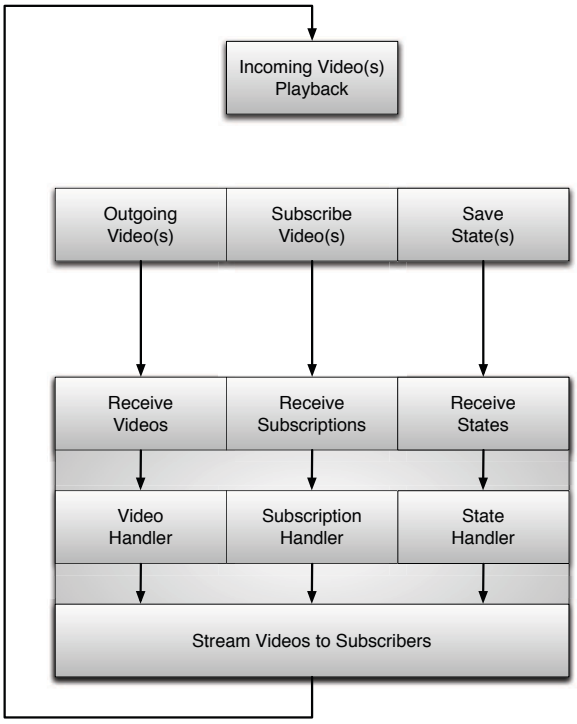


Fig. 2. The architecture of pVD.

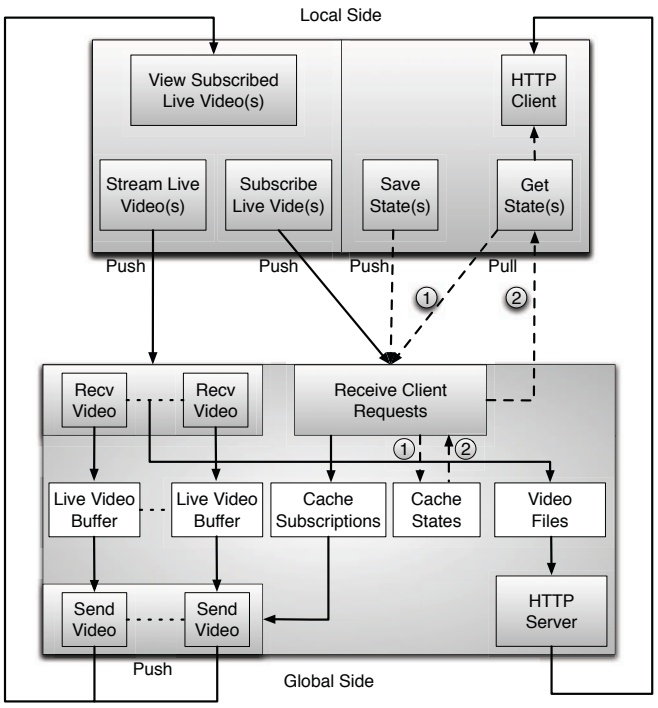


Fig. 3. The design of pVD.

III. Architecture

The architecture, shown in Figure 2, comprises functionality for handling in- and outgoing videos between a single user's computers. A subscription model for videos is used. To display a camera feed or a video at a computer, it must subscribe to the video. pVD allows a computer to send subscriptions to it. pVD will receive all subscriptions and use them to manage the switching of between computers.

A central part of the architecture is the distribution of live and stored videos to individual computers according to the subscriptions and the state of the videos. The functionality defined by the architecture includes streaming of outgoing videos, receiving incoming videos, buffering of video streams, and playback of videos.

Videos are handed over between computers by saving the current video play back state, including where in the video the user last stopped watching, to the pVD. When the video is streamed to a new computer, the state information is used to let a user continue watching the video on the new computer from where it left off at the other computer.

The architecture allows two users to provide videos from and to each other's computers. User A will call user B through some channel (say, telephone or chat) and gets the address (present prototype uses the IP address) to the global side pVD of user B (call it pVD B). User A will indicate this address when starting a subscription to a video on one of user B's computers. The subscription handler of pVD A uses the address to contact pVD B and registers the subscription with itself as the receiver. When user B starts streaming a video, pVD B will look at its subscription data and send the stream onwards to pVD A. In turn, pVD A will forward the video in a normal fashion to user A's computer.

To aid privacy, a user must from the user interface of the computer with the video explicitly acknowledge the streaming of the video to another user's computer each time a streaming is started.

IV. Design and Implementation

The design of the system is shown in Figure 3. The system is separated into a local and a global side. The local side executes on each device. It comprises a user interface that sends requests to start and stop subscriptions, starts play-back of incoming live and stored videos from other computers, streams outgoing live video from cameras and sends stored videos. It also keeps track of the necessary state for handing over videos between computers.

The global side executes on one of the user's computers, typically a PC or laptop, with the necessary resources to serve or communicate with the other devices (bandwidth, enough storage and memory). It is assumed to be always on and accessible to the other computers.

The local side pushes videos, subscriptions and state data to the global side. The global side receives incoming videos and data, and pushes out video streams to computers with subscriptions. The global side manages information about subscriptions and the state of live and stored videos.

The local side is concurrent to the degree supported by the operating system running on the computer. Some smart phone operating systems may have limited support for concurrency. The global side is designed as a concurrent system executing on a general-purpose operating system. This is done to make it simpler and more flexible, and to be able to benefit with regards to performance from multiple cores.

Each frame in of a live video includes the sending computer ID, video ID, frame counter, and a time stamp for when the frame was captured. A subscription message indicates the user ID, computer ID of the viewer, and video ID. There are three state related messages to save, get and remove where (frame number) in a given video a specific user and computer are at.

Video files are served by a HTTP server at the global side. The system was implemented using Python and Python OpenCV. It runs on Linux and Mac OS X.

V. Evaluation

To characterize the performance of pVD, a set of experiments was conducted using ten computers. All computers were 2011-2012 Mac Minis at 2.7 GHz, 8 GB of memory, and connected by wire to the same 1 Gbit/sec Ethernet switch. Six computers were used to represent a user's computers having videos and cameras. These executed the local side. One computer was used to run the global side pVD. Three computers were used to represent local side viewers that subscribed to the produced video streams.

We measured the **subscribe round-trip latency**: the time it took from the local pVD sent a request to start a subscription until it was received and processed by the global side pVD and an acknowledgement was received back at the local pVD. To measure the subscribe round-trip latency, the subscribing computer records the time when it requests a subscription, and the time when an acknowledgement arrives, and calculates the delay. We increased the number of computers from one to six. Each computer sent one or ten subscription requests. The subscription experiment uses TCP/IP as the transport protocol.

We measured the **video end-to-end latency**: the delay from something happens in front of the camera at one computer until it is visible on the display at a subscribing computer. To measure the video end-to-end latency, we set up one local pVD computer with a camera capturing a user, and another local pVD computer subscribing to the camera and displaying the camera output onto a display. We arranged the user and the display such that a high frame rate video camera could record both on the same video. We recorded several videos of the user and the display. We then counted frames to see how many frames it took from the user initiated a movement until the movement became visible at the display.

We measured **resource usage** of the global pVD computer and the participating local pVD computers. Using the Python psutil module [16], we measured the CPU utilization at the global pVD computer, and the incoming and outgoing network traffic for both it and each of the other computers.

Videos were represented by point clouds from two Microsoft Kinect cameras per local pVD computer. These were sent using UDP messages, resulting in about 13.5 Mbits/sec per camera, or about 26.7 Mbits/sec of data from each local pVD computer. This is equivalent to about four HD videos (4 to 8 Mbits/sec) from each computer to the global pVD computer. In the results, we report the number of HD stream equivalents.

To simulate local pVD viewers, we used 3 Mac Minis as viewers, with each viewer receiving a copy of every stream sent to the global pVD computer. We gradually increased the number of camera computers from two to six, increasing the number of video streams to the global pVD computer and the number of outgoing streams from the global pVD computer.

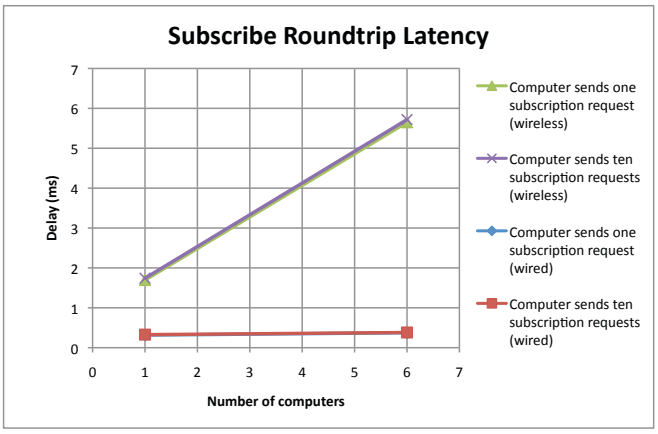


Fig. 4. Subscribe roundtrip latency on wireless and wired network. There is one subscriber per local pVD computer. Each subscriber sends one request in the first experiment and ten requests in the second experiment.

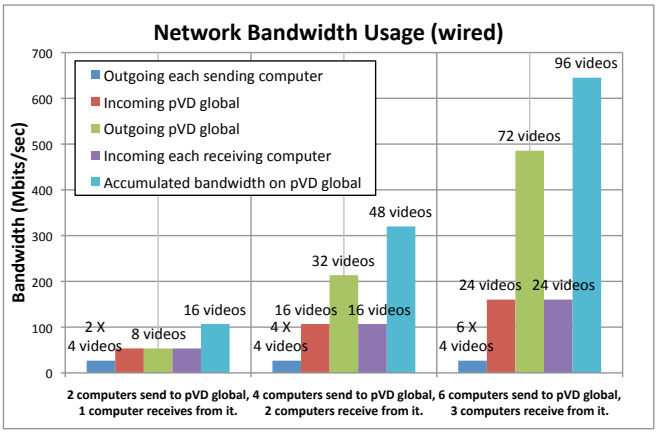


Fig. 5. Incoming and outgoing network bandwidth using wired connection.

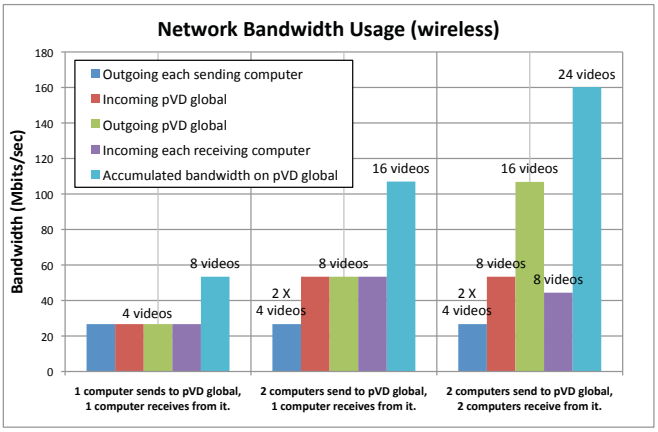


Fig. 6. Incoming and outgoing network bandwidth using wireless connection.

Figure 4 shows the subscribe round-trip latency. The round-trip latency is about 315 microseconds for one computer with one subscription request, and about 380 microseconds for six computers with ten subscription requests each. This is an insignificant increase. We conclude that the subscription

mechanism in the global pVD scales well with the number of computers and videos we expect a user to have.

The video end-to-end latency was between 90-125 ms. The variation in latency comes from several factors, including the distributed architecture of the prototype, the projector frame rate (120 fps), the video camera frame rate (240 fps), and that the Kinect (30 fps) can add 30 ms to the latency. We conclude that the video end-to-end latency is low enough to allow for interactive use.

In a study of latency [17], a 100 ms delay was noticeable by humans, but found acceptable. More than 200 ms delay made interaction uncomfortable. The sum of the subscribe message latency and the end-to-end latency is less than 200 ms. While at the borderline, the pVD system is able to stream live video events with latencies making it useful for interaction.

Figure 5 shows the network bandwidth usage at the computers involved and the number of HD stream equivalents in each experiment. With two local pVD computers, the global pVD receives four HD streams (26.7 Mbits/sec) from each computer resulting in a total of eight HD streams (53.4 Mbits/sec) on the network simultaneously. The local pVD viewer computer receives all of the eight video streams from the pVD global computer, resulting in a max load of sixteen HD videos in flight simultaneously on pVD global.

With six computers streaming to pVD global, we increased the number of pVD viewers to three. All of the viewers subscribe to every stream, so the global pVD sends out three times the incoming bandwidth (72 HD streams at 480 Mbits/sec). The total number of videos in flight simultaneously is 96 on pVD global. This pushes the system beyond an expected normal usage, but we have not observed any significant packet loss.

In summary, the accumulated bandwidth on pVD global with two senders and one viewer is 107 Mbits/sec, with four senders and two viewers is 320 Mbits/sec, and with six senders and three viewers is 645 Mbits/sec.

The CPU utilization on pVD global increases from 3.88 to 12.26% when it receives 8 to 24 videos and simultaneously sends 8 to 72 videos.

On a Gigabit network, the system can support in total 96 streams in the experiment. The CPU utilization is also less than 15% in this case. This is much more than the normal usage. We conclude that the results show the pVD global computer can easily be supported on even a low-end computer, and still have resources (like CPU or bandwidth) available for other applications and systems.

To characterize the impact a wireless network has upon pVD, we configured a system where the pVD global computer is connected by a wired 1 Gbit/sec Ethernet to an Apple Airport Extreme 802.11n (4th Generation) WiFi access point, and where the other computers use the WiFi network.

Figure 4 shows the subscribe roundtrip latency. The roundtrip latency is about 1.7 ms for one computer with one subscription request, and about 5.7 ms for six computers with ten subscription requests each. The video end-to-end latency was between 90-125 ms.

For the wireless configuration, figure 6 shows the network bandwidth usage and the number of HD stream equivalents in each experiment. With two computers streaming to and one computer receiving from the pVD global computer, eight videos were sent to and fully received at the receiving computer. The accumulated bandwidth at pVD global computer was 107 Mbits/sec. The receiving computer received 53.4Mbit/sec. When a second receiver was added, for a total of two receivers, each received only 44Mbit/sec instead of 53.4Mbit/sec. We believe the reduced bandwidth can be removed by a more modern wireless network with better performance and resistance to interference from other nearby wireless networks. However, the experiment shows that it is possible to wirelessly stream at least eight HD videos to the pVD global computer and to wirelessly receive at least eight HD videos from the pVD global computer.

VI. Discussion

pVD is based around a manual approach to controlling both video switching and privacy. A user must have access to all computers serving and consuming the videos. A user interacts directly with the pVD user interface on the sending and receiving computers. A user is the glue to bind together computers. When videos are sent between users a sending user must manually accept a one-time streaming of a video to another user. The sending user can at any time halt the streaming. This provides for some control of the privacy for the sending user. To strengthen the privacy we could have added techniques like time-outs for video streams, halting them automatically when the time-out occurs. However, this adds complexity, and we wanted to keep the pVD system as rudimentary as we could within reasons. The pVD system overall uses a simple and robust approach customized for a single user with a handful of computers. However, it does not extend and scale to many computers and to many users, and was not meant to do so.

pVD can do hand-over of video streams between computers, letting a video start playing again from where it was stopped at another computer. The state of all videos are stored at and handled by the pVD global side server, and not by the pVD local side computers. Because pVD is meant for a single user with just a handful of computers, there are no performance issues in doing this centralized. It also aids in doing hand-over of videos between computers by having the state of the videos at one place. However, if the pVD global looses state about the videos, the user must recreate it. We don't expect this to be an issue because of the usage domain with just a single user and a handful of computers.

The approach to share live and stored videos between multiple users will not scale to many users. It is intended to let a few users share live and stored videos in a case-by-case fashion. Users need to talk to each other to exchange enough information to connect. This can be automated and made more efficient, but we wanted to keep it basic and simple, and to involve the users in the sharing to reduce unintended sharing. While the multiple user approach demands users interaction, we still believe it is useful for simple ad hoc interaction and sharing between family and friends.

We have deliberately used a Gigabit wired Ethernet for some of the experiments where the goal was to measure the

performance behavior of the pVD global. A typical usage scenario is to have the pVD global computer connected by wire to a wireless access point and has a user's computers share videos with each other through a wireless network. A single 8 Mbits/sec HD video stream will in this set-up at the worst consume about 16 Mbits/sec of the wireless network. Wireless networks typically range from 54 Mbits/sec to 300 Mbits/sec. A wireless network should in practice be able to support a pVD configuration with a handful of computers and video streams. For the intended usage domain this is enough. Emerging wireless networks like the 802.11ac technology [18] can achieve 1300 Mbps and should together with future computers allow for even better performance for pVD.

VII. Conclusions

In this paper, we present a personal video distribution system for simple ad hoc sharing of live (camera) and stored video streams between a single user's computer.

The pVD system uses subscriptions to bind together a video and camera on one computer with the other computers. The user must have physical access to each computer to set up a subscription, and to start the streaming and receiving of a video. When a sender and a receiver run simultaneously, videos will start to flow according to the subscriptions. The system saves video state and allows the user to continue watching a video on another device, picking up from the same position in the video.

The system avoids issues with relying on third parties and access rights by being designed for very small scale and to be run only on a user's personal computers. The system can also be kept simple from not having to scale to support a large number of videos, computers and users.

The bandwidth is not a critical issue for the pVD prototype system and the system does not occupy too much system resources, like CPU. When all computers are on a Gigabit wired network, the system can support many simultaneous streams using a standard PC desktop for the pVD global side. The system also supports a handful of simultaneous streams on a wireless network. The system responses fast enough to user's requests and the latency for streaming live events will be noticed, but accepted by the user.

Acknowledgment

Many thanks to the technical staff at the department. This work was funded in part by the Norwegian Research Council, projects 187828, 159936/V30, 155550/420, and Tromsø Research Foundation (Tromsø Forskningsstiftelse).

References

[1] S. M. Habib, S. Ries, and M. Muhlhauser, "Cloud computing landscape and research challenges regarding trust and reputation," in *Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2010 7th International Conference on*. IEEE, 2010, pp. 410–415.

[2] I. Ion, N. Sachdeva, P. Kumaraguru, and S. Čapkun, "Home is safer than the cloud!: privacy concerns for consumer cloud storage," in *Proceedings of the Seventh Symposium on Usable Privacy and Security*. ACM, 2011, p. 13.

[3] E. Chin, A. P. Felt, V. Sekar, and D. Wagner, "Measuring user confidence in smartphone security and privacy," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*. ACM, 2012, p. 1.

[4] [Online]. Available: http://www.pps.tv/

[5] [Online]. Available: http://www.pptv.com/

[6] [Online]. Available: http://www.livecast.com/

[7] [Online]. Available: http://qik.com/

[8] [Online]. Available: http://www.upnp.org/

[9] [Online]. Available: http://www.dlna.org/

[10] C. Ullrich, R. Shen, R. Tong, and X. Tan, "A mobile live video learning system for large-scale learning—system design and evaluation," *Learning Technologies, IEEE Transactions on*, vol. 3, no. 1, pp. 6–17, 2010.

[11] K. Wolf, S. Linckels, and C. Meinel, "Teleteaching anywhere solution kit(tele-task) goes mobile," in *User Services Conference: Proceedings of the 35 th annual ACM SIGUCCS conference on User services*, vol. 7, no. 10, 2007, pp. 366–371.

[12] H.-Y. Chang, Y.-Y. Shih, and Y.-W. Lin, "Cloudpp: A novel cloud-based p2p live video streaming platform with svc technology," in *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*, vol. 1. IEEE, 2012, pp. 64–68.

[13] F. V. Hecht, T. Bocek, R. G. Clegg, R. Landa, D. Hausheer, and B. Stiller, "Liveshift: Mesh-pull live and time-shifted p2p video streaming," in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*. IEEE, 2011, pp. 315–323.

[14] E. Kim and C. Lee, "An on-demand tv service architecture for networked home appliances," *Communications Magazine, IEEE*, vol. 46, no. 12, pp. 56–63, 2008.

[15] A. Kaheel, M. El-Saban, M. Refaat, and M. Ezz, "Mobicast: a system for collaborative event casting using mobile phones," in *Proceedings of the 8th International Conference on Mobile and Ubiquitous Multimedia*. ACM, 2009, p. 7.

[16] [Online]. Available: http://code.google.com/p/psutil/

[17] A. Pavlovych and W. Stuerzlinger, "Target following performance in the presence of latency, jitter, and signal dropouts," in *Proceedings of Graphics Interface 2011*. Canadian Human-Computer Communications Society, 2011, pp. 33–40.

[18] [Online]. Available: http://www.apple.com/airport-extreme/

14.3 Masking the Effects of Delays in Human-to-Human Remote Interaction

This paper was published in Federated Conference on Computer Science and Information Systems, FedCSIS 2014, Warsaw, Poland, September 7-10, 2014.

Preprints of the Federated Conference on
Computer Science and Information Systems pp. 725–734

Masking the Effects of Delays in Human-to-Human Remote Interaction

Fei Su, John Markus Bjørndalen, Phuong Hoai Ha, Otto J. Anshus
Department of Computer Science
UiT The Arctic University of Norway
fei.su@uit.no, jmb@cs.uit.no, phuong@cs.uit.no, otto@cs.uit.no

***Abstract*—Humans can interact remotely with each other through computers. Systems supporting this include teleconferencing, games and virtual environments. There are delays from when a human does an action until it is reflected remotely. When delays are too large, they will result in inconsistencies in what the state of the interaction is as seen by each participant. The delays can be reduced, but they cannot be removed. When delays become too large the effects they create on the human-to-human remote interaction can be partially masked to achieve an illusion of insignificant delays. The MultiStage system is a human-to-human interaction system meant to be used by actors at remote stages creating a common virtual stage. Each actor is remotely represented by a remote presence created based on a stream of data continuously recorded about the actor and being sent to all stages. We in particular report on the subsystem of MultiStage masking the effects of delays. The most advanced masking approach is done by having each stage continuously look for late data, and when masking is determined to be needed, the system switches from using a live stream to a pre-recorded video of an actor. The system can also use a computable model of an actor creating a remote presence substituting for the live stream. The present prototype uses a simple human skeleton model.**

***Index Terms*—Effects of Latency; Mask the effects of delays; Temporal Casual Synchrony; Remote Interaction.**

I. Introduction

IN DISTRIBUTED acting, actors at different stages, physically separated by distance, interact to create a coherent play. The interaction can be lazy, allowing for large delays without breaking the illusion of being at the same stage. This is, for example, the situation when actors do a relaxed handshake, or don't interact directly at all. The interaction can also be eager, where even small delays break the illusion. This is, for example, the case when actors do fast action/reaction with causally related movements between each other, or move in synchrony as done in dancing.

Fig. 1 depicts distributed acting. Three stages, in Tromsø, Porto, and Florence, have a total of four actors doing eager interaction, dancing together. In Tromsø, there are two actors physically present, while there is one actor in Porto and one in Florence. At each stage, each actor is represented by a remote presence in the form of an independent streaming video.

Distributed acting is complicated by each stage having a different clock, and by communication delays and jitter. The clock at each stage can easily be sufficiently synchronized with a reference clock, but delays and jitter are unavoidable and are the result of the finite speed of light, and of the technologies and systems applied to create a distributed stage gluing together the individual stages.

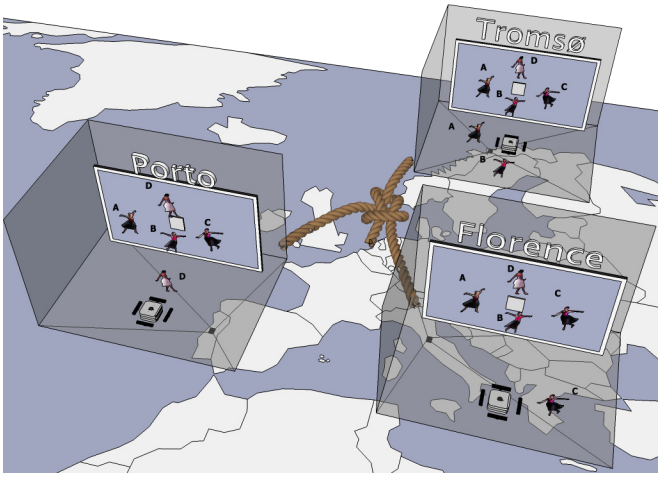


Fig. 1. Four dancers at different stages dance together. Each stage is equipped with sensors to detect actors and a display to visualize the remote presence of all the performers.

The speed of light defines the lower bound of a non-zero delay from an event happens until it can be observed. Table I shows the time needed for light to travel distances that may be typical in distributed acting. It takes about 3 microseconds between buildings, 30 milliseconds between cities and about 134 milliseconds around Earth's equator. The time it takes for light to travel from an actor to another and back is twice this amount of time. However, the actual delays experienced by actors interacting through a computer-based system are even higher.

TABLE I
Travel time at the speed of light

1 km	3.3 μs	Between buildings
1000 km	33 ms	Between cities
4000 km	134 ms	Around equator
2.4 x 10^{19} km	2.5M years	To Andromeda Galaxy

Figure 2 describes the total delay when observing a remote event. Delays are created by the sensors tracking actors, transfer of data from sensors to computers, processing of the sensor input, network transmission, on-route processing,

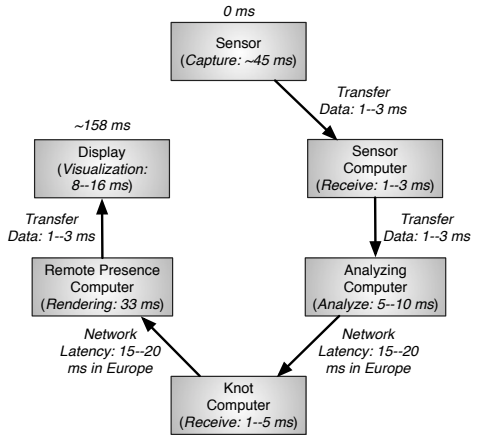


Fig. 2. Every Phase will add delay

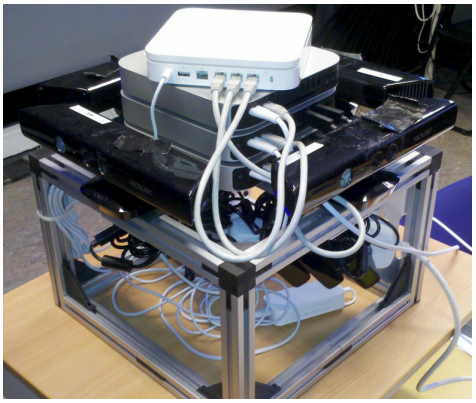


Fig. 3. The 360 degrees actor detection sensor rig comprising four 3D Kinect cameras, two Mac Mini computers, and wireless access point. One per stage is used.

receiving and processing the received data, and preparing and visualizing the data locally. The delays can be significantly larger than what is indicated in the figure if more processing is applied. These delays can be reduced and partially masked, but they can never be removed.

Delays are important when people interact. It has been documented [1], [2], [3], [4] that people accept delays below 200ms as insignificant when interacting tightly. When the delays grow beyond 200ms they become harder and harder to ignore, and actors can be expected to have problems interacting as if they were on the same physical stage.

The goal of the MultiStage system [5] is to aid actors at stages around the world in interacting with each other as if they were on the same stage. Each stage has a set of sensors, shown in figure 3, detecting and tracking the movements of the actors on the stage. The actors at the other stages are each represented by a remote presence. A remote presence is based upon having data about an actor available such that the actor's movements can be recreated remotely. A simple case is to have data representing a streaming video of the actor, and show it on a large display to visualize the actor in full scale. A more advanced case is when an actor's movements are used as input into a computation creating a remote presence of the actor. The remote presence can be visualized on a display or control a robot.

Several experiments were conducted to determine the objective and subjective performance of the system. Objective metrics include the delays in different parts of the prototype system, and processing and network resource usage. Subjective metrics include how much delay an actor will notice and tolerate when interacting, and when an actor experience that the switching of the masking in and out is smooth.

II. Masking Approaches

In [5], we define **loose temporal causal synchrony** to be when actions by actors happen causally in the correct order, but with no special demands on delays. **Interactive temporal causal synchrony** is when actions by an actor is seen in causal order and with delays as actors are used to when being on the same stage face to face. To achieve this even with delays and jitter being unavoidable, the idea is to mask the effects of delays as seen by the actors.

In the **Act-By-Actor**-approach, the actors react to the remote presences as if they were the actual actors. How the interaction looks and how it feels to actors and audiences depends on how large the delays are, by how much they vary, and by how good the actors are at compensating.

In the **Act-By-Director**-approach, a director keeps time and tells actors when to do actions according to a shared script or to a script for each actor. Even if the actors act on command it will seem to an audience as if they interact freely with each other.

A variant is to select a stage to be the **live** stage. The others are secondary stages. The start time for a performance at a secondary stage is the start time for the live stage minus the delay between them. Consequently, performances at secondary stages are started a little earlier than at the live stage such that when the live stage starts, the input from the secondary stages arrive. At the live stage the actors and an audience will experience a performance where local actors are in synchrony with the remote presences representing the remote actors. However, actors at a secondary stage will be out of sync with the remote presences. By switching which stage is the live stage at suitable points in the performance, each stage can be the live stage for a time.

A second variant of this approach is to **delay each local remote presence** at a stage. A local remote presence is the remote presence of an actor shown and heard at the stage where the physical actor also is. The effect is that an actor and an audience will experience a local and a remote event at the same time because they have both been delayed equally much. To make this approach practical, the delay cannot be so high as to make the actors and audience noticing it too much. Because delays between stages in practice tend to be different, this approach is most practical for just two stages with about equal delay between them.

A third variant is to **delay all remote presences** at a stage until data for the slowest remote presence arrives. With varying delays between the stages, they will soon be out of synchronization with each other. However, the local and remote presences at a stage will be in synchronization with each other. The delay waiting for the slowest can be long enough to be noticeable for actors and an audience. Consequently, the actors at a stage can be out of synchronization with the remote presences.

In the **Act-By-Wire**-approach, remote presences are manipulated to hide the effects of delays when delays reach predefined threshold values. Manipulations include just-in-time blending in of prerecorded videos of remote presences of actors, and just-in-time blending in of on-demand computed remote presences. A prerecorded and an on-demand computed remote presence will to a varying degree succeed in creating the illusion of low insignificant delays. If there is a script of what an actor should do at a given time, then a prerecorded remote presence can be created and played back at the correct time when delays go too high. When instead of using a static pre-recorded video a computation is run to create the remote presence, a wide range of possibilities are in principle available. These include blurring the movements of an actor such that delays are not so obvious, and predicting what an actor was going to do. We have not explored these possibilities yet.

III. Related Literature

Several systems try to enable interaction between local and remote users. The Distributed Immersive Performance (DIP) [6] and [7], is a multi-site interaction and collaboration system for interactive musical performances. In experiments, they artificially delayed the local stage and found out that (i) the tolerable latency for slow paced music is much higher than for fast paced music; (ii) to help performers pick up aural cues it is better having a low audio latency than synchronizing video and audio; and (iii) a roundtrip video delay of more than 230ms makes synchronization hard for the users. In [8], a series of experiments on the DIP system is described with focus on the audio delay, and how the delay affects musician's cooperation. An artificial delay of 50ms to the remote room's audio stream was tolerable. With the same latency added at both rooms it became possible to play easily together with a delay of up to 65ms. While they report on the effects of delays on audios, we report on the effects of delays on videos, and how they can be masked.

Other distributed collaboration systems include [9], [10], and [11]. These do not consider the effects of delays and how to mask them when users interact across distance.

Several techniques [12], [13], [14], [15] and [16], exist to reduce or hide network latency in network games and in distributed systems. The Dead-Reckoning (DR) technique is used in distributed simulations and to hide latency mostly in network games. Computers that own an entity will send unique information about the entity to other computers on the network. The information includes the position, velocity, and acceleration of the entity or more. Each computer simulates the movement of the entity. The computer which owns the entity will also simulate the entity as well as check the real state of the entity. When the simulated value and real value differs more than a threshold, the computer will send update information to the other computers. The dead-reckoning technique is a general way to decrease the amount of messages communicated among the participants.

IDMaps [17] measures the distance information on the Internet. This is used to predict latencies. King [18] uses recursive DNS queries to predict latency between arbitrary end hosts. In [19] a structural approach to latency prediction technique based on Internet's routing topology is proposed. In [20] the network latency is reduced based on estimates of the network path quality between end points. These approaches can be useful even if we don't mask latencies themselves, but the effects of delays. Predicting the very near future latency can be useful because we can start the masking right before large delays happen. The Local-Lag (LL) technique [21], provides for better fairness between local and remote players by making all see approximately the same delays. A local operation is delayed for a short time. During this short time period the operation is transmitted to remote computers participating in the game, and all computers can then execute the operation closer in time to each other. However, with more than two participants seeing significantly different latencies, the fairness cannot be maintained for all computers. In [22] and [23], the LL is integrated with DR to synchronize participants and keep better consistency among all computers.

In [14] and [22], some of the drawbacks of the above mentioned DR and LL techniques are identified. While the LL technique ensures fairness for two players, or for multiple with the same latencies between them, the fairness is not preserved when the latencies become too different. The same is the case for the DR approach because when a computer does an update, the time it takes to have data about this delivered at the other computers will vary depending on the latencies between the local computer and each of the other computers. This can result in a situation where a local player and some of the remote players can do actions earlier than other remote players.

Even if it is worthwhile to reduce network latencies and other delays, and do overlapping between communications and processing, delays cannot be removed. In this paper, we present several techniques to mask the effects of delays, and we also measure the cost of applying each technique.

There are several projects which have studied the effect of latency when remote users interact, including [24], [25], [2], [26], [4], [3], and [27]. When the latency from a user does an action until it is reflected in, say, a game, is more than 200ms, the user will notice the delay and his actions and scores are impacted by it. In a first person shooter game there is a 35% drop in shooting accuracy at 100ms of latency, and the accuracy drops sharply when the latency increases further. More than 200ms of latency should be avoided. For some sports and role-playing games a latency of 500ms can

be acceptable. Consequently, latency reduction and hiding techniques should aim at achieving end-to-end latencies less or equal to these numbers. When this cannot be achieved, then masking the effects of the various delays becomes interesting to apply as well.

In [28], a comparison is made between the end-to-end latency of an immersive virtual environment and a video conferencing system. The tolerable latency for verbal communication was found to be 150 ms. This was achieved by the teleconferencing system, but not the virtual environment system. A video was done capturing a person repeatedly moving an arm up and down. A video was also done of the same person as represented by the system. Synchronized cameras were used to be able to synchronize the two videos. The latency from the person moved an arm until it was reflected through the system was measured to be 100-120ms for the teleconferencing system, and 220-260ms for the virtual environment when the avatar for the user had been preloaded.

In [29] several techniques were used to reduce the latency for the head tracking system of an immersive simulation system. The techniques included disabling buffering and having a more direct path to the tracker hardware. This results in an almost 50% reduction in latency, from around 90ms to around 50ms.

Packet jitter [30] is the variation in the packet delay. Variations in packet size, buffer delay, and routing create packet jitter. The influence of the jitter in games is measured in [26], [31], [32], and [33]. They conclude that jitter had only a minor impact on the win probability, the scores and the user experience. However, when jitter increases, the tracking accuracy of a target, the users ability to keep a small and consistent distance between the center of the target and the cursor, declines.

In [34] they consider unfairness created by the cumulated errors between players. The system improves fairness by equalizing for all players, the errors of where an object of the game is placed and what it is doing. This resulted in a significant improvement in consistency between what players observed even for 100ms of delay between players at different computers.

IV. System Overview

Figure 4 shows the MultiStage system. The design and implementation is described in detail in [5].

The system is divided into a local and a global side. The local side of MultiStage primarily focuses on what is happening locally on a single stage. The global side is the glue binding stages together, taking care of distribution of data between stages, and doing analytics needing data from multiple stages.

The local stage monitoring (LSM) system detects local state at the stage, including actors and their movements, and streams it to the local stage analysis (LSA) system. The LSA analyzes the data to detect gestures (not expanded on in this paper), and forwards the data and data about detected gestures to the global side.

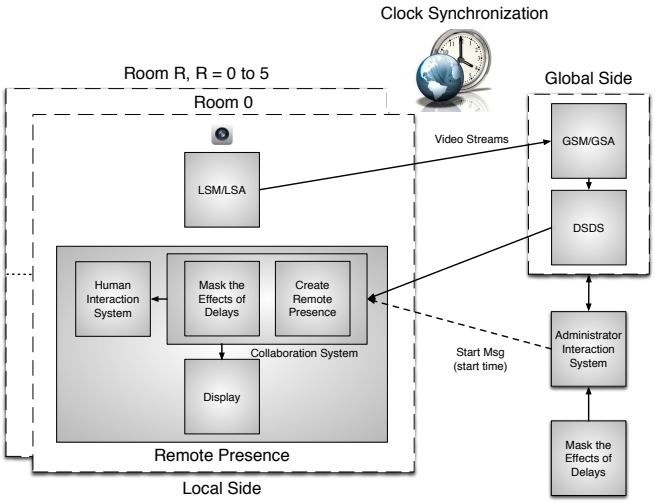


Fig. 4. The MultiStage system.

The LSM system produces an individual stream for each actor. This allows for great flexibility in treating each actor individually when looking for gestures, and where each actor's remote presence is manifested and located in relation to the others on the stages. In the present prototype, an individual stream for each actor is achieved by using a Kinect camera per actor. The system assumes that just a single actor is within the 3D field of view of the camera. All objects outside of this 3D space are ignored. The advantage of having a one-to-one relationship between actors and cameras is that it takes very little processing to create individual streams for the actors. This helps in reducing the delay from an actor moves until it is manifested in the remote presence at remote stages. The disadvantage is that when the number of actors increases, so must the number of cameras. Presently the prototype supports four actors per stage using four Kinect cameras arranged back to back. The back to back configuration avoids having the infrared dot cloud used by the cameras (to achieve depth information) to interfere with each other. While more Kinect cameras can be used, care must be taken to avoid interference. A more advanced sensor suite will help avoid this problem.

The remote presence system at each stage subscribes to streams from the global side. The data is used to locally create remote presences. In the prototype, remote presences are visualized on a big display. The visualization of a remote presence can take three forms. It can be 2D streaming videos based on color images captured by four Kinect cameras at each stage. Alternatively, 3D point streaming cloud videos can be used. These are created using color and depth images captured by the Kinect cameras. Finally, a remote presence can be visualized as an animated human skeleton created locally at each stage.

The LSM uses the Kinect cameras to sense actor movements. In principle, if the LSA identifies actor body movements, the data about this makes its way to the remote pres-

ence, and the computed human skeleton moves accordingly. In the present prototype, a script defining what each actor should do is used. When delays become to high, the human skeleton remote presence computation for an actor receives commands taken from the script. These commands are typically of the type "raise left arm" and "lower right arm". Computing a model of a human skeleton locally, and letting it react to just streaming movement commands, saves network bandwidth vs. distributing streaming videos.

The remote presence system includes the masking system. It looks for incoming data about remote actors being too delayed to do remote presences without the local actors noticing the delay. If the delays are too large, the masking system applies several techniques to mask the effects of the delays as seen by the actors. A limited form of masking is also done outside of the remote presence system. In this case the masking system provides information to the administrator interaction system (see below) such that it can tell the human interaction system at each stage what to do, like individual delayed start-up times of a performance for each stage.

The human interaction system at each stage informs actors what to do, and provides them a countdown for when they should start doing it. In the prototype, a display per stage is used to visualize this for the actors.

The global side monitoring (GSM) receives data streams from the local stages. It forwards the data to the global side analysis (GSA) system. The GSA system does analytics on the data streaming in from the stages looking for global state. An interesting global state is a collective gesture. It is comprised of several gestures done by several actors possibly at different stages. The idea is that when a given number of actors have done a certain gestures, this should result in actions taken at the stages, like, say, turning on a light or doing some modifications to the remote presences.

The GSA system forwards all data and information about global gestures to the distributed state distribution system (DSDS). The DSDS manages subscriptions from the remote presence system at each stage, and deliver streams to the subscribers.

The administrator interaction system lets a director manage the system, including setting and distributing to all stages the start time of a performance. Each computer in the system has a performance monitor measuring several metrics including latency between the computers and bandwidth. These measurements are made available to the administrator interaction system.

The sub-systems implementing the local side executes on computers local to a stage. This is done to achieve low local latencies, and reduce network bandwidth. It also distributes the global workload, and isolates the stages such that if one stage fails, the other stages have a higher probability of not being affected. The sub-systems implementing the global side executes on computers that are located relative to the stages to achieve high bandwidth and low latencies. The administrator interaction system is located on a computer which is convenient to use by an director.

Multistage is a distributed system, and the computers can have different clock values. The system assumes that all computers have the same time, and the clocks are therefore synchronized.

Experiments measuring the performance of the prototype have been done both with all stages locally on the same local area network, as well as kept more than 1500 km apart (Tromsø to Oslo and back). The system currently scales across the Internet with good performance to three stages, and comprises in total 15 computers, 12 cameras, and at least 12 outgoing and 36 incoming data streams.

The system was primarily implemented in Python. The OpenKinect Libfreenect library is used to fetch RGB and depth images from the cameras. The LSA motion detection using Python OpenCV is taken from Robin David on GitHub [35]. The human skeleton model is implemented in Python, using Pygame. Pygame is used to display the actor script. Python Tkinter module is used to display the Administrator Interaction System. The system runs on Linux and Mac OS X.

V. Design and Implementation of Masking the Effects of Delays

To do masking, several functionalities must be realized at each stage. A **shared clock** is assumed by the system. This is achieved with sufficient accuracy by using Network Time Protocol (NTP) [36] to set the local clocks. A **performance monitor** measures and computes the communication delays between all computers. To do so, every packet sent is time stamped. It also measures the clock differences between the computers at a stage and the DSDS distribution computer to determine if clock synchronization is needed to maintain the shared clock. The performance monitor is present at every computer of the system.

A **shared and individual performance start-times** are distributed by using the administrator interaction system to send a message with the performance start time to each stage. We assume that when needed there are predefined **actor scripts** available telling each actor what and when to do an action. In the prototype a display at each stage shows a count down until the next action is to be done, and visualizes with a simple drawing what the action is.

The following masking approaches are shown in figure 5. For all approaches we assume that the stages have already initiated subscriptions to data streams from each other, and that the streaming is in effect.

Live Stage: The administrator interaction system uses the performance monitor to measure the latency from the detection computer at each secondary stage to the distribution server. It also measures the latency from the distribution server to the remote presence computer at the live stage. The effective latency from a secondary stage to the live stage is the sum of these two latencies. A secondary stage's performance start time is the start time at the live stage minus the latency between the live and the secondary stage.

The administrator interaction system now sends a message to each stage with the start time of the performance and

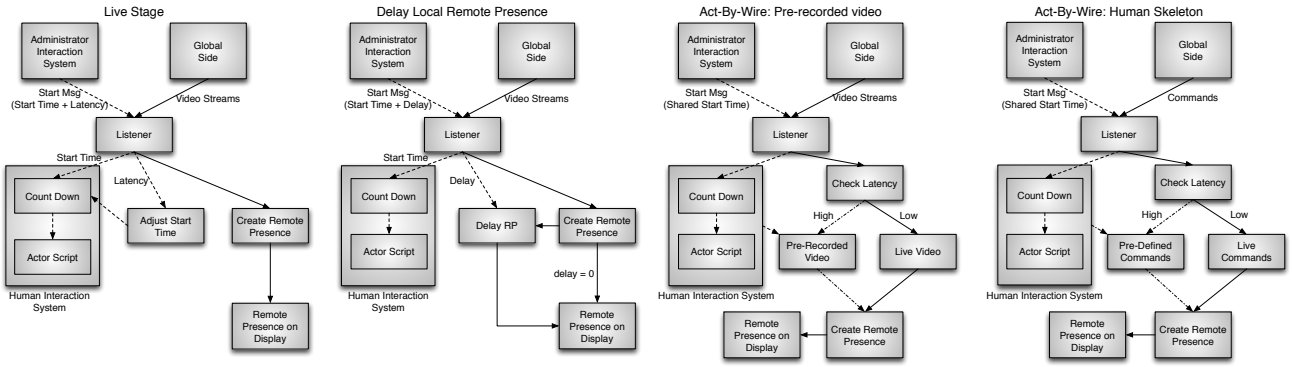


Fig. 5. Design and Implementation of the techniques to mask the effects of delays

the latency that should be decreased to the start time for that particular stage. The human interaction system at each secondary stage will now do a countdown with the start time of the live stage modified by the latency to the live stage. When the countdown ends, a visualization of what each actor should do is displayed. The human interaction system now acts as a director, counting-down to the next action of each actor, and then visualizing the action.

Delay Local Remote Presences: The administrator interaction system uses the performance monitor to measure the delay from the detection computer at each stage to the distribution server. It also measures the delay from the distribution server to the remote presence computer at the stage. If the delays are close an average delay is computed, and this approach to masking can be applied. The administrator interaction system sends a message to each stage with the start time of the performance and the average delay between the stages. The human interaction system starts a countdown at the given start-time. At a stage, each remote presence representing a local actor at the stage is locally delayed by the average delay. The remote presences from other stages are not delayed by the receiving stages.

Delay Locally the remote presences until data for the most delayed remote presence arrives: As for the Live Stage masking approach, the administrator interaction system uses the performance monitor to measure the delay from the detection computer at each stage to the distribution server. It also measures the delay from the distribution server to the remote presence computer at the stage. The effective delay from detection side of a stage to the display side of a stage is the sum of these two delays.

The administrative interaction system sends a message to each stage with the same start time, and the delay from every stage to the stage receiving the message. Each stage calculates by how much remote presences from each stage should be delayed to play back close in time to the remote presences coming from the stage with the longest delay. The human interaction system starts a countdown, and tells the actors what to do and when to do it. The create remote presence system creates remote presences as fast as it can, but remote presences from each stage are individually delayed by the calculated amount for each stage.

Act-By-Wire, blend in prerecorded video or compute a remote presence: The administrator interaction system sends the same start-time to the human interaction system at each stage. It starts a countdown and tells the actors what to do and when to do an action. For every image (or video frame) arriving to be used to create a remote presence, we check if the delta between the send timestamp of the image and the receive time is large enough to warrant masking. If more than a certain percentage of images are late, we start masking. If the percent goes down, we stop the masking. The threshold values used are based on subjectively trying the system on humans with different delay values, and determining when humans notice the delays in several settings, see later for more. We typically use a delay of about 280ms as the threshold for starting to do masking.

To mask short-term delays, the system check for delays over the last few seconds. The exact number of seconds used is tunable, depending upon how sensitive humans in a particular setting are to delayed remote presences.

The video used to mask the effects of delays is pre-recorded. The human interaction system does a countdown, and tells an actor what to do and when to do it, and a video is recorded. When later the same script is used during a performance, and the delays go above the threshold, the pre-recorded video blends in and takes over for the streaming video coming from a remote stage.

The masking system keeps ready the pre-recorded video in memory, and when masking is determined to be needed after checking the latency, it streams the pre-recorded video to the create the remote presence instead of the live streaming video.

Alternatively, instead of using a pre-recorded video, a model of an actor can be used. Instead of streaming a pre-recorded video to create a remote presence, the masking system streams the output from an implementation of the model. The model can receive input about detected body movements from the LSA (through the distribution server) of the remote stage. It can also use the script from the human interaction system to determine what an actor is meant to do. Presently, just a simple

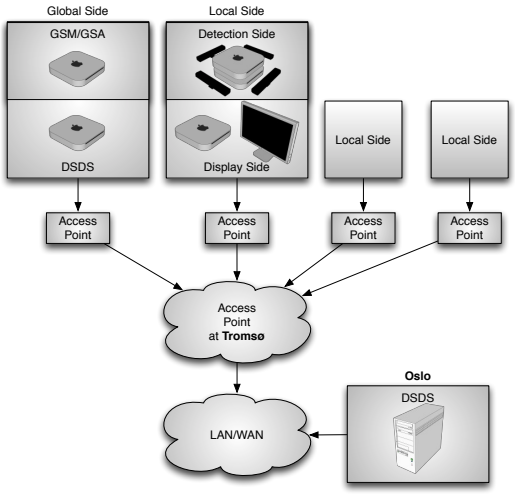


Fig. 6. The configuration of the experiments.

human skeleton model is used with arm movements taken from a script defining what an actor should do. It is future work to explore models and predicting actor behavior more fully.

VI. Evaluation

Several experiments were conducted to identify some of the effects of latency on the actors, and to document the measurable performance of the masking system. For the experiments the system was configured as given in figure 6. Computers used were Mac Minis at 2.7GHz and with 8GB 1333 MHz DDR3 memory. For all experiments all local side stages were on the same 1Gbit/s switched Ethernet LAN inside the Department of Computer Science at the University of Tromsø. The global side DSDS computer was either on the same LAN as the stages, or located on a Planetlab [37] computer at the University of Oslo, 1500km away.

System end-to-end one-way latency: The time it takes for a physical event happening on a stage to be picked up by the cameras and until a visualization of the actor is actually displayed on the same stage. We used a video camera with a high frame rate to record several videos of a user and the remote presence done on a display behind the user. We then counted frames to see how many frames it took from the user moved to the visualization caught up. On a LAN the end-to-end latency was between 90-125ms. With the DSDS at the computer in Oslo, the end-to-end latency was between 100-158ms. The variation in measured latency is because of several factors, including the distributed architecture of the prototype and the frame rate of the projector, video camera (240 fps) and the Kinects (30 fps), and other traffic on the LANs and WAN.

Global-to-Local round-trip latency: The latency going from the DSDS computer to a stage computer and back. We measured this by recording the time when we send a message from DSDS to a stage, and recording when a reply message comes back to DSDS. When all stages and the global side were on the same LAN, the round-trip latencies were between 1-2ms. When the DSDS system was on a computer in Oslo the round-trip latencies were around 32ms. This matches well with measurements reported by PingER [38] for Europe.

Actor-to-actor round-trip latency: The delay that actors will experience from when they do an action until they see the remote presence of another actor reacting. The typical latency between actors is two times the system end-to-end latency. Using the measured results from the system end-to-end one-way latency, the actor-to-actor round trip latency is from 180 to 316ms depending on where the DSDS computer is located.

Human response latency: The time it takes for a human actor to react to another actor's action. We used a high frame rate camera to record two actors' actions, and counted frames from when one actor initiated an action until the other actor responded to the action. The actions used were rapid and slow moving arm movements. The human response latency is about 345ms. We did not find that the latency varied significantly with the speed of an action.

Human noticeable latency: This is the latency at which a human actor will notice that an action is delayed. We simultaneously observed an actor and the corresponding remote presence. When the actor moves an arm, the remote presence moves an arm. In software we artificially added a delay to the remote presence until we noticed that the remote presence lagged behind the actor. When the added latency is more than 100ms, we did notice a difference of the movement between the actor and the remote presence.

Human tolerable latency: This is the latency an actor can tolerate before the illusion of being on the same stage with other actors breaks. We observed an actor shaking hands with another actor on the same stage. We then moved one of the actors to a remote stage, and repeated the shaking of hands. We now observed an actor shaking hands with a remote presence of the other actor. The delay between two actors were artificially increased until we subjectively decided that the handshake was not happening as fast as it did when the actors were physically on the same stage. We tried both rapid hand movement and slow hand movement. We subjectively decided that for a rapid hand movement, it is not tolerable when 150-200ms latency was added. The total actor-to-actor round-trip latency is in this case about 350-400ms. For slow hand movement, it is not tolerable when 600ms latency was added. The actor-to-actor roundtrip latency is about 800ms.

For handshake type of interaction, longer delays bordered on creating a feeling that the remote actor was being obnoxious by delaying just a bit too long before responding to a hand shake. However, this was not experienced unless we artificially added delays. This indicates that the prototype is able to maintain the illusion of being on the same stage for handshake type of interactions. However, we observe that the typical actor-to-actor round-trip latency in Europe is around 300ms or more. Consequently, when actors do fast and rapid interaction, the system can expect to have to mask the effects of the delays.

When to start masking: We simultaneously observed an actor moving an arm, and the corresponding remote presence.

In software we artificially added a random delay to every image used to create the remote presence. We tried different combinations of delays and for how many of the images were delayed. We found that when more than 50% of the received images during a period of three seconds were delayed 280ms or more there is a subjectively clearly visible lag in the remote presence vs. the actor. We therefore determine that when 50% of the images arrive 280ms late during the last three seconds, this is the threshold for when to start masking. This is a threshold that can be changed to customize for different usage scenarios.

When to stop masking: When masking is active, we need to establish a threshold for when to stop masking. We artificially create a situation where more than 50% of the images used to create a remote presence arrive too late. Consequently masking is done by the system. For the experiment we used the Act-By-Wire pre-recorded masking approach. We gradually decreased the percentage by 5% from 50% to 30%. We observe the switching back and forth between the live streaming of the remote presence and the pre-recorded stream. When 35-40% of the images arrive late the switch from the pre-recorded to the live streaming results in a transition without the observer noticing obvious effects of the delay. A higher percentage leads to a sooner switch, but the transition can be too fast and resulting in a blending in of the live streaming video with noticeable delays. A lower percentage results in keeping the pre-recorded video playing too long, and this can become noticeable by itself. The goal is to find a balance between when to start masking and when to stop. This can be different for different user activities and needs.

Above, we checked for late images during the last three seconds. A shorter period will lead to less delay in starting masking when needed, and a longer period is slower in starting masking. For shorter periods, a higher threshold for stopping the masking will reduce the likelihood of switching back and forth. For longer periods, a lower threshold for stopping the masking will increase the likelihood of switching back to the live streaming.

Cost of Masking: The CPU utilization at a remote presence computer without and with the masking technique active was measured. Two cameras were used sending images for two remote presences to a single remote presence computer. The CPU utilization without masking was about 22%. When masking was done for both remote presences using two pre-recorded videos the CPU utilization was basically the same, 22%. When masking was done using two human skeletons, the CPU utilization at the remote presence computer went down to 9%.

We explain this by observing that a significant part of the CPU load was consumed to display videos, making the masking itself insignificant. The very simple human skeleton approach is clearly less CPU demanding. We explain this by the simplicity of the model and that they use the display much less than the videos do.

The overhead of checking if masking is needed and to actually get the masking takes effect is about 40ms in average.

Table II shows the maximum system-end-to-end one-way latency at which each masking approach is in principle at least partially successful at masking the effects of delays.

VII. Discussion

Some of the masking techniques we applied need a synchronization of the clocks at every computer in, and consequently at, every stage of the system. The Network Time Protocol (NTP) provides time accuracy in the range of 1-30ms. The exact accuracy is highly dependent on the location of the computers vs. the NTP servers. If computers are on the same local area network, this will bring them close, around 1ms, to each other. If they are separated by the Internet, the clocks can be synchronized within tens of milliseconds to each other. However, network congestion and routing can cause the clock value used by each computer to be off hundreds of milliseconds. Therefore we do frequent NTP based clock settings and check explicitly for the clock difference between the computers to see if the clocks are more than 10ms off. If they are, we repeat using NTP to try to get all clocks within 10ms of each other. To further ensure that clocks are close enough, before the performance start time is sent to each stage, we again check the clock difference between the computer distributing data to all stages and the remote presence computers at every stage. The clock difference relevant for a stage is included in the message sent to each stage. A stage can then correct its performance start time accordingly if needed.

The experiments measured the objective metrics. No user studies were performed. The determination of thresholds was done naively based on the opinion of a few persons observing actors and remote presences.

The experiments used simple movements by an actor, primarily hand and arm movements. The results can be expected to be different for other actions done by actors, like body rotation, jumping, and dancing.

Different approaches to masking the effects of delays should be expected and to be needed based on what actors are doing. When actors do slow movements and the delays are low, the Act-By-Actor approach can be sufficient. However, it cannot mask the effects of larger delays. The Act-By-Director approach tells actors what to do and when to do an action. All actors are as such seen by an audience at a stage to be synchronized. This approach can mask the effects of large delays. The live stage approach will make just a single stage look synchronized. The other will typically be out of synchronization with the live stage and each other. The approach delaying the local remote presences by the amount of the delay to remote stages will make all stages synchronized if the artificial added delay is smaller than 65ms for audio and 300-400ms for video.

The approach of letting each stage do local delays of every remote presence waiting for the most delayed will make each stage to be in synchrony, but the stages will not be inter-stage synchronized. The Act-By-Wire approach can synchronize actors and remote presence of actors at all stages. However, it makes use of pre-recorded and creates on-the-fly remote

TABLE II
APPROACHES TO MASKING THE EFFECTS OF DELAYS. THE DELAY VALUES ARE THE MAXIMUM SYSTEM-END-TO-END ONE-WAY LATENCIES FOR WHEN AN APPROACH WILL BE AT LEAST PARTIALLY SUCCESSFUL AT MASKING THE EFFECTS OF DELAYS.

Approaches to masking the effects of delays	**Satisfactory synchrony between all remote presences at every stage**	**Satisfactory synchrony between all actors at every stage**	**Satisfactory synchrony between all actors and all remote presences at every stage**
Act-By-Actor	< 190-325ms	< 190-325ms	< 190-325ms
Act-By-Director	< 390-525ms	Any	< 390-525ms
Live Stage	Any (only at live stage)	< 390-525ms	Any (only at live stage)
Delay Local Remote Presence	Any	Any	< 390-525ms
Delay Locally All Remote Presences Waiting for the Slowest	Any	Any	< 390-525ms
Act-By-Wire (blend in pre-recorded remote presence)	Any	Any	Any
Act-By-Wire (blend in on the fly created remote presence)	Any	Any	Any

presences. These can be quite different from, say, a video of the actual actors.

All the masking approaches were tried in the prototype system. However, they are primarily documented as principles. To evaluate where they fit best in an actual interaction, they should be used, and the results should be studied.

The most advanced masking approach, Act-By-Wire using a model of the human to create the remote presence, can be applied with much more complex models than a human skeleton. This is future research. However, when a computable model of an actor is used, its execution should ideally produce results fast enough to not create further delays. If the model demands too long running time to create the needed output, a simpler model may have to be used. Alternatively, predictive techniques may be needed to have output ready when it is needed. The predictions can be based on pre-written scripts defining what a human is meant to be doing at any given time, or it can be based on analyzing the humans' actions in the near past. Predicting the behavior of an actor in the MultiStage system is future research.

VIII. CONCLUSION

In computer supported human-to-human interaction across distance, delays cannot be avoided. Consequently, while reducing the delays are well worth doing, sometimes they still become too large to ignore for humans. When this is the case, some of the effects of delays can be masked to create an illusion for the humans interacting, and for observers, that they are in the same room or on the same stage. However, the illusion created by masking has several limitations depending on which masking approach is used. There are two principally different types of masking. One type coordinates the interaction at suitable times to create a better illusion. The other frequently monitors the delays, and substitutes delayed data with data already available at each stage. Depending on the type of interaction, a suitable masking approach should be selected. The most complex approach, Act-By-Wire, will in all situations in principle create an illusion where interacting humans are fooled to believe that there are no significant delays perturbing the interaction. However, this approach can also create unexpected representations of remote humans, and when this happens it becomes clear that what is shown is only an approximation of the remote reality. The masking approaches we developed and did performance measurements on, demanded insignificantly more resources than not using them, and can even in the most complicated case when using Act-By-Wire, be switched in and out with insignificant delays.

Based on informal use of the system, we found that even 800ms of delay while interacting using slow movements in some cases were tolerable. However, the general case seems to be that delays above 200ms is noticeable when having remote presences based on vision and visualizations. We found that an actor-to-actor round-trip delay of above 200ms is frequently the case, and masking is consequently frequently needed.

ACKNOWLEDGMENT

Many thanks to the technical staff at the department. This work was funded in part by the Norwegian Research Council, projects 187828, 159936/V30, 155550/420, and Tromsø Research Foundation (Tromsø Forskningsstiftelse).

REFERENCES

[1] A. Pavlovych and W. Stuerzlinger, "Target following performance in the presence of latency, jitter, and signal dropouts," in *Proceedings of Graphics Interface 2011*. Canadian Human-Computer Communications Society, 2011, pp. 33–40.

[2] L. Pantel and L. C. Wolf, "On the impact of delay on real-time multiplayer games," in *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*. ACM, 2002. doi: http://dx.doi.org/10.1145/507670.507674 pp. 23–29.

[3] [Online]. Available: http://www.measurepolis.fi/alma/ALMA%20Human%20Reaction%20Times%20as%20a%20Response%20to%20Delays%20in%20Control%20Systems.pdf

[4] X. Jiang, F. Safaei, and P. Boustead, "Latency and scalability: a survey of issues and techniques for supporting networked games," in *Networks, 2005. Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication*, vol. 1. IEEE, 2005. doi: http://dx.doi.org/10.1109/ICON.2005.1635458 pp. 6–pp.

[5] F. Su, G. Tartari, J. Bjørndalen, P. Ha, and O. Anshus, "Multistage: Acting across distance," in *Information Technologies for Performing Arts, Media Access, and Entertainment*, ser. Lecture Notes in Computer Science, P. Nesi and R. Santucci, Eds., vol. 7990, no. 978-3-642-40049-0. Springer Berlin Heidelberg, 2013. doi: http://dx.doi.org/10.1007/978-3-642-40050-6_20 pp. 227–239.

[6] A. A. Sawchuk, E. Chew, R. Zimmermann, C. Papadopoulos, and C. Kyriakakis, "From remote media immersion to distributed immersive performance," in *Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence*. ACM, 2003. doi: http://dx.doi.org/10.1145/982484.982506 pp. 110–120.

[7] R. Zimmermann, E. Chew, S. A. Ay, and M. Pawar, "Distributed musical performances: Architecture and stream management," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, vol. 4, no. 2, p. 14, 2008. doi: http://dx.doi.org/10.1145/1352012.1352018

[8] E. Chew, C. Kyriakakis, C. Papadopoulos, A. Sawchuk, and R. Zimmermann, "Distributed immersive performance: Enabling technologies for and analyses of remote performance and collaboration."

[9] A. Basu, A. Raij, and K. Johnsen, "Ubiquitous collaborative activity virtual environments," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 2012. doi: http://dx.doi.org/10.1145/2145204.2145302 pp. 647–650.

[10] A. Tang, M. Pahud, K. Inkpen, H. Benko, J. C. Tang, and B. Buxton, "Three's company: understanding communication channels in three-way distributed collaboration," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, 2010. doi: http://dx.doi.org/10.1145/1718918.1718969 pp. 271–280.

[11] H. H. Baker, N. Bhatti, D. Tanguay, I. Sobel, D. Gelb, M. E. Goss, W. B. Culbertson, and T. Malzbender, "Understanding performance in coliseum, an immersive videoconferencing system," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, vol. 1, no. 2, pp. 190–210, 2005. doi: http://dx.doi.org/10.1145/1062253.1062258

[12] [Online]. Available: http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php

[13] Y. W. Bernier, "Latency compensating methods in client/server in-game protocol design and optimization," in *Game Developers Conference*, vol. 98033, no. 425, 2001.

[14] Z. Li, X. Tang, W. Cai, and S. J. Turner, "Fair and efficient dead reckoning-based update dissemination for distributed virtual environments," in *2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation (PADS)*. IEEE, 2012. doi: http://dx.doi.org/10.1109/PADS.2012.18 pp. 13–22.

[15] T. K. Capin and I. S. Pandzic, "A dead-reckoning algorithm for virtual human figures," in *Virtual Reality Annual International Symposium, 1997, IEEE 1997*. IEEE, 1997. doi: http://dx.doi.org/10.1109/VRAIS.1997.583066 pp. 161–169.

[16] V. Y. Kharitonov, "Motion-aware adaptive dead reckoning algorithm for collaborative virtual environments," in *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*. ACM, 2012. doi: http://dx.doi.org/10.1145/2407516.2407577 pp. 255–261.

[17] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "Idmaps: A global internet host distance estimation service," *Networking, IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 5, pp. 525–540, 2001. doi: http://dx.doi.org/10.1109/90.958323

[18] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002. doi: http://dx.doi.org/10.1145/637201.637203 pp. 5–18.

[19] H. V. Madhyastha, T. Anderson, A. Krishnamurthy, N. Spring, and A. Venkataramani, "A structural approach to latency prediction," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 2006. doi: http://dx.doi.org/10.1145/1177080.1177092 pp. 99–104.

[20] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye, "Measurement and estimation of network qos among peer xbox 360 game players," in *Passive and Active Network Measurement, doi: http://dx.doi.org/10.1007/978-3-540-79232-1_5*. Springer, 2008, pp. 41–50.

[21] P. Huang, Y. Ishibashi, N. Fukushima, and S. Sugawara, "Interactivity improvement of group synchronization control in collaborative haptic play with building blocks," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2010, p. 2.

[22] Y. Zhang, L. Chen, and G. Chen, "Globally synchronized dead-reckoning with local lag for continuous distributed multiplayer games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2006. doi: http://dx.doi.org/10.1145/1230040.1230071 p. 7.

[23] A. Malik Khan, S. Chabridon, and A. Beugnard, "A dynamic approach to consistency management for mobile multiplayer games," in *Proceedings of the 8th international conference on New technologies in distributed systems*. ACM, 2008. doi: http://dx.doi.org/10.1145/1416729.1416783 p. 42.

[24] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in dead-reckoning based distributed multiplayer games," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, 2004. doi: http://dx.doi.org/10.1145/1016540.1016559 pp. 161–165.

[25] T. Yasui, Y. Ishibashi, and T. Ikedo, "Influences of network latency and packet loss on consistency in networked racing games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2005. doi: http://dx.doi.org/10.1145/1103599.1103622 pp. 1–8.

[26] M. Bredel and M. Fidler, "A measurement study regarding quality of service and its impact on multiplayer online games," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2010. doi: http://dx.doi.org/10.1109/NETGAMES.2010.5679537 p. 1.

[27] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, pp. 40–45, 2006. doi: http://dx.doi.org/10.1145/1167838.1167860

[28] D. Roberts, T. Duckworth, C. Moore, R. Wolff, and J. O'Hare, "Comparing the end to end latency of an immersive collaborative environment and a video conference," in *Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*. IEEE Computer Society, 2009. doi: http://dx.doi.org/10.1109/DS-RT.2009.43 pp. 89–94.

[29] G. Papadakis, K. Mania, and E. Koutroulis, "A system to measure, control and minimize end-to-end head tracking latency in immersive simulations," in *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*. ACM, 2011. doi: http://dx.doi.org/10.1145/2087756.2087869 pp. 581–584.

[30] [Online]. Available: http://www.serviceassurancedaily.com/2008/06/latency-and-jitter/

[31] A. Pavlovych and C. Gutwin, "Assessing target acquisition and tracking performance for complex moving targets in the presence of latency and jitter," in *Proceedings of the 2012 Graphics Interace Conference*. Canadian Information Processing Society, 2012, pp. 109–116.

[32] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of factors affecting players' performance and perception in multiplayer games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2005. doi: http://dx.doi.org/10.1145/1103599.1103624 pp. 1–7.

[33] G. Armitage and L. Stewart, "Limitations of using real-world, public servers to estimate jitter tolerance of first person shooter games," in *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*. ACM, 2004. doi: http://dx.doi.org/10.1145/1067343.1067377 pp. 257–262.

[34] S. Aggarwal, H. Banavar, S. Mukherjee, and S. Rangarajan, "Fairness in dead-reckoning based distributed multi-player games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2005. doi: http://dx.doi.org/10.1145/1103599.1103608 pp. 1–10.

[35] R. David, "Motion-detection-opencv." [Online]. Available: https://github.com/RobinDavid/Motion-detection-OpenCV

[36] [Online]. Available: http://www.ntp.org/

[37] [Online]. Available: https://www.planet-lab.eu/

[38] [Online]. Available: http://www-wanmon.slac.stanford.edu/cgi-wrap/pingtable.pl

Bibliography

[1] C. Griwodz, N. Pål Halvorsen, G. H. O. A. Distributed Systems, Simula Research Laboratory, U. o. T. N. W. L. D. S. U. o. T. F. E. Tore Larsen, Computer Science, N. Roman Vitenberg, and A.-L. S. M. C. U. C. T. V. S. W. E. L. A. F. H. T. R. Distributed Systems, University of Oslo. Verdione virtually enhanced real-life synchronized interaction - on the edge. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.7670&rep=rep1&type=pdf

[2] F. Su, G. Tartari, J. Bjørndalen, P. Ha, and O. Anshus, "Multistage: Acting across distance," in *Information Technologies for Performing Arts, Media Access, and Entertainment*, ser. Lecture Notes in Computer Science, P. Nesi and R. Santucci, Eds. Springer Berlin Heidelberg, 2013, vol. 7990, no. 978-3-642-40049-0, pp. 227–239.

[3] G. Tartari, D. Stodlet, J. Bjorndalen, P. H. Ha, and O. J. Anshus, "Global interaction space for user interaction with a room of computers," in *Human System Interaction (HSI), 2013 The 6th International Conference on.* IEEE, 2013, pp. 84–89.

[4] R. Rodrigues and P. Druschel, "Peer-to-peer systems," *Commun. ACM*, vol. 53, no. 10, pp. 72–82, Oct. 2010. [Online]. Available: http://doi.acm.org/10.1145/1831407.1831427

[5] Y. Sato, K. Hashimoto, and Y. Shibata, "A new remote camera work system for teleconference using a combination of omni-directional and network controlled cameras," in *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on.* IEEE, 2008, pp. 502–508.

[6] A. Sawchuk, E. Chew, R. Zimmermann, C. Papadopoulos, and C. Kyriakakis, "From remote media immersion to distributed immersive performance," in *Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence.* ACM, 2003, pp. 110–120.

[7] R. Zimmermann, E. Chew, S. Ay, and M. Pawar, "Distributed musical performances: Architecture and stream management," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, vol. 4, no. 2, p. 14, 2008.

[8] A. Nagendran, R. Pillat, C. Hughes, and G. Welch, "Continuum of virtual-human space: towards improved interaction strategies for physical-virtual avatars," in *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*. ACM, 2012, pp. 135–142.

[9] A. Tang, M. Pahud, K. Inkpen, H. Benko, J. Tang, and B. Buxton, "Three's company: understanding communication channels in three-way distributed collaboration," in *Proceedings of the 2010 ACM conference on Computer supported cooperative work*. ACM, 2010, pp. 271–280.

[10] J. Aronson. [Online]. Available: http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php

[11] P. Huang, Y. Ishibashi, N. Fukushima, and S. Sugawara, "Interactivity improvement of group synchronization control in collaborative haptic play with building blocks," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2010, p. 2.

[12] M. Särelä, T. Rinta-aho, and S. Tarkoma, "Rtfm: Publish/subscribe internetworking architecture," *ICT Mobile Summit*, vol. 29, pp. 73–82, 2008.

[13] [Online]. Available: http://code.google.com/p/psutil/

[14] F. Su, J. M. Bjørndalen, P. H. Ha, and O. J. Anshus, "Masking the effects of delays in human-to-human remote interaction," in *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference on*. IEEE, 2014, pp. 719–728.

[15] ——, "pvd—personal video distribution," in *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*. IEEE, 2013, pp. 687–692.

[16] [Online]. Available: http://www.ntp.org/

[17] [Online]. Available: http://golang.org/

[18] D. Stodle, O. Troyanskaya, K. Li, and O. Anshus, "Tech-note: Device-free interaction spaces," in *3D User Interfaces, 2009. 3DUI 2009. IEEE Symposium on*. IEEE, 2009, pp. 39–42.

[19] [Online]. Available: http://www.horde3d.org/

[20] D. Sakamoto, T. Kanda, T. Ono, H. Ishiguro, and N. Hagita, "Android as a telecommunication medium with a human-like presence," in *Human-Robot Interaction (HRI), 2007 2nd ACM/IEEE International Conference on*. ACM, 2007, pp. 193–200.

[21] M. Dou, Y. Shi, J. Frahm, H. Fuchs, B. Mauchly, and M. Marathe, "Room-sized informal telepresence system," in *Virtual Reality Workshops (VR), 2012 IEEE*. IEEE, 2012, pp. 15–18.

[22] B. Petit, J. Lesage, C. Menier, J. Allard, J. Franco, B. Raffin, E. Boyer, and F. Faure, "Multicamera real-time 3d modeling for telepresence and remote collaboration," *International journal of digital multimedia broadcasting*, vol. 2010, 2009.

[23] S. Essid, X. Lin, M. Gowing, G. Kordelas, A. Aksay, P. Kelly, T. Fillon, Q. Zhang, A. Dielmann, V. Kitanovski *et al.*, "A multi-modal dance corpus for research into interaction between humans in virtual environments," *Journal on Multimodal User Interfaces*, pp. 1–14, 2012.

[24] F. Tecchia, L. Alem, and W. Huang, "3d helping hands: a gesture based mr system for remote collaboration," in *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*. ACM, 2012, pp. 323–328.

[25] K. Misawa, Y. Ishiguro, and J. Rekimoto, "Livemask: A telepresence surrogate system with a face-shaped screen for supporting nonverbal communication," in *Proceedings of the International Working Conference on Advanced Visual Interfaces*. ACM, 2012, pp. 394–397.

[26] K. Kim, J. Bolton, A. Girouard, J. Cooperstock, and R. Vertegaal, "Telehuman: effects of 3d perspective on gaze and pose estimation with a life-size cylindrical telepresence pod," in *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems*. ACM, 2012, pp. 2531–2540.

[27] [Online]. Available: http://openkinect.org/wiki/Main_Page

[28] [Online]. Available: http://www.pygame.org/news.html

[29] A. Pavlovych and W. Stuerzlinger, "Target following performance in the presence of latency, jitter, and signal dropouts," in *Proceedings of Graphics Interface 2011*. Canadian Human-Computer Communications Society, 2011, pp. 33–40.

[30] L. Pantel and L. C. Wolf, "On the impact of delay on real-time multiplayer games," in *Proceedings of the 12th international workshop on Network and operating systems support for digital audio and video*. ACM, 2002, pp. 23–29.

[31] [Online]. Available: http://www.measurepolis.fi/alma/ALMA%20Human%20Reaction%20Times%20as%20a%20Response%20to%20Delays%20in%20Control%20Systems.pdf

[32] X. Jiang, F. Safaei, and P. Boustead, "Latency and scalability: a survey of issues and techniques for supporting networked games," in *Networks, 2005. Jointly held with the 2005 IEEE 7th Malaysia International Conference on Communication*, vol. 1. IEEE, 2005, pp. 6–pp.

[33] T. T. P. D. V. S. ROGER ZIMMERMANN, C. P. A. F. ILIA TOSHEFF), CHRISTOS KYRIAKAKIS, and A. VOLK, "The internet for ensemble performance? distributed immersive performance."

[34] E. Chew, C. Kyriakakis, C. Papadopoulos, A. Sawchuk, and R. Zimmermann, "Distributed immersive performance: Enabling technologies for and analyses of remote performance and collaboration." NIME 06, 2006.

[35] A. Basu, A. Raij, and K. Johnsen, "Ubiquitous collaborative activity virtual environments," in *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*. ACM, 2012, pp. 647–650.

[36] H. H. Baker, N. Bhatti, D. Tanguay, I. Sobel, D. Gelb, M. E. Goss, W. B. Culbertson, and T. Malzbender, "Understanding performance in coliseum, an immersive videoconferencing system," *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, vol. 1, no. 2, pp. 190–210, 2005.

[37] Y. W. Bernier, "Latency compensating methods in client/server in-game protocol design and optimization," in *Game Developers Conference*, vol. 98033, no. 425, 2001.

[38] Z. Li, X. Tang, W. Cai, and S. J. Turner, "Fair and efficient dead reckoning-based update dissemination for distributed virtual environments," in *2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation (PADS)*. IEEE, 2012, pp. 13–22.

[39] T. K. Capin and I. S. Pandzic, "A dead-reckoning algorithm for virtual human figures," in *Virtual Reality Annual International Symposium, 1997, IEEE 1997*. IEEE, 1997, pp. 161–169.

[40] V. Y. Kharitonov, "Motion-aware adaptive dead reckoning algorithm for collaborative virtual environments," in *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*. ACM, 2012, pp. 255–261.

[41] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "Idmaps: A global internet host distance estimation service," *Networking, IEEE/ACM Transactions on Networking (TON)*, vol. 9, no. 5, pp. 525–540, 2001.

[42] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary internet end hosts," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 5–18.

[43] H. V. Madhyastha, T. Anderson, A. Krishnamurthy, N. Spring, and A. Venkataramani, "A structural approach to latency prediction," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 2006, pp. 99–104.

[44] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye, "Measurement and estimation of network qos among peer xbox 360 game players," in *Passive and Active Network Measurement*. Springer, 2008, pp. 41–50.

[45] Y. Zhang, L. Chen, and G. Chen, "Globally synchronized dead-reckoning with local lag for continuous distributed multiplayer games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2006, p. 7.

[46] A. Malik Khan, S. Chabridon, and A. Beugnard, "A dynamic approach to consistency management for mobile multiplayer games," in *Proceedings of the 8th international conference on New technologies in distributed systems*. ACM, 2008, p. 42.

[47] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in dead-reckoning based distributed multi-player games," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*. ACM, 2004, pp. 161–165.

[48] T. Yasui, Y. Ishibashi, and T. Ikedo, "Influences of network latency and packet loss on consistency in networked racing games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2005, pp. 1–8.

[49] M. Bredel and M. Fidler, "A measurement study regarding quality of service and its impact on multiplayer online games," in *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*. IEEE Press, 2010, p. 1.

[50] M. Claypool and K. Claypool, "Latency and player actions in online games," *Communications of the ACM*, vol. 49, no. 11, pp. 40–45, 2006.

[51] D. Roberts, T. Duckworth, C. Moore, R. Wolff, and J. O'Hare, "Comparing the end to end latency of an immersive collaborative environment and a video conference," in *Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*. IEEE Computer Society, 2009, pp. 89–94.

[52] G. Papadakis, K. Mania, and E. Koutroulis, "A system to measure, control and minimize end-to-end head tracking latency in immersive simulations," in *Proceedings of the 10th International Conference on Virtual Reality Continuum and Its Applications in Industry*. ACM, 2011, pp. 581–584.

[53] [Online]. Available: http://www.serviceassurancedaily.com/2008/06/latency-and-jitter/

[54] A. Pavlovych and C. Gutwin, "Assessing target acquisition and tracking performance for complex moving targets in the presence of latency and jitter," in *Proceedings of the 2012 Graphics Interace Conference*. Canadian Information Processing Society, 2012, pp. 109–116.

[55] M. Dick, O. Wellnitz, and L. Wolf, "Analysis of factors affecting players' performance and perception in multiplayer games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2005, pp. 1–7.

[56] G. Armitage and L. Stewart, "Limitations of using real-world, public servers to estimate jitter tolerance of first person shooter games," in *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*. ACM, 2004, pp. 257–262.

[57] S. Aggarwal, H. Banavar, S. Mukherjee, and S. Rangarajan, "Fairness in dead-reckoning based distributed multi-player games," in *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*. ACM, 2005, pp. 1–10.

[58] C. Y. Chen, J. H. Fu, T.-L. Sung, P.-F. Wang, E. Jou, and M.-W. Feng, "Complex event processing for the internet of things and its applications," in *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*. IEEE, 2014, pp. 1144–1149.

[59] J. B. Leners, H. Wu, W.-L. Hung, M. K. Aguilera, and M. Walfish, "Detecting failures in distributed systems with the falcon spy network," in *Proceedings of the*

Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011, pp. 279–294.

[60] [Online]. Available: https://wiki.python.org/moin/TkInter

[61] [Online]. Available: https://www.planet-lab.eu/

[62] [Online]. Available: http://www-iepm.slac.stanford.edu/pinger/

[63] [Online]. Available: https://youtu.be/qiUcZdkyCy4?list=PLxuUYr6dflLOl7ebGD74qr1uMGdOX31j0

[64] [Online]. Available: https://youtu.be/uNgojjFukRw

[65] [Online]. Available: https://youtu.be/hPXJ0JT3hv0

[66] [Online]. Available: https://youtu.be/n4Ty_mi7JGo

[67] [Online]. Available: https://youtu.be/ZXS6W3A9Hxo

[68] [Online]. Available: https://youtu.be/p6G5tISK9LY

[69] [Online]. Available: https://youtu.be/FTm1bOd2jpU

[70] [Online]. Available: https://youtu.be/Ea7F-5YXNHo

[71] M. Meehan, S. Razzaque, M. C. Whitton, and F. P. Brooks, "Effect of latency on presence in stressful virtual environments," in *IEEE Virtual Reality, 2003. Proceedings.* IEEE, 2003, pp. 141–148.

[72] S. M. Habib, S. Ries, and M. Muhlhauser, "Cloud computing landscape and research challenges regarding trust and reputation," in *Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2010 7th International Conference on.* IEEE, 2010, pp. 410–415.

[73] I. Ion, N. Sachdeva, P. Kumaraguru, and S. Čapkun, "Home is safer than the cloud!: privacy concerns for consumer cloud storage," in *Proceedings of the Seventh Symposium on Usable Privacy and Security.* ACM, 2011, p. 13.

[74] E. Chin, A. P. Felt, V. Sekar, and D. Wagner, "Measuring user confidence in smartphone security and privacy," in *Proceedings of the Eighth Symposium on Usable Privacy and Security.* ACM, 2012, p. 1.

[75] [Online]. Available: http://www.akamai.com/

[76] [Online]. Available: http://www.bootstrapcdn.com/

[77] [Online]. Available: http://www.bittorrent.com/

[78] [Online]. Available: http://www.pps.tv/

[79] [Online]. Available: http://www.pptv.com/

[80] [Online]. Available: http://www.livecast.com/

[81] [Online]. Available: http://qik.com/

[82] [Online]. Available: http://www.upnp.org/

[83] [Online]. Available: http://www.dlna.org/

[84] [Online]. Available: http://www.wi-fi.org/discover-wi-fi/wi-fi-certified-miracast

[85] [Online]. Available: http://www.apple.com/airplay/

[86] C. Ullrich, R. Shen, R. Tong, and X. Tan, "A mobile live video learning system for large-scale learning—system design and evaluation," *Learning Technologies, IEEE Transactions on*, vol. 3, no. 1, pp. 6–17, 2010.

[87] K. Wolf, S. Linckels, and C. Meinel, "Teleteaching anywhere solution kit(tele-task) goes mobile," in *User Services Conference: Proceedings of the 35 th annual ACM SIGUCCS conference on User services*, vol. 7, no. 10, 2007, pp. 366–371.

[88] H.-Y. Chang, Y.-Y. Shih, and Y.-W. Lin, "Cloudpp: A novel cloud-based p2p live video streaming platform with svc technology," in *Computing Technology and Information Management (ICCM), 2012 8th International Conference on*, vol. 1. IEEE, 2012, pp. 64–68.

[89] F. V. Hecht, T. Bocek, R. G. Clegg, R. Landa, D. Hausheer, and B. Stiller, "Liveshift: Mesh-pull live and time-shifted p2p video streaming," in *Local Computer Networks (LCN), 2011 IEEE 36th Conference on.* IEEE, 2011, pp. 315–323.

[90] E. Kim and C. Lee, "An on-demand tv service architecture for networked home appliances," *Communications Magazine, IEEE*, vol. 46, no. 12, pp. 56–63, 2008.

[91] A. Kaheel, M. El-Saban, M. Refaat, and M. Ezz, "Mobicast: a system for collaborative event casting using mobile phones," in *Proceedings of the 8th International Conference on Mobile and Ubiquitous Multimedia.* ACM, 2009, p. 7.

[92] L. Gou, J.-H. Kim, H.-H. Chen, J. Collins, M. Goodman, X. L. Zhang, and C. L. Giles, "Mobisna: a mobile video social network application," in *Proceedings of the Eighth ACM International Workshop on Data Engineering for Wireless and Mobile Access*. ACM, 2009, pp. 53–56.

[93] Y. Huang, Z. Li, G. Liu, and Y. Dai, "Cloud download: using cloud utilities to achieve high-quality content distribution for unpopular videos," in *Proceedings of the 19th ACM international conference on Multimedia*. ACM, 2011, pp. 213–222.

[94] Y. Wang, P. Zhao, D. Zhang, M. Li, and H. Zhang, "Myvideos: a system for home video management," in *Proceedings of the tenth ACM international conference on Multimedia*. ACM, 2002, pp. 412–413.

[95] [Online]. Available: http://www.apple.com/airport-extreme/