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.

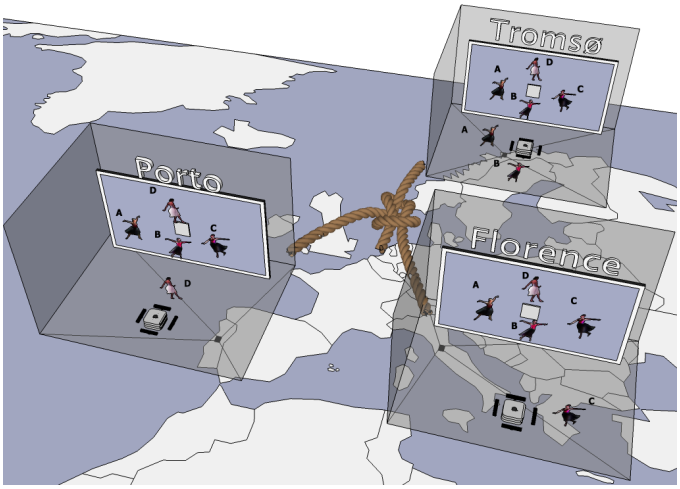

Fig. 1. Four dancers at different stages dance together. Each stage is equipped with sensors to detect actors and a display to visualize the remote presence of all the performers.

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

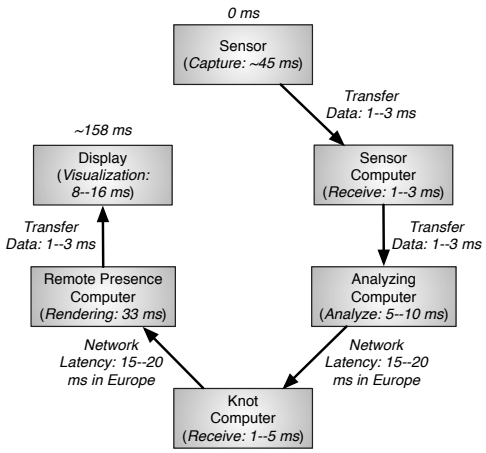

Fig. 2. Every Phase will add delay

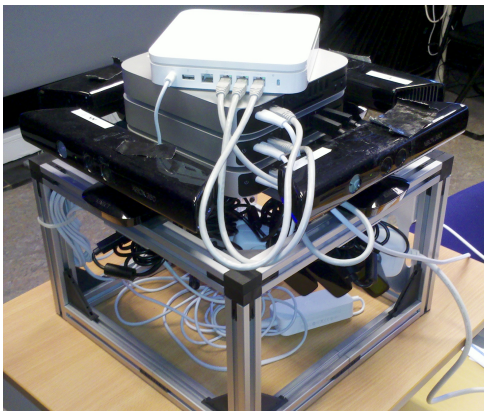

Fig. 3. The 360 degrees actor detection sensor rig comprising four 3D Kinect cameras, two Mac Mini computers, and wireless access point. One per stage is used.

the sensor input, network transmission, on-route processing, 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),

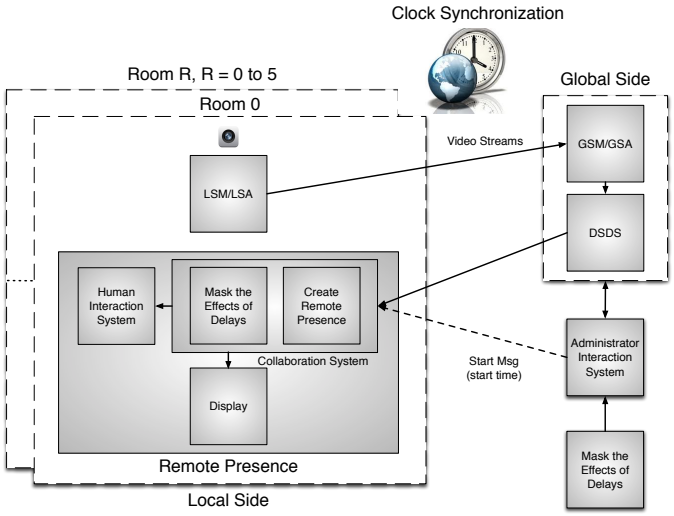

Fig. 4. The MultiStage system.

and forwards the data and data about detected gestures to the global side.

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 move-

ments, the data about this makes its way to the remote presence, 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.

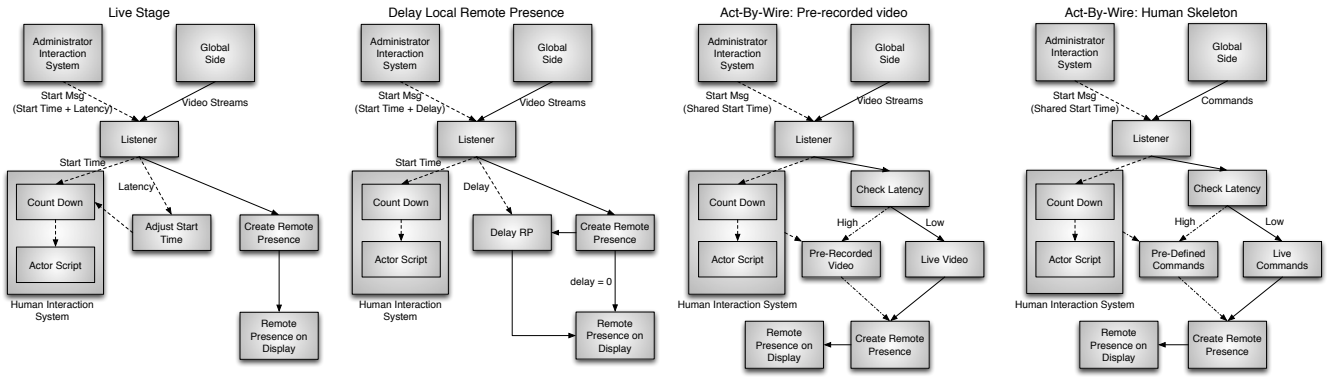

Fig. 5. Design and Implementation of the techniques to mask the effects of delays

The administrator interaction system now sends a message to each stage with the start time of the performance and 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

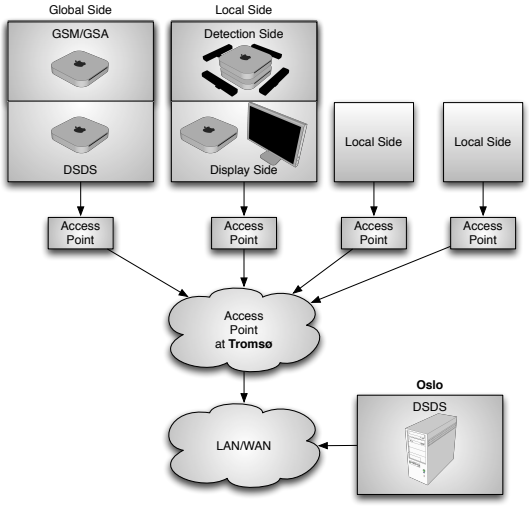

Fig. 6. The configuration of the experiments.

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 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

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

actors and remote presence of actors at all stages. However, it makes use of pre-recorded and creates on-the-fly remote 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 multi-player 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