The MusiNet project: Addressing the challenges in Networked Music Performance systems

D. Akoumianakis*, C. Alexandraki†, V. Alexiou‡, C. Anagnostopoulou§, A. Eleftheriadis‡, V. Lalioti§, Y. Mastorakis‖**, A. Modas‡, A. Mouchtaris‖** , D. Pavlidi‖**, G. C. Polyzos††, P. Tsakalides‖** , G. Xylomenos††, P. Zervas†

* TEI of Crete, Dept. of Informatics Engineering, Heraklion 71500, Greece

† TEI of Crete, Dept. of Music Technology and Acoustics Engineering, Rethymnon 74100, Greece

‡ University of Athens, Dept. of Informatics and Telecommunications, Athens 15784, Greece

§ University of Athens, Dept. of Music Studies, Athens 15784, Greece

‖ University of Crete, Dept. of Computer Science, Heraklion 70013, Greece

** FORTH-ICS, Heraklion 70013, Greece

†† Athens University of Economics and Business, Dept. of Informatics, Athens 10434, Greece

***Abstract*—This paper presents the progress in the MusiNet research project, which aims to provide a comprehensive architecture and a prototype implementation of a Networked Music Performance (NMP) system. We describe the Musinet client and server components, and the different approaches followed in our research effort in order to culminate in the most appropriate scheme in terms of delay and quality for the audio and video streams involved. We also describe the MusiNet user interface, which allows an integrated communication between the participants and the proposed NMP system.**

I. Introduction

Networked Music Performance (NMP) systems can have a catalytic effect in music creation and social interaction, since they allow collaboration between musicians in geographically remote and disparate locations. Lowering the cost of collaboration, such systems can promote music creativity, education and cross-cultural interaction, enabling live performances, rehearsals, improvisation or distant music learning.

The appearance of high speed research/educational computer networks, such as Internet2 in the USA and GEANT in Europe, triggered the appearance of NMP system projects, such as Jacktrip [1], Distributed Immersive Performance (DIP) [2], SoundJack [3] and DIAMOUSES [4]. However, these systems either pose significant limitations to NMP interaction, or require excessive amounts of resources and direct access to high speed networks.

In our previous work [5] we introduced the MusiNet project, which aims at significantly reducing the resource requirements of NMP and studying in depth the sociological and musicological aspects of such systems. The central objective of the MusiNet project is to address the main challenges of NMP systems. The most crucial one is the elimination of latencies during the capturing, transmission, reception and reproduction of the audiovisual information. The maximum delay between any two endpoints must be kept below 25 msec [6] in order not to hinder musicians participating in an NMP session. Other challenges include the elimination of network bandwidth bottlenecks, the synchronization of the exchanged information, and handling the high sensitivity of NMP to network data loss. In this work we present the progress accomplished towards unraveling the full potentials of NMP systems, by outlining implementation aspects of the primary components of MusiNet, i.e., the client, server and user interface.

Correspondence should be addressed to e-mail: mouchtar@csd.uoc.gr.

The remainder of this paper is structured as follows. Section II-A presents the Musinet client, Section II-B presents the different implementation approaches for the MusiNet server, while Section II-C describes the user interface of the proposed system and Section II-D describes a novel method for real-time delay estimation of audio streams. We present evaluation and comparative results for the proposed schemes in terms of delay in Section III. We conclude and refer to future work in Section IV.

II. Advances in the MusiNet system

In this section we describe our work on the primary components of MusiNet, i.e., the MusiNet client and the MusiNet server, shown in Fig. 1. We also describe in Section II-C the user interface that we designed, which enables a natural interaction between the system and the remote participants.

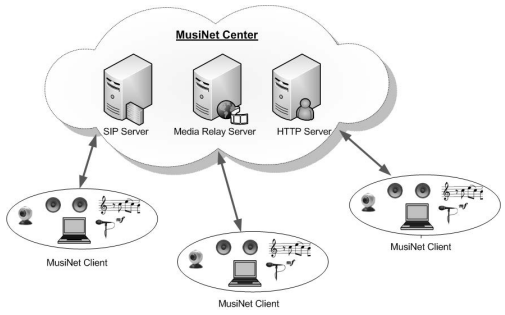


Fig. 1. The general architecture of MusiNet.

A. The MusiNet Client

For the MusiNet client, we based our implementation on BareSIP [7], an open-source portable and modular Session

Initiation Protocol (SIP) User Agent with audio and video support. Although BareSIP is a minimalistic VoIP client, it supports the vast majority of state-of-the-art audio and video codecs (including the ultra-low delay OPUS audio codec [5], [8]) and numerous networking and communication protocols. Despite its large list of supported features, BareSIP is robust and fast with a low memory footprint. BareSIP depends on two libraries: `Libre`, a generic library for real-time communications with asynchronous I/O support, and `Librem`, a portable library for real-time audio and video processing. BareSIP, along with its dependencies, is an open source project compliant with the C89/C99 language standards. It employs a well-designed, object-oriented, highly-modular structure, that makes it suitable for extensions or modifications.

BareSIP in its pure form is restricted to Peer-to-Peer (P2P) communication. For example, say that a user has established a P2P call with a second user, and wants to start another P2P call with a third user. For each new call, a new instance of BareSIP is generated and a new session is established, independently from existing sessions; in order for a user to connect to N other users, N instances of BareSIP should be instantiated, N sessions should be set up, and N encoding and decoding processes should be executed at his/her terminal. To adapt BareSIP to the mode of operation required by MusiNet, which involves the connection of multiple users to a conference room, we have to modify BareSIP's architecture. Specifically, a single BareSIP instance should be used, with a single session established between the user and the server. Depending on the architecture of the server (discussed in Section II-B), either a single encoding and decoding process should be performed at each user's terminal, or one encoding and N decoding processes, for a conference room with a $N + 1$ participants.

In a typical conferencing application, the server mixes the multiple audio and video streams in a single stream which is transmitted to the clients. However, in a NMP system it is better for the participants to be able to create their own local downmix. For this reason, the system architecture of MusiNet includes a Selective Forwarding Unit (SFU) [9], [10], which simply relays each incoming stream to all other participants. This avoids the delays due to transcoding, but burdens clients with the downmix process of numerous streams. For a conference room with $N + 1$ participants, the server receives at least $N + 1$ streams (e.g., when only audio is involved) and retransmits each of these streams N times. Similarly, each user receives at least N streams, one from each other participant.

We have considered two ways of multiplexing the individual streams to be transmitted to each participant: (a) Real Time Protocol (RTP) session multiplexing, and (b) SSRC multiplexing.[1] The first technique uses a separate RTP session for each stream, ending up in a different port at the receiver. The second technique uses a single session for all streams *of the same media type*, hence all, say, audio packets end up in the same port of each client, distinguished only by their SSRC. Our current practice is to use the second technique, since by using fewer sessions and ports it simplifies network address translator (NAT) traversal. However, in this scheme, upon reception of an audio or video packet the client has to associate it with a participant, based on side information provided by the server. For this reason, we also considered adding user IDs to separate media streams. These IDs are injected to the audio and video packets at the server, before the packets are relayed to all participants. This allows uniquely identifying the audio and video packets of each user with a single ID (for more details on this solution see Section II-B2).

Another structural characteristic of BareSIP is that, by default, a single thread per port is created. In our case, however, with SSRC multiplexing each client receives N individual streams at the same port. As the number of participants increases, system performance is expected to decrease, increasing the average packet processing delay. A key MusiNet requirement is to minimize the total end-to-end delay, thus it is important to eliminate delays in all system components. To address this issue, we developed a multi-threaded approach for the receiving part of each terminal, which is further described below. The transmitting part remains identical.

Initially, when a user joins a conference room through our server, all transmission operations of the User Agent are initialized. When that user receives the session configuration information from the server (i.e., the number and properties of the other participants) a list of nodes is locally created. Each node represents a remote user and stores all relevant information. A new thread per media stream is created for each user, with the IDs of these threads being stored in the list of nodes. Each such thread is responsible for handling the received packets associated with a particular transmitter and SSRC. The single thread normally created per port by BareSIP when a session is established, simply acts as a demultiplexer, with the task of identifying the media type of a received packet and its sender (based on the SSRC), and delivering that packet to the appropriate media processing thread. Finally, when a user hangs up a call, the other participants are informed, hence the dedicated processing threads and the resources that keep his/her relevant information are released at each terminal.

B. The MusiNet Server

For the MusiNet server we have pursued three parallel lines of development: (a) modifying FreeSwitch [12], an existing SIP server that offers audio Multipoint Control Unit (MCU) functionality, (b) developing `pktswitch` [13], an ultra-low delay Selective Forwarding Unit (SFU) and (c) modifying the BareSIP client so as to operate as a transparent server with minimal operations; we will refer to this server-oriented BareSIP modification as `BareServer`.

On the FreeSwitch front, our work has focused on replacing the existing audio mixing functionality with a straightforward packet relaying functionality for both audio and video streams. While FreeSwitch has the advantage of supporting SIP functionality required for the MusiNet system, such as user presence notifications, its basic design as an audio mixer

[1] SSRC, which stands for synchronization source identifier, is an integer number that uniquely identifies a stream in an RTP session [11].

makes it hard to optimize for delay. For example, rather than immediately relaying received packets, FreeSwitch tries to align multiple packet streams in order to synchronize their mixing, a redundant and costly (in terms of delay) operation.

Therefore, our research has focused on `pktswitch`, which follows the latest research developments in high performance network I/O systemsbut lacks SIP support, and on `BareServer`, which complies with the project's strict latency constraints, according to our preliminary experimental results, and also supports SIP functionality.

1) The `pktswitch` SFU: The basic concept behind `pktswitch` is to exploit the netmap [14] packet handling framework in order to minimize packet processing delay [15]. In order to assess the validity of our approach, we first implemented a baseline, socket-based, SFU which handles a single type of control packet, indicating new client registrations, and three types of media packets, audio, video base layer and video enhancement layer. A pre-configured table in the SFU shows which media packet types each client wishes to receive. The SFU consults that table for each incoming packet and sends a copy of the packet to each interested client. A final copy is sent back to the originating client as an "echo" packet. A test client periodically transmits packets of each type to the SFU, tracking the time until each "echo" packet arrives. Note that the test client and server do not support SIP but rely on a custom protocol and pre-configured settings.

The disadvantage of the socket-based SFU is that each socket call can only pass a single packet from the *Network Interface Card* (NIC) to the SFU and vice versa, requiring a context switch between kernel and user space; one call is needed to receive the packet and one call is needed to send the packet to each recipient. This overhead grows with the number of clients. Since the SFU barely modifies the media packets, we have resorted to the netmap framework to minimize this overhead. With netmap, the SFU waits for packets from either the *netmap RX* (NRX) ring, connected to the NIC, or the *host RX* (HRX) ring, connected to the host networking stack. The SFU issues a `poll` system call to wait for new packets. When this call returns, if there are new packets in the NRX ring, they are all moved to user space, without further kernel mediation.

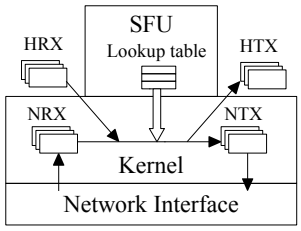


Fig. 2. The netmap-based `pktswitch`.

Media packets are passed to a media handler, which inspects the packet header and consults the SFU table to determine which clients should receive it. The packet is copied as many times as needed to the *netmap TX* (NTX) ring, connected to the NIC, simply changing its destination address to that of the desired client. The incoming packet is also moved to the NTX ring, to be sent as an "echo" packet to the originating client. Control packets are processed by a control handler, updating its lookup table. Finally, packets destined to the host OS are copied to the *host TX* (HTX) ring, i.e., the host networking stack, for processing by other applications. As a result, packets are always moved to/from user space in large batches, eliminating redundant context switches. An overall view of the application layer SFU is given in Fig. 2.

2) The BareServer: Since BareSIP already supports multiple concurrent peer-to-peer calls and has a modular design which simplifies extensions, we have modified it to behave as a server, the BareServer. The BareServer is a minimal SFU, simply relaying packets, with no transcoding or mixing.

Each BareSIP client makes a peer-to-peer call to the BareServer as it would normally do with another plain BareSIP client. All necessary protocol negotiations needed in order to establish the call are supported by default. When a connection is established between a BareSIP client and the BareServer, the client transmits its own local audio and video streams, but BareServer's default local audio and video transmit is disabled. In Section II-A we referred to two possible ways for media stream demultiplexing, one based on SSRCs and one based on user IDs. In the first case, the BareServer simply retransmits a received packet to all other clients. In the second one the BareServer, after the RTP header is decoded, re-encodes it including the user ID in the first CSRC slot of the RTP header,[2] which is not used for any other purpose in our implementation, and transmits the packet to all participants involved.

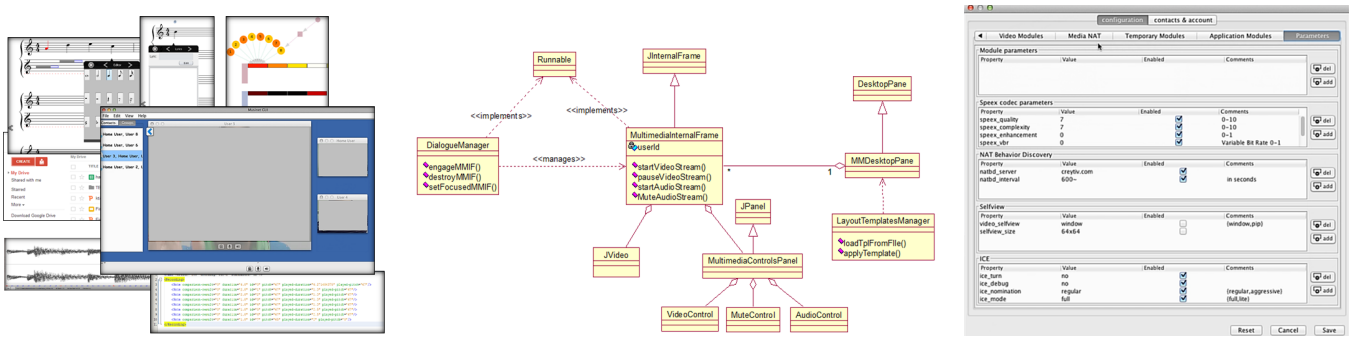
Using either approach, every client connection to the BareServer will be handled as a different call, so it is important to increase the maximum number of calls according to the anticipated maximum number of participants, which by default is set to four. The side information (e.g., the user ID) about room participants is transmitted every 1 sec through Real Time Control Protocol (RTCP) application-specific messages. For each client, its user ID, its uniform resource identifier (URI) and its alias name are included in the side information.

To achieve higher throughput for the BareServer, a threaded implementation is applied. For each client, two threads are created, one for audio and one for video. The task of each thread is to relay its client's packets to all other connected clients. A producer-consumer model is used, i.e., when a new audio or video packet from a client is received, it is appended at its audio or video packet list respectively. The threads read those lists and relay the packets sequentially, in the same order they were received.

C. The MusiNet HCI design and media configuration

With respect to interaction design, the approach followed rests on the "material turn" in human-computer interfaces (HCI) [16]–[18]. In terms of design commitments, the material-turn in HCI signifies a redirection of attention from

[2]CSRC, which stands for contributing source identifier, is an integer number that uniquely identifies one of the sources contributing to a translated stream, identified by the SSRC of the translating entity.



(a) (b) (c)

Fig. 3. (a) The UI and the underlying digital materials. (b) Structure of the container object. (c) Configuration of parameters, contacts and accounts.

digital artifacts or the application layer of interaction design toward the materials from which digital products and services are built in the first place. For our purposes, a material turn entails configuring different genres of digital technologies implicated in computer-mediated collaborative music making and performance in such a way that they operate interdependently, as the need arises. Thus, a prominent issue is treating technologies such as BareSIP, file sharing services, an augmented graphical toolkit for music composition and 2D visualization, as digital materials from which virtual experiences are built. At the core of this effort has been the appropriation (or crafting of new) Application Programming Interfaces (APIs) to ensure that these technologies act interdependently and constitute a coherent whole. As Fig. 3(a) indicates, this layering of different digital materials enables loose coupling of distinct functionalities that may be invoked as needed either prior, during or after network music performance sessions.

To address the challenges, several lines of technical developments have been undertaken. Firstly, dedicated container objects were implemented (see Fig. 3(b)) to host media types from BareSIP. In addition, a multithreaded dialogue manager was designed on top of the window manager to handle the users' entry to and exit from the workspace, as well as the in-coming events to be delegated to appropriate container objects for rendering multimedia content. Thirdly, a custom component was used to bridge the gap between the Java User Interface API and the BareSIP core API. Finally, several native modules were implemented to allow configuration of BareSIP parameters and user contact lists. An instance depicting parameter configuration is presented in Fig.3(c).

D. Real-Time Delay Estimation of Audio Streams

Since audio signal delay is the most significant parameter in enabling music collaboration [19], [20], we need a method to accurately quantify the end-to-end delay of audio streams. In order to measure one-way delay in real-time audio communication between distant endpoints, we have resorted to the use of pilot signals. Taking into consideration the general properties of music and the human auditory system, we designed the pilot signal so as to minimize its audible effects, considering that there is a trade-off between the system's delay estimation accuracy and the pilot signal's audibility. We provide the key quantities that can be modified correspondingly, with respect to the targeted/desirable application mode.

At the transmitter, we split the audio signal in time intervals equal to the duration of an audio packet. Then, we perform a spectral analysis and properly adjust the pilot signal, avoiding any undesirable additive noise. We also designed an algorithm to estimate the one-way delay based on the receiver's local clock, given an initial time reference of the first pilot signal's injection at the transmitter. The proposed system operates in real-time, adding a negligible delay to the total end-to-end latency of the audio stream due to the pilot signal's adaptation processing. Additionally, the structure of the pilot provides multiple access capability [19]. Initial simulations verify the effectiveness of the adopted techniques. As a next step, we will embed this technique to the MusiNet system in order to allow each user to get informed about the delay of each audio stream received while the system is actually being used.

III. Evaluation of the MusiNet System

In this section we evaluate our proposed system through real experimental setups. In the first set of experiments we measure the delay induced by the `pktswitch` SFU (Section II-B1). In the second set we compare the default implementation of the BareSIP client with the modified version described in Section II-A, using the BareServer SFU (Section II-B2).

A. Server-induced delay estimation

To assess the delay induced by the SFU (see Section II-B1), we measured the difference between the time a media packet was picked for processing and the time its final "echo" packet was scheduled to be sent. We used a set of identical machines using the Realtek RTL8111/8168B Gigabit Ethernet controller and running Linux kernel 3.2.63, connected via an isolated Gigabit LAN, with one machine acting as the SFU. Each experiment consisted of a warmup period, followed by a 60 s period of measurements. For efficient and high-resolution delay measurements, we counted CPU cycles. To ensure repeatability, we ran each experiment multiple times and verified that results had very low variance. Figure 4 shows the packet processing delay measured for sessions with 6 and 8 clients. The media payload sizes are set to 80, 160 and 320 bytes for the audio, video base and video enhancement streams,

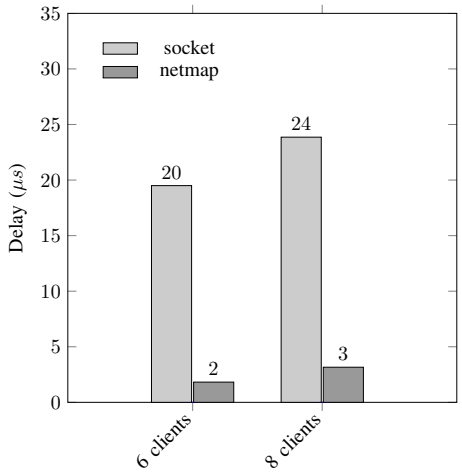


Fig. 4. Packet processing delay: clients streaming at 750 pps (1.12 Mbps).

respectively, and a new packet per stream is transmitted every 4 ms, leading to or 750 packets per second (pps), or a combined data rate of 1.12 Mbps per client. The netmap-based SFU shows significant improvements, with a delay that is on average only 11.3% of the delay of the socket-based SFU. Further experiments show that the netmap-based SFU requires only 25% of the packet processing overhead of the socket-based SFU, which means that it can handle four times the load of the socket-based SFU before packet queuing is needed to handle the increased load [13].

B. Delay estimation experiments of the MusiNet System

In this experimental set we compare the default implementation of the BareSIP client with the modified version described in Section II-A, considering sessions with two to five participants. With two users we can directly compare two different but equivalent systems, while with more than two users we can evaluate the performance of the multi-threading receiver developed to support conferencing. We also estimate the time delay of the server processing per received packet.

The total time for media transmission at each client is the sum of the individual time delays for capturing media, encoding media and transmitting the corresponding RTP packets. Media capture is performed by an external module, using appropriate APIs. Thus, a video or audio handler is responsible for transferring captured data to the main BareSIP code to be encoded, packetized and transmitted. For audio transmission, the recording buffer used to store the samples from the audio source is of size equal to the packet duration, i.e., (`ptime`); when filled, the specified handler is called. The captured data correspond to a single packet, and thus, the encoding, packetization and transmission of a single packet is performed directly by BareSIP's main code. For video, when a frame is captured, the corresponding handler is called. BareSIP undertakes the encoding, packetization and transmission of video packets, as well, but in this case, the video encoder receives an entire frame and returns the encoded data that may be included in one or more fixed size (`pktsize`) RTP packets. In both cases, using the system call `gettimeofday`, we estimate the time needed for the transmission of a single packet, excluding the functions responsible for audio and video capturing. Therefore, these numbers reflect the performance of BareSIP's main code.

Another parameter affecting delay is the packet size. A larger buffer means that bigger intervals of the audio source are captured, increasing the end-to-end delay, since we have to wait more for the buffer to fill. As mentioned above, in our performance evaluation we ignored capture delay. Increasing the packet size, however, also means handling larger packets, which leads to increased processing delay per packet, although there are fewer packets transmitted per stream.

The experiments were conducted on a LAN, where we made a set of calls with nearly identical video/audio content lasting $\approx$ 60 sec; all results refer to average values. The measurements were performed on an Apple Mac mini, running a single instance of BareSIP, with additional participants hosted on a second one that ran one instance of BareSIP for each participant. The client computer configuration involved an Intel Core i5 processor at 2.5 GHz running MacOS X version 10.9.5. The Opus codec was used for audio coding and H.263 for the video coding, CoreAudio for audio capture and playback, AVCapture API for video capture, and OpenGL for video display. The server was built on a MacBook Pro with an Intel Core i7 processor at 2.7 GHz running MacOS X 10.9.5.

In Table I we present the processing delay per packet in microseconds (μsec), for audio and video. "P2P" indicates the use of the default, peer-to-peer version of BareSIP, without an intermediate server. The abbreviation "VC" stands for our video conferencing mode, with the integer next to it denoting the number of participants. In this configuration all streams are routed through a media server.

TABLE I

Delay of audio transmission per packet (μsec)					
ptime	**P2P**	**VC 2**	**VC 3**	**VC 4**	**VC 5**
20	166.54	170.08	173.11	174.6	173.24
10	115.39	120.31	120.92	120.92	120.9
5	72.6	76.36	76.19	76.4	79.74
Delay of video transmission per packet (μsec)					
pktsize	**P2P**	**VC 2**	**VC 3**	**VC 4**	**VC 5**
1024	1663.98	1778.97	1804.79	1797.44	1821.24

For both audio and video transmission, we observe that as the number of participants and the packet size are increased, the processing delay generally increases. In the case of audio, by decreasing the packet duration at each tested mode we notice a significant delay reduction of up to 57%. For both audio and video, by increasing the number of users and keeping the packet size fixed, we observe a delay increase of up to 10% in terms of μsec, which does not depend on the number of clients, making the difference negligible.

At each client, the total time for media reception is the sum of time for receiving RTP packets, jitter buffer processing, decoding and rendering the media. Media rendering is performed by an external module. After the decoding operation, specific handlers are called to either store the decoded audio samples to the playout buffer, or to display the video frame. In the case

of video, a single packet may include a whole, or a part of a frame. In both cases, using the system call `gettimeofday`, we estimate the time needed for the reception of a single packet, excluding the system functions responsible for audio and video reproduction. Besides the media content and the packet duration, another parameter that significantly affects the receiving operation is the size of the jitter buffer, which is set in terms of frames/packets. In Table II we present the processing delay per packet in msec for audio and video, respectively, including the jitter buffer size as an extra parameter.

TABLE II

Delay of audio reception per packet (msec)						
jbuf	**ptime**	**P2P**	**VC 2**	**VC 3**	**VC 4**	**VC 5**
10	20	199.658	200.087	200.069	199.914	200.083
10	10	99.834	100.065	100.064	100.034	100.025
10	5	49.989	50.045	50.025	50.028	50.014
5	20	99.773	100.099	100.099	100.078	100.048
5	10	49.959	50.059	49.956	50.049	50.031
5	5	25.011	25.047	25.046	25.044	25.037
1	20	20.003	20.097	20.122	20.098	20.056
1	10	10.029	10.073	10.057	10.065	10.061
1	5	5.031	5.056	5.062	5.059	5.060
Delay of video reception per packet (msec)						
jbuf	**pktsize**	**P2P**	**VC 2**	**VC 3**	**VC 4**	**VC 5**
10	1024	206.363	274.162	285.208	275.401	283.088
5	1024	105.546	129.341	140.409	137.017	139.058
1	1024	20.652	21.865	23.623	23.001	23.778

For audio reception (the most critical issue for MusiNet), keeping the packet duration and the jitter buffer size fixed, the estimated delay time is almost identical for each tested mode. Specifically, the estimated delay is approximately equal to the product of the packet duration multiplied by the size of the jitter buffer. For video reception, keeping the jitter buffer size constant, we notice that as the number of the participants is increased, the processing delay increases by up to 38%. Furthermore, at each tested mode, decreasing the size of the buffer we observe that the estimated delays are reduced almost by the same percentage.

Regarding the performance of the BareServer, we demonstrate the per packet relaying delay in Table III. We observe that, as the number of the participants and the packet time are increasing, the system is stressed and its processing delay is increased up to 260%. Note, however, that these measurements are in microseconds.

TABLE III

Server measurements for audio (μsec)				
ptime	**VC 2**	**VC 3**	**VC 4**	**VC 5**
20	96.75	147.22	249.12	217.79
10	87.94	114.93	152.49	134.31
5	68.45	91.68	114.79	139.98

Aiming at achieving the lowest delay possible, we would naturally choose the lowest value for the `ptime` parameter as well as the jitter buffer size. However there is a trade off between the delay and the perceived audio quality which should be taken into account.

IV. Conclusion

The goal of the MusiNet project is to integrate ultra low delay audio coding, scalable video coding, low delay relaying and musician collaboration techniques into a unified prototype towards a full potential NMP system for the average user. In this work we have presented the progress accomplished towards this goal in terms of implementation of the primary units of the system. In future work we will present complementary delay measurements as well as objective and subjective evaluation results in real conditions.

Acknowledgements

This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS–MusiNet.

References

[1] J.-P. Cáceres and C. Chafe, "JackTrip: Under the hood of an engine for network audio," in *Proceedings of International Computer Music Conference*, 2009, p. 509–512.

[2] A. A. Sawchuk, E. Chew, R. Zimmermann, C. Papadopoulos, and C. Kyriakakis, "From remote media immersion to distributed immersive performance," in *ACM SIGMM Workshop on Experiential Telepresence (ETP)*, 2003, pp. 110–120.

[3] A. Carôt, A. Renaud, and V. B., "Network music performance (NMP) with Soundjack," in *NIME Conference*, 2006.

[4] C. Alexandraki et al, "Towards the Implementation of a Generic Platform for Networked Music Performance: The DIAMOUSES approach," in *ICMC*, 2008, pp. 251–258.

[5] D. Akoumianakis et al, "The MusiNet project: Towards unraveling the full potential of Networked Music Performance systems," in *The 5th International Conference on Information, Intelligence, Systems and Applications, (IISA)*, July 2014, pp. 1–6.

[6] C. Chafe et al, "Effect of time delay on ensemble accuracy," in *International Symposium on Musical Acoustics*, 2004.

[7] Creytiv software. [Online]. Available: http://creytiv.com

[8] J.-M. Valin, G. Maxwell, T. B. Terriberry, and K. Vos, "High-quality, low-delay music coding in the OPUS codec," in *Audio Engineering Society Convention 135*, Oct. 2013.

[9] A. Eleftheriadis, R. Civanlar, and O. Shapiro, "Multipoint videoconferencing with scalable video coding," *Journal of Zhejiang University SCIENCE A*, vol. 7, no. 5, pp. 696–705, April 2006.

[10] M. Westerlund and S. Wenger, "RTP topologies," RFC Editor, RFC 5117, April 2015.

[11] H. Schulzrinne et al, "RTP: A transport protocol for real-time applications," RFC Editor, RFC 3550, July 2003.

[12] FreeSwitch home page. [Online]. Available: https://freeswitch.org/

[13] G. Baltas and G. Xylomenos, "Evaluating the impact of network I/O on ultra-low delay packet switching," in *Proc. of the IEEE International Symposium on Computer and Communications (ISCC)*, 2015.

[14] L. Rizzo, "Netmap: a novel framework for fast packet I/O," in *USENIX Advanced Technology Conference*, 2012.

[15] G. Baltas and G. Xylomenos, "Ultra low delay switching for networked music performance," in *Proc. of the International Conference on Information, Intelligence, Systems and Applications (IISA)*, 2014.

[16] D. Akoumianakis, "Socio-materiality of online music ensembles: An analysis based on cultural artifacts & affordances," in *Science and Information Conference (SAI)*, Oct 2013, pp. 304–314.

[17] D. Akoumianakis et al, "Collaborative music making as "remediated" practice," in *18th International Conference on Digital Signal Processing (DSP)*, July 2013, pp. 1–8.

[18] M. Wiberg, "Methodology for materiality: interaction design research through a material lens," *Personal and Ubiquitous Computing*, vol. 18, no. 3, pp. 625–636, 2014.

[19] V. Alexiou and A. Eleftheriadis, "Real-time high-resolution delay estimation in audio communication using inaudible pilot signals," in *Submitted to the International Symposium on Communications, Control and Signal Processing (ISCCSP)*, 16 December 2014.

[20] ——, "Real time delay estimation in audio communications using inaudible pilot signals," in *Preparation for submission*, 2015.