Telemersive Toolkit

Exchange Multi Media Streams for Distributed Networked Performance Installations

Martin Fröhlich*
Roman Haefeli†
Patrick Müller‡
martin.froehlich@zhdk.ch
roman.haefeli@zhdk.ch
patrick.mueller@zhdk.ch
Zuerich University of the Arts
Zürich, ZH, Switzerland

Figure 1: telemersive-gateway, Network Matrix GUI Expanded, Screenshot for project 'Osmosis', 2023.

ABSTRACT

This paper describes the Telemersive Toolkit [hereafter referred to as TTkit], a system that enables artists and educators to set up complex low-latency multimedia streaming infrastructures between multiple computers connected via networks. TTkit addresses the challenge for non-technical personnel to create their own streaming environment that does not require reconfiguration of the local network infrastructure and allows computers to be connected across different network configurations and firewall setups. The TTkit provides an intuitive user interface called telemersive-gateway [hereafter referred to as Gateway] to set up the network and displays the current status of all streams – video, audio, motion tracking data, control data – exchanged within the network. Each Gateway on the network can monitor and configure the streaming setup of all other Gateways on the network, making it a very convenient tool for troubleshooting and helping artists, researchers as well as educators (e.g. to assist novice users) in setting up their individual machines. In order to evaluate TTkit, the Telematic Perfomance Format Research Group has used it for its three most recent project realisations in the form of telematic multimedia music and theater performances, one of which, called 'Osmosis', is described in this paper to give a clearer understanding of the practical application of this system. Our findings suggest that TTkit is ready to be used by a wider community, and we believe that it will help artists and educators to create their own networked performance environments without spending too much time setting up the technical infrastructure, and thus focus more on the artistic freedom that the new system offers.

*Main author
†Developer of subsystem
‡Prinicpal investigator

IMX '24, June 12–14, 2024, Stockholm, Sweden

ACM ISBN 979-8-4007-0503-8/24/06
https://doi.org/10.1145/3639701.3663637

CCS CONCEPTS

• **Applied computing** → **Media arts**; • **Networks** → **Network monitoring**; • **Information systems** → **Multimedia streaming**; • **Human-centered computing** → **Open source software**; • **Computer systems organization** → **Real-time systems**.

KEYWORDS

Telematics, Software, Tool, Network, Multimedia, Streaming, UDP, OSC, Motion capture, Remote control, CLI, UltraGrid

ACM Reference Format:
Martin Fröhlich, Roman Haefeli, and Patrick Müller. 2024. Telemersive Toolkit: Exchange Multi Media Streams for Distributed Networked Performance Installations. In *ACM International Conference on Interactive Media Experiences (IMX '24), June 12–14, 2024, Stockholm, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3639701.3663637

1 INTRODUCTION

With the opening salvos of Covid19 and the subsequent closure of public life in the early spring of 2020, society as a whole suddenly turned to digital collaboration in its ways of working together. The performing arts in particular, with their closed theatres, cinemas and performance stages, were faced with the existential challenge of continuing their art in a remote way. And while some infrastructure was already in place to meet this new demand, new tools had to be developed to meet the individual needs of the artistic community and their specific aesthetic development processes.

2 RELATED WORK

While the pandemic was a galvanizing moment to spur innovations in the way we can remotely collaborate, the quest [11] to realize telematic practices has its roots much earlier. The introduction of the internet in the early 90's of the last century marks the moment when it became feasible for everybody equipped with the right frustration tolerance to bring remote locations together in order to experiment collaboratively with theater [7, 21], dance [26] and music [25], including audio and video transmission and eventually further control data as OSC[1] or MIDI. These early adopters [8, 27, 36] used commercially available video conferencing software [20, 28, 38], but the available network bandwidth, latency issues and compression technology made it a very, let's say, pixelated experience. However, these systems were designed for a different interaction scheme and not ideal for live performances.

The first generation of tools for Network (Music) Performances [32] allowed greater integration with existing audio and video software. Especially on the audio side, probably because of the time-sensitive nature of music and the focus on high fidelity, a number of solutions [5, 17] became available. On the video side, high-quality video transmission software solutions originally developed for educational purposes or for telemedicine, namely Ultragrid [19] have also been employed for such performances, but they still require some technical acumen or even programming [22] to implement into the desired infrastructure.

The second generation, with some of the core technological challenges [4, 12](low latency, fidelity) now out of the way, is focusing more on usability and collaboration. Some first generation tools evolved to second generation tools, like Jacktrip [10] or LoLa [17]. More recent audio streaming tools, catalysed by the Covid-pandemic, like Jamulus [16] or Sonobus [24], emphasise ease of use. Audio streaming software including video features like Soundjack [3], Farplay [14], Elk [13], Source-Live [33] all make it easy for musicians to connect and play together. Their interfaces are well designed and intuitive. While they meet the requirements of low latency, fidelity and usability, they are still essentially video conferencing solutions for musicians. Other audio-streaming tools like Artsmesh [15, 35] or TPF-Tools [18] propose UI solutions for the use of multichannel audio streaming between locations, an issue that is unresolved in most of the plug-and-play tools mentioned above.

Other approaches seek to serve other performative arts. 'Distributed Theatre Performances' [6] helps theatre productions connect multiple stages, their local audiences and an online audience through live video (and audio). 'Co-creation stage' [31], which is entirely web-based, is designed to facilitate collaboration [34] and the involvement of diverse communities in the realisation of operas. Both tools have built-in cueing systems to mix and match different video sources and target screens. And both tools seem to have been designed with a virtual TV studio in mind, but with bi-directional features, unlike its inspiration.

The true children of Covid19 are Quacktrip [29], Digital Stage [2], VDO.ninja [37] and LiveLab [9]. Quacktrip is a Pure Data implementation based on Jacktrip and audio only, aimed mainly for musicians. Digital Stage has a wider application within the performing arts. It has a web interface for rehearsals, desktop solutions for more complex configurations and a custom hardware solution. VDO.ninja can connect up to 10 peers with audio and video (the bandwidth is the limit since all its transmissions are peer-to-peer). It stands out from the rest as it only needs a browser based on chromium[2] and no registration is required. It is widely used by the web streaming communities in combination with OBS[3]. LiveLab is also a pure browser based solution and shows a very similar feature set as VDO.ninja.

3 MOTIVATION

It is obvious that it is impossible to create one tool that fits all. The approach described in this paper is to present a transmission solution that can combine and seamlessly connect the rich content creation tools that already exist. It provides the necessary interfaces to manage, monitor and troubleshoot all data connections within the network, and the required low latency and fidelity expected of a modern distributed performance tool.

One reason TTkit and its accompanying services are designed as they are boils down to the usage of UltraGrid as the core transmission utility for video and audio. While Ultragird may not be the lowest latency solution for audio [30, 32], it is to our knowledge for video. It is open source, very robust, feature rich and can interface with almost everything in hardware and software.

However, it is designed for single peer-to-peer connection. And while there is a simple QT-based UI available, its is at its core a CLI application, which makes the usage for novices intimidating. But the major problem to overcome is on the network side: the prevailing network architectures with modern NAT-Routers[4] create security hurdles that need to be overcome, and one solution is the UDP proxy [18]. By expanding on that idea it was possible to use UltraGrid

[1] OSC (open sound control) is a widely used data protocol in media installations: https://en.wikipedia.org/wiki/Open_Sound_Control

[2] Chromium - https://de.wikipedia.org/wiki/Chromium_(Browser)

[3] OBS - Open Broadcaster Software - https://obsproject.com/

[4] NAT (network address translation) routers separate the LAN from the Internet. This increases security, because local computers are invisible from the Internet.

in other typologies like 'one to many'. And since OSC can also be transported via UDP, the topology 'many to many' (imitating OSCGroups [1]) worked with this approach, too.

Soon projects were realized that needed more than a few connections between machines. And since for every connection two instances of UltraGrid have to be started, one on the sender side and one or many on the receiving end, a lot of similar looking command line windows on diverse machines in different locations started cluttering the window space. Then additional data types had to be streamed at the same time. The complexity increased by the subsequent need for more proxies to run on the server on multiple ports. Using remote desktop solutions was an option, but has the disadvantage of additional performance load on the GFX card. In short: Troubleshooting became a nightmare.

A new solution had to be found.

4 SYSTEM

4.1 Terminology

To be concise in the description of this system, the following terminology is used.

- `server`: the central computer with a public IP
- `router`: the set of services that runs on said server
- `client`: a computer that resides in a LAN
- `peer`: the local instance of Gateway running on said client
- `user`: the person that uses said client
- `room`: the session a peer joins to exchange data
- `proxy`: UDP-proxy script running on the server
- `channel`: one of 20 data connections available within a room
- `peer-device`: the app running within a channel on a peer

4.2 Overview

There is a wide range of real-time content creation software available to create immersive media experiences, most of which now have the ability to communicate with each other: In addition to the usual suspects such as MIDI and OSC, the tools able for audio allow seamless connections between audio applications. Similar advances have been made for video applications with Spout[5] and Syphon[6], which allow textures to be shared on the GPU, a very efficient and low-latency solution. Other tools share their streams in native protocols via UDP multicast.

Unfortunately, there are limitations: Some of these solutions only work between applications running on the same client, others are limited to a simple network topology such as a LAN, or cannot be bridged to a VPN. From the point of view of such an application, Gateway is just another application running on a client like all the others, so all these limiting factors are out of the way.

Once Gateway is connected to a router and joined to a room, the UI provides access to the room matrix (Figure 1). Each row represents one of the peers connected to the room, with the local peer always in the top row. The columns represent the available channels. Each cell contains one peer-device (described in 5.5). Connections can only be established between two or more peers when they select the same device type within the same channel.

[5]Spout - texture sharing technology for Windows - https://spout.zeal.co/
[6]Syphon - texture sharing technology for OSX - https://syphon.github.io/

To realise a connection diagram as shown in Figure 2, the user dedicates a channel to each of the data streams to be transmitted, creates the peer-devices and configures them depending on whether they are of the sending or receiving type. The UI helps the user to find the available data streams that need to be sent and, similarly, on the receiving side, the data sinks to which the data needs to be passed on to.

The outstanding feature of the UI is its synchronicity with all other peers. It is possible to create, configure, start, stop and remove devices running on other clients on any of the Gateway's instances connected in the same room. This makes it possible to manage all the peers from one client or help remotely connected users to troubleshoot their potential problems.

Figure 1 shows how the interface looks like in case of a realized project that is described in this paper. In this special case it is the look from an additional peer that only monitors the connections.

5 IMPLEMENTATION DETAILS

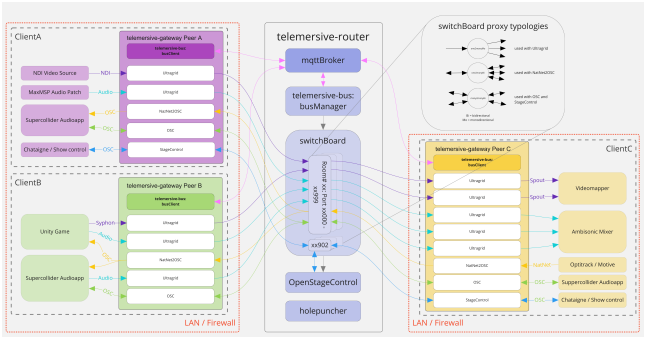

Figure 2: Detailed network diagram of the telemersive-gateway and the telemersive-router

5.1 Introduction

At first glance (Figure 2), the architecture may look like a classic server-client system. This is not the case. The server has no database. Instead, MQTT messages with a retained flag keep the system consistent. And there is no central instance that has any idea what a room's network topology looks like. This information is stored with the individual peers. It is therefore possible to set up the same network when peers choose to connect to a different router installed on a different server, without having to transfer any other data.

5.2 Router

The router (also known as the telemersive-router) is responsible for instantiating and managing rooms and relaying the UDP streams that need to be exchanged between peers. These tasks are divided into the following services:

5.2.1 The mqttBroker. is the backbone for the telemersive-bus, coordinating the peers and the router. Each peer must connect to this broker to access the available rooms. We are using the third party developed Mosquito Broker [23] for this purpose.

5.2.2 The busManager. is the server part of the telemersive-bus, a nodejs library, and can be thought of as the brain of the router. It is connected to the mqttBroker. The busManager is responsible for the live cycle of a room (described in 5.3).

5.2.3 The switchBoard. is a python script that starts and stops the relay-stations, the so called proxies (described in 5.4). It is accessed by the busManager via a REST-API.

5.2.4 OpenStageControl. [7] is a third party developed web based control surface application. Each room has its own instance. It is created and removed by the switchboard upon commands from the busManager. This instance plays no active role in the management of a room. It is an additional service that is available via its own device (described in 5.5).

5.2.5 holepuncher. is an idiosyncratic name for the third party developed nat-helper[8]. The service helps UltraGrid devices to establish peer-to-peer connections instead of sending the stream through the proxies. It does not play an active role in the management of a room. The installer for the router contains an already compiled binary.

5.3 Room

A room can be thought of as a session that peers join to share data. However, it is more than just a list of peers. To get a better picture of how the router's services work together, it is helpful to understand the life cycle of a room:

5.3.1 Connecting to the broker. A peer must first connect to the mqttBroker. If successful, it will receive a list of available rooms that are currently active on the router from the busManager.

5.3.2 Join non-existent room. If the peer tries to join a non-existent room, the busManager creates the room and sets its password, which cannot be changed during the live cycle of the room.

5.3.3 Create room. When creating a room, the busManager assigns a number to the room, starting with 11. This room number is also an indication of the ports assigned to this room on the server: if the room number is 11, the assigned ports are 11000 - 11999, if the room number is 13, the assigned ports are 13000 - 13999. However, not all ports are used. Because there are only 65525 ports available and the numbering starts with 11000, in theory the limits of rooms running on one server is 53. For practical reasons[9] the limit is rather with 49000 and so the number of rooms should not exceed 38, though no limitations is yet built into the busManager, for the simple reason we never exceeded 3 or 4 concurrent rooms, yet.

5.3.4 Create room proxies. Then the busManager commands switchBoard to create the proxies: Each room is assigned 20 channels, with 4 proxies running on each channel. Table 1 shows the port assignments.

[7] Open Stage Control - an open source web based control surface application - NOT to be confused with Open Sound Control - https://openstagecontrol.ammd.net/

[8] nat-helper - Hole punching coordinator for UltraGrid - https://github.com/CESNET/UltraGrid/tree/master/nat-helper

[9] The range 49152-65535 contains dynamic or private ports that could cause conflicts with other services

proxy typology		'one' port		'many' port
many2manyBi	on port:			xxcc9
one2manyMo	on ports:	xxcc2	->	xxcc6
one2manyMo	on ports:	xxcc4	->	xxcc8
one2manyBi	on ports:	xxcc0	<->	xxcc1

Table 1: Port assignments of UDP proxies, where xx = room nr and cc = channel nr

5.3.5 Instance OpenStageControl. The final task in room creation is to set up an instance of OpenStageControl, along with an additional many2manyBi proxy to communicate with it.

5.3.6 Concluding Room Creation. Once all instances and proxies are started, the busManager acknowledges the room creation to the requesting peer and returns the room number that the peer needs to make the correct connections from the individual devices running in its channels.

5.3.7 Join existing room. The busManager checks access attempts to join the room by other peers. It informs all connected peers in the room about peers joining or leaving the room.

5.3.8 Housekeeping. The busManager regularly checks and informs about the joined peers. If a peer doen't response to a housekeeping ping, the busManager will remove the peer from the room by giving notice to all joined peers.

5.3.9 Room Removal. The busManager removes the room when all peers have left or none of the joined peers respond to its pings. It initiates the removal of all associated proxies and the OpenStageControl instance.

To summarise. If a user wants to invite another user to the same room, he has to provide the access information to the mqttBroker and the name and password of the room. For a Gateway user, managing a room is as simple as providing a name and password. If the room doesn't exist, a new room will be created with the given name and password. If the room exists, the peer will access the room if the passwords match.

5.4 Proxy

A proxy is a Python script that is started and stopped by the switchBoard and listens on one or two ports, acting as a simple UDP relay station. The proxy allows two (or more) clients in separate private networks to exchange messages, even when none of the clients uses a public IP address. There are several types of proxies to support different topologies:

- `many2manyBi` is a many-to-many bi-directional proxy. Used by the OSC and StageControl device.
- `one2manyBi` is a one-to-many bi-directional proxy. Used by the natnet2OSC device.
- `one2manyMo` is a one-to-many mono directional proxy. Used by the UltraGrid device

For proxies to identify which clients should receive data, the client applications interested in the data stream must send a message (any byte sequence) at regular intervals. As long as the proxy

receives any message, it keeps the client's address in its destination list and forwards packets from the sender to all destinations. If after a predefined timeout (e.g. 10 seconds) no message has been received from a receiver, the proxy will remove its address from the destination list to keep network traffic to a minimum. For bidirectional proxies there is a specific message called heartbeat message that keeps the address in the destination list but is otherwise ignored and is not passed on.

This methods not only enables clients to exchange messages with other clients that are otherwise invisible to each other, it also allows to cross firewalls that are typically used in NATed networks.

UltraGrid can natively accomplish this because it is designed for bidirectional peer-to-peer connections. However, many applications rely solely on uni-directional UDP sockets (e.g., for communication via OSC) and are therefore incapable of sending and receiving messages on the same socket—a prerequisite for receiving data from proxies. Such applications require a bridge to receive data from the proxy. The OSC- and StageControl-device offer this service.

5.5 Peer device

There are currently 4 device types available:

- `OSC`: opens a bi-directional UDP bridge connection for general use. It allows to send and receive OSC data between individual peers.
- `StageControl`: opens a bi-directional UDP bridge connection to the running Open Stage Control instance and all the connected peers. It also provides a button to open the control interface of Open Stage Control in the default web browser.
- `UltraGrid`: configures and starts an UltraGrid instance for video and / or audio and allows to define different parameters (e.g. video and audio codecs, bitrate and framerate, number of audio channels, etc.). One UltraGrid instance (an therefore one Gateway channel) can deal with as many audio channels as an attached audio device can deliver.
- `natnet2OSC`: configures and starts a Natnet2OSC[10] instance and allows to define different parameters (e.g. address and argument shape, yup-to-zup or righthand-to-lefthanded transformation, etc). It converts Motive's[11] native data protocol to OSC and sends it further to listening peer-devices.

5.6 Gateway

The Gateway application is developed using Max[12]. It works mainly as a UI wrapper for the peer devices running inside externals, which keep the running processes in their own thread and away from Max's main scheduling thread. This ensures that the UI runs smoothly and responsively at all times.

To start and stop CLI applications like UltraGrid and NatNet2OSC from within Max, special externals were developed. Their job is to ensure that the sub-processes in which the CLI applications are started are also stopped and properly cleaned up, even if Gateway crashes for some unforeseen reason. This externals works on both Windows and OSX and are realized using Max-Go[13]. While the footprint of these externals are much larger than normal C or C++ externals for Max, their stability and reliability makes them worthwhile, as solutions based on the shell[14] object suffered from various problems, including unwanted daemons blocking ports and thus requiring machine reboots to get rid of, or firewalls triggering excessively.

The Gateway also stores the settings of the local peer and the enabled local devices. Once it joins a room, it loads the last saved state and starts all its local devices. It is possible to export and import settings.

The main logic of Gateway is in the nodejs script, the busClient.

5.7 busClient

The telemersive-bus is a nodejs library mode of two parts, the busManager - which was covered in detail in a previous chapter - and the busClient, which will be covered here. While the busManager takes care of the live-cycle of the rooms and their access, a busClient is the main administrative logic of a peer. It is responsible for keeping up to date with the internal states of all other connected peers. And it does this without any central oversight by using the MQTT protocol's retained messages in the following way.

The MQTT protocol is based on a publish/subscribe pattern: services connected to a broker[15] can publish or subscribe to topics[16]. For a service to keep up to date with specific topics, it only needs to subscribe to the topic and it will receive any published changes. If the topic is published with a retain flag, the last value of the topic is stored by the broker, and as soon as a new service subscribes to the topic, it gets the last value set to the topic.

Once a busClient has successfully joined a room, it will publish its current state (e.g. UI settings) to that topic with a retain flag:

```
tBus/rooms/<roomName>/<peerID>/<UI>/<current>/<state> <value>
```

where peerID is a UUID[17] assigned to a peer at startup. Then the busClient subscribes to the following topic:

```
tBus/rooms/<roomName>/#
```

By using the multi-level wildcard `#`, it gets all changes to all topics from all peers in the room it has joined. All busClient needs to do from here is to convert UI changes to topics and vice versa.

6 USE CASE

The project presented here is not the latest to use this tool, but it is the most comprehensive stress test to date in terms of the number of connections and machines using TTkit simultaneously.

6.1 Osmosis

Osmosis was a concert that highlighted the asymmetry between telematically connected spaces. It was performed during the 2022 edition of the NowNet Arts Conference on 31 October, with a live

[10] NatNet2Osc - Optitrack native motion data to OSC conversion tool - https://github.com/tecartlab/app_NatNetThree2OSC
[11] Motive Software by Optitrack - Motive pairs with OptiTrack cameras to track and capture motion - https://www.optitrack.com/software/motive/
[12] Max by Cycling'74 - https://cycling74.com/
[13] Max-Go - Toolkit for building Max externals with Go - https://github.com/256dpi/max-go
[14] Max Shell Object - https://github.com/jeremybernstein/shell
[15] MQTT broker is a server that receives all messages from services and then routes them to the appropriate destination services.
[16] MQTT topic is a string consisting of one or more levels separated by a forward slash (looks like an OSC address)
[17] Universally Unique Identifier (UUID) is a 128-bit label used for information in computer systems

Figure 3: Osmosis - Screenshot from Vimeo transmission

broadcast via OBS and Vimeo to the conference venue, combining a split-screen selection of available video sources and a binaural sound mix. [39] gives a comprehensive overview of the project, the theoretical background, a description of the scenographic setup of the two rooms and the complex audio routing involved. It does not go into the details of the video setup. This chapter aims to change that.

6.2 Setup

Although the two spaces (hereafter referred to as IAS and Action) had completely different scenographies, they shared some key aspects (see also Figure 4):

- `Video`: Both spaces had a dedicated MacPro with 4 cameras attached to a decklink capture card with OBS streaming/recording.
- `Mocap`: Both had an Optitrack Mocap system running
- `Projectors`: Both had a SPARCK[18] dynamic projection mapping running with two projectors. SPARCK got the required video feeds via Spout from the Gateway.
- `Audio`: Both had a dedicated MacBook Pro laptop for controlling the mulichannel audio streams and to facilitate the sound mix to the loudspeaker systems (with a 37 loudspeaker array for Ambisonic rendering in the IAS room)

The Action room differed in that it had an additional 3 Realsense[19] cameras attached to a computer that captured them using the python-based Space Stream script[20]. An additional projector was also attached to the same machine. All of the machines shown had an instance of Gateway running.

Figure 1 is a screenshot of the Gateway GUI just after the show and depicts the same setup as the connection diagram seen in Figure 4. A total of 31 video, 4 mocap and 4 OSC streams were used, either to or from the router. The video streams were full HD, the Realsense streams had a resolution of 1700x480. All were encoded in h.264 and the bitrate limit was set at 20Mbit, resulting in excellent quality with no noticeable glitches. For the multichannel audio streaming, the TPF-Tools were used independently from TTkit.

[18] Spacial Augmented Reality Contruction Kit (SPARCK) https://github.com/immersive-arts/Sparck2. See also https://tecartlab.com.
[19] Intel Realsense D455 - https://www.intelrealsense.com/depth-camera-d455/
[20] Space Stream - Send RGB-D images over spout / syphon - https://github.com/cansik/space-stream

In more recent projects where synchronicity between audio and video streams was more important than ultra-low audio latency, we streamed multichannel audio through TTkit via UltraGrid as well and with very stable results.

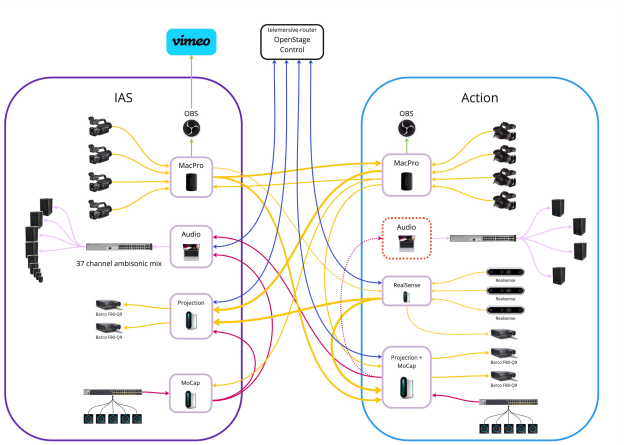

Figure 4: Osmosis spaces and their devices and the connecting data streams for video, mocap and control

6.3 Findings

While the smooth video reception was obviously a testament to our university internet backbone, the success of the resulting project proved the concept of this type of system architecture. This was particularly important as there was only this one performance in a fixed time slot, with a live audience on the other side of the planet watching our efforts.

OpenStageControl was used in this setting for the first time to see if it could be used as a show control solution. The four OSC streams were the link to this instance. We were able to reliably trigger all cues during the show with the browser-based tool and the experience convinced us to make it a permanent part of the TTkit.

7 CONCLUSION

We have successfully developed a robust, easy to install and open source software that extends the functionality of an already existing and feature rich tool like UltraGrid, with added OSC streaming capabilities as well as an OpenStageControl integration for show control. This enables the realization of highly complex multimedia system architectures spanning many machines, physical locations and network infrastructures. The experience gained from several projects that relied heavily on the new capabilities helped to refine and fine-tune usability and reliability.

The peer synchronization feature of the UI has been proven to be a viable approach, giving one person the power to monitor, maintain and control the entire network from any one of the Gateway instances. It is now the core feature of the experience of interacting with TTkit, alongside all the other features.

We were also able to show how the tool can be used in a real world setting that pushed the boundaries by sending more than

30 simultaneous video and mocap data streams and successfully interfacing with a wide range of real-time content creation software, enabling the creation of two telematically linked immersive experiences, giving credence to the given name of a Telemersive Toolkit[21].

ACKNOWLEDGMENTS

The development of TTkit was realized by the Telematic Perfomance Format[22] Research Group and its research project "Spatial Dis/Continuities in Telematic Performances", funded by the Swiss National Science Foundation. The TPF group is based at the Institute for Computer Music and Sound Technology and the Immersive Arts Space of the Zurich University of the Arts. The group consists of Patrick Müller, Benjamin Burger, Joel De Giovanni, Martin Fröhlich, Roman Haefeli, Eric Larrieux, Johannes Schütt, Hannah Walter and Matthias Ziegler. We would like to thank the members of our research team and all the other users of the prototypes, whose patience and feedback have made it what it is today. Special thanks goes to the developers of UltraGrid, who were open to suggestions and quickly implemented features in line with their philosophy, and to Joël Gähwiler for writing the custom max externals.

REFERENCES

[1] Ross Bencina. 2024. OSCgroups. http://www.rossbencina.com/code/oscgroups Accessed: 2024-1-24.
[2] Digitale Bühne. 2024. digital-stage. https://digital-stage.org/startseite Accessed: 2024-1-25.
[3] Alexander Carôt. 2024. Soundjack. https://www.soundjack.eu/ Accessed: 2024-1-25.
[4] Alexander Carôt, Christian Hoene, Holger Busse, and Christoph Kuhr. 2020. Results of the Fast-Music Project—Five Contributions to the Domain of Distributed Music. *IEEE Access* 8 (2020), 47925–47951. https://doi.org/10.1109/access.2020.2979362 Related Work: Research to create low latency connections for Audio and Video in distributed networked music collaborations - related to soundjack.
[5] Chris Chafe, Scott Wilson, Randal Leistikow, Dave Chisholm, and Gary Scavone. 2000. A SIMPLIFIED APPROACH TO HIGH QUALITY MUSIC AND SOUND OVER IP. In *Proceedings of the COST G-6 Conference on Digital Audio Effects*. https://dafx.de/paper-archive/index.html
[6] Teresa Chambel, Paula Viana, V Michael Bove, Sharon Strover, Graham Thomas, Rene Kaiser, Marian F Ursu, Manolis Falelakis, and Andras Horti. 2015. Enabling Distributed Theatre Performances through Multi-Camera Telepresence. *Proceedings of the 3rd International Workshop on Immersive Media Experiences* (2015), 21–26. https://doi.org/10.1145/2814347.2814351 Related Work: this system emphasises the lines of communications between the different performance locations, their individual local audioences and home audience and an authoring tool to control and mix all these different communcation channels.
[7] Maria Chatzichristodoulou. 2014. Cyberformance? Digital or Networked Performance? Cybertheaters? Virtual Theatres?… Or All of the Above? In *CyPosium - the book*, Annie Abrahams and Helen Varley Jamieson (Eds.). Link Edition, Brescia, 19–30. www.lulu.com
[8] John Crawford. 2005. Active space. *ACM SIGGRAPH 2005 Sketches on - SIGGRAPH '05* (2005), 111–es. https://doi.org/10.1145/1187112.1187246 Introduction: Active space - interactive system for realizing networked multi-site performances.
[9] CultureHub. 2024. LiveLab. https://www.culturehub.org/livelab Accessed: 2024-1-25.
[10] Juan-Pablo Cáceres and Chris Chafe. 2010. JackTrip: Under the Hood of an Engine for Network Audio. *Journal of New Music Research* 39, 3 (2010), 183–187. https://doi.org/10.1080/09298215.2010.481361 Related Work: Jacktrip: a low latency audio peer to peer transmission framework.
[11] Steve Dixon. 2007. Digital Performance - chapter 17 - Telematics- Conjoining Remote Performance Spaces. In *Digital Performance: A History of New Media in Theater, Dance, Performance Art, and Installation*. 419–435. https://doi.org/10.7551/mitpress/2429.003.0024 Introduction.
[12] Carlo Drioli, Claudio Allocchio, and Nicola Buso. 2013. Networked Performances and Natural Interaction via LOLA: Low Latency High Quality A/V Streaming System. In *Information Technologies for Performing Arts, Media Access, and Entertainment (Lecture Notes in Computer Science)*. 240–250. https://doi.org/10.1007/978-3-642-40050-6_21 Related Work: Creating the basic infrastructure of low latency tech and then develop a desktop app to access the network: Installation is not for the faint hearted and requires very specific hardware setup..
[13] Elk.audio. 2024. Elk. https://www.elk.audio/ Accessed: 2024-1-25.
[14] Farplay. 2024. Farplay. https://farplay.io/ Accessed: 2024-1-25.
[15] Kenneth Fields. 2012. Syneme: Live. *Organised Sound* 17, 1 (2012), 86 – 95. https://doi.org/10.1017/s1355771811000549 Introduction: Tools and Works using state of the arts and custom build software like artsmesh.
[16] Volker Fischer. 2024. Jamulus. https://jamulus.io/ Accessed: 2024-1-25.
[17] GARR. 2024. LoLa. https://lola.conts.it/ Accessed: 2024-1-25.
[18] Roman Haefeli, Johannes Schütt, and Patrick Müller. 2019. TPF-TOOLS - A MULTI-INSTANCE JACKTRIP CLONE. In *Proceedings of the 17th Linux Audio Conference (LAC-19)*. http://lac.linuxaudio.org/2019/doc/haefeli.pdf Related Work: Convenient Userinterface around core audio low latency infrastructure.
[19] Petr Holub, Luděk Matyska, Miloš Liška, Lukáš Hejtmánek, Jiří Denemark, Tomáš Rebok, Andrei Hutanu, Ravi Paruchuri, Jan Radil, and Eva Hladká. 2006. High-definition multimedia for multiparty low-latency interactive communication. *Future Generation Computer Systems* 22, 8 (2006), 856–861. https://doi.org/10.1016/j.future.2006.03.014 Ultragrid paper.
[20] iVisit. 2024. iVisit. https://download.cnet.com/ivisit/3000-2150_4-10013125.html Accessed: 2024-1-25.
[21] Helen Varley Jamieson. 2008. *Adventures in Cyberformance*. Ph. D. Dissertation.
[22] Wolfgang Jäger. 2010. *Audio over OSC*. Technical Report. https://phaidra.kug.ac.at/detail/o:11413
[23] Roger A Light. 2017. Mosquitto: server and client implementation of the MQTT protocol. *The Journal of Open Source Software* 2, 13 (2017), 265. https://doi.org/10.21105/joss.00265
[24] Sonosaurus LLC. 2023. Sonobus. https://sonobus.net/ Accessed: 2024-1-25.
[25] Patrick Müller, Johannes Schütt, and Matthias Ziegler. 2019. Towards Telematic Dimension Space. In *Proceedings of the International Conference on New Interfaces for Musical Expression*. 393–400.
[26] L. Naugle. 1998. Digital dancing. *IEEE MultiMedia* 5, 4 (1998), 8–12. https://doi.org/10.1109/93.735864 Introduction: Dancing performances based on early video conferencing tool CU-SeeMe.
[27] Lisa Marie Naugle. 2002. Distributed Choreography: A Video-Conferencing Environment. *A Journal of Performance and Art* 24, 2 (2002), 56–62. http://www.jstor.org/stable/3246553 introduction: early adopters of video conferencing environments for distributed choreography.
[28] PictureTel. 2024. PictureTel. https://de.wikipedia.org/wiki/PictureTel_Corporation Accessed: 2024-1-25.
[29] Miller Pucket. 2024. Quacktrip. https://msp.ucsd.edu/tools/quacktrip/ Accessed: 2024-1-25.
[30] Benjamin Paul Redman. 2021. *Utilizing Internet-Based Videoconferencing for Instrumental Music Lessons*. Ph. D. Dissertation.
[31] Héctor Rivas, Ana Domínguez, Stefano Masneri, Iñigo Tamayo, Mikel Zorrilla, Pedro Almeida, Alina Striner, Jie Li, and Pablo Cesar. 2021. Co-creation Stage-a Web-based Tool for Collaborative and Participatory Co-located Art Performances.pdf. In *Proceedings of the 2021 ACM International Conference on Interactive Media Experiences*. 267–274. https://doi.org/10.1145/3452918.3465483 Related Work:web client based audio and video distribution system for collaborative performances..
[32] Cristina Rottondi, Chris Chafe, Claudio Allocchio, and Augusto Sarti. 2016. An Overview on Networked Music Performance Technologies. *IEEE Access* 4 (2016), 8823–8843. https://doi.org/10.1109/access.2016.2628440 Introduction: Music performances specific need for low latency solu.
[33] source live. 2024. source-live. https://www.source-elements.com/products/source-live/ Accessed: 2024-1-25.
[34] Alina Striner, Thomas Röggla, Mikel Zorrilla, Sergio Cabrero Barros, Stefano Masneri, Héctor Rivas Pagador, Irene Calvis, Jie Li, and Pablo Cesar. 2022. The Co-Creation Space: Supporting Asynchronous Artistic Co-creation Dynamics *(Companion Computer Supported Cooperative Work and Social Computing)*. 18–22. https://doi.org/10.1145/3500868.3559459 Related work: Cocreation social network for collaborating in the design and rehearsal of perfomative art works.
[35] Syneme. [n. d.]. Artsmesh. https://www.artsmesh.com/ Accessed: 2024-1-25.
[36] Atau Tanaka. 1999. Network Audio Performance and Installation. In *ICMC Preceedings*, Vol. 1999. http://hdl.handle.net/2027/spo.bbp2372.1999.460
[37] VDO.Ninja. 2024. Video Ninja. https://vdo.ninja/ Accessed: 2024-1-25.
[38] Wikipedia. 2024. CUSeeMe. https://en.wikipedia.org/wiki/CU-SeeMe Accessed: 2024-1-25.
[39] Matthias Ziegler. 2023. Osmosis Asymmetries in Telematic Performance. *Journal of Network Music and Arts* 5, 1 (2023).

[21] Telemersive Toolkit - https://github.com/telemersion/
[22] https://networkperformance.space/