Computer Music through the Cloud: Evaluating a Cloud Service for Collaborative Computer Music Applications

Antonio Deusany de Carvalho Junior, Marcelo Queiroz
Instituto de Matemática e Estatística
Universidade de São Paulo
dj,mqz@ime.usp.br

Georg Essl
Electrical Engineering and Computer Science
University of Michigan
gessl@umich.edu

ABSTRACT

Cloud computing offers a new level of very large scale distributed networking offering unprecedented level of networked participation and performance. This makes cloud services an attractive candidate for computer music concerts and applications. However, important network measures that are relevant for computer music performances, such as latency, have as yet not been explored in detail in this context. This paper presents results of an evaluation regarding the round-trip-time (RTT) for exchange messages through Pusher cloud service using mobile devices at different places on the American continent, exploring their use for very long distance collaborative performances. Average RTT varied from 230 to 578 milliseconds. Minimum round-trip time clocked in at 166ms in one of our tests between USA and Brazil. The details of our study can help develop collaborative network performances that can build strategies to incorporate expected latency patterns into either the back-end architecture or into aspects of the performance itself.

1. INTRODUCTION

New technologies can break the rules of old paradigms with possible solutions for classic problems. Cloud computing is an example of a tremendously successful solution for large scale problems that were not feasible before it. Cloud computing has gained confidence and popularity, and its use has become an easily accessible commodity. Although it is applicable for many computational problems, evaluation of the characteristics of cloud services for computer music applications is still lacking.

At first sight, the structure of the cloud is useful for computing scientific algorithms, but we can also make use of the fast data distribution between different locations. This approach can help many applications to behave as a responsive computer music environment for collaboration through the Internet. Another important point is that cloud computing creates an abstraction of the complex architecture that would be needed for an efficient communication between a large number of devices at the same time. Leveraging this abstraction makes the creation of very large scale networked performances substantially easier than before.

Musical collaboration often uses local and remote networks. Local networks have the advantage of guaranteeing low latency, making the technology relatively straight-forward to use in a performance setting, yet numerous long-distance Internet music performances have also been successfully conducted in the past. In most of those cases we notice that a specific network setup is necessary beforehand and the performers also have to manually configure network settings in order to suit their needs depending on many variables, such as the number of participants, the number of data streams and the exact nature of exchanged data. As opposed to that, balancing, scalability, and setup will be done behind the scenes on cloud computing solutions.

An important challenge of long distance networking comes from network latency in the communication between devices from different locations. A latency of 100ms and above implies some difficulty for network interaction even with experienced musicians [1], and if we have a sensitive ensemble performance, this threshold may be as low as 20ms [2]. One way to tackle this problem in local networks is using lightweight networking protocols such as UDP [3, 4], that has a low overhead and fast data diffusion. The potential for packet loss can often be of minimal impact to the ability to perform in practice.

In the case of long-distance networks there are no ready-made solutions, and performers may well have to grapple with the specific latency present in any given configuration. In order to facilitate this baseline understanding our work aims at evaluating the cloud computing latency and usability in the long-distance case.

We developed a mobile application using a cloud service for communication between mobile devices in different locations. We opted to exchange textual messages based on symbolic data that can be synthesized on local devices, avoiding audio transmission. One of our premises is that we can take advantage of mobile devices processing capabilities, that were already evaluated in previous works [5, 6]. This approach is also based on the heavy use of MIDI and OSC on computer music performances, and its support by most programming languages used on mobile music applications.

Although most cloud computing applications require a configuration of virtual machines or web servers, we are exploring push-based cloud services as an alternative. These are widely used to service large scale distributed push-notifications in mobile apps. In particular we are using the cloud service offered by Pusher [1] for our work. Pusher makes the design of large scale data distribution easy through accessible libraries, providing a convenient gateway for developing scalable networked data distribution solutions.

In the next sections of the paper we will introduce related work before a discussion on cloud services and the Pusher cloud service. The evaluation and results are going to be presented in Sections 5 and 6, respectively, and we will conclude with the discussion and analysis of these results.

2. BACKGROUND AND RELATED WORKS

Experiments with network music date back to the 1970's with *The League of Automatic Music Composers* [7]. The set up of this group of composers was based on desktop computers and a local network. Each member would be responsible for some musical feature during the performance, and the interaction would take place through the local network, and later, through phone lines. An important pioneering example of mobile wireless networked performance is Golan Levin's "Dialtones" [8]. This performance, based on audience participation with mobile devices, was achieved by the definition of a ringtone for each participant on a determined seat, and the use of an application to call specific devices at any time. These kinds of music performances were only possible due to the creative use of new technologies available at each period, the commercialization of the personal computers and wired networking in the first instance, and popularization of mobile devices and wireless networking in the second case.

We have had an increasing number of important works during the 21st century, and this fact may be associated to the accelerated technological evolution of this period. Every new product or feature presented to the market is used on musical performances and installations soon after receiving musician's attention. Some examples are touchscreen sensors [9], cameras [10], mobile sensors [11, 5, 12], smartphones with all their features [13], laptops orchestras [14], and browsers with HTML5 and new Javascript libraries [15, 16]).

New communication technologies also encourage musical performances. Even advances in network protocol technologies and transmission media can alter and improve the possibility space for music performances. The protocols used on network music had great progress thanks to advances made in network protocols starting with TCP, and then UDP, SCTP, RTP, and RTSP [17]. Additionally, advances in communication regarding the quality of data transmission are impressive. Nowadays, one of the most popular transmission medium is the optical fiber used on gigabit Internet interconnection that reaches hundreds of billions of bits per second in some parts of the network.

The use of local network novelties can be seen on "Worldscape laptop orchestra" [3]. Here the authors decided to use the recent wireless standard 802.11n, due to its high frequency band and datarate that improved laptop communication if compared to old standards. Freeman's "massMobile" [18], Allison's "Nexus" [19], and Hindle's "SWARMED" [20] present different ways of using web servers and web technology for musical performances. In these works, web services fill the gap between user interaction and local installations or performances. An interesting aspect of "massMobile" [18] is the use of 3G and 4G technologies to ease audience participation.

Cloud computing suggests itself as the next logical step in network capability, ready to be used for music applications and performances. The "CloudOrch" [21] as proposed by Hindle is one of the first attempts to use the advantages of cloud computing in musical ways. The idea was to deploy virtual machines for client and server users, create websockets for intercommunication, and stream audio from cloud instruments to both desktop computers and mobile devices using web browsers. The author dubbed it "a sound card in the cloud" and presented latency results from 100 to 200ms between the Cybera cloud and the University of Alberta, Canada, using the HTTP protocol. Although similar, our work took full advantage of the cloud computing resources and focused on cloud services that allow the use of virtual machines with a high level of abstraction while retaining comparable performance.

3. CLOUD SERVICES

An important core aspect of cloud computing relies on an abstraction of distributed servers in order to simulate a centralized network with resource replication and balancing. Early on cloud computing was mostly used for scientific calculations, grid computing, or large scale computing. All these application areas required substantial computational power and hence benefited from many cores.

The cloud computing evolution pointed out the possibility of cloud services as a solution for simple tasks. Cloud services are services that are deployed on a cloud computing structure to take advantage of its computation and distribution qualities. A common use is the data replication service at websites and mobile applications, so even if we have increasing access at some moment, the cloud service can instantiate another machine or machines to provide minimum latency and avoid processing overhead on the servers.

In our work we decided to evaluate the push notification service of Pusher. The main intention here is to provide an easy way to exchange messages between mobile devices around the world with minimum codification effort.

4. PUSHER CLOUD SERVICE

Numerous mobile applications have a focus on fast message delivery to a high number of devices in many parts of the

[1] `https://pusher.com/`

world. A good example of this service is an email application that sends us a notification whenever we receive a new message without requiring us to actively request new information. This approach is called push notifications, and one implementation of it is Pusher. This cloud service uses cloud computing solutions in order to offer a service that delivers messages through web sockets and HTTP streaming.

One of the advantageous features supported by Pusher API is its support of HTTP Keep-Alive feature. Once connected to Pusher cloud service, it is possible to send and receive messages to/from the cluster without starting a new connection, saving the overhead of restarting new TCP connections. The Pusher service offers the possibility of creating real-time applications considering all of its resources.

Although Pusher has many paid plans with more resources available, we decided to evaluate the free plan and verify if it can be used successfully in specific use cases. The free plan has a limit of 100,000 messages per day and it counts the API requests and messages delivered to each client. We can have up to 20 clients connected at the same time and all clients will send and receive messages only from the US-East cluster server that is situated in Northern Virginia.

The service also has some restrictions for all plans, free and paid alike. Clients are allowed to send no more that 10 messages per second and will be disconnected from the service when they exceed this limit. This limitation is due to the overhead on distributing messages among a thousand users. Every message has a size limit of 10 kilobytes, but it is possible to request an upgrade in order to send larger messages.

The requirement to exchange messages between users is to create a channel and have users connected to the same channel. The API supports public, private, and presence channels. Public channels are used only to send messages from the server to the users connected, like a feed. Private channels need a prefix “private-”, require server authentication, and offer the option to accept messages from its users. The presence channel is similar to the private channel, and includes the feature of requesting information about connected users. The channels may have different events that can be bound by clients, e.g. a client can bind the event “client-event” and receive notifications when a new event with the same name is sent to the channel. On private and presence channels, client events must have the “client-” prefix. An example code used to send and receive messages through the Pusher cloud service using a private channel is presented on Listing 1.

In this piece of code we have omitted some information that depends on the application. The first is the “AUTH_PAGE”, that needs to be configured to give a unique authentication code to every socket connection. The “PUSHER_API_KEY” is the key received when creating an application on Pusher. After creating the channel, we need to bind an event with an event listener. The event listener requires the code that is expected to run every time the event occurs. The “MESSAGE” is a string with any values defined to be sent, e.g. numbers need to be converted directly to string or through Base64 binary-to-text encoding schema.

Pusher can send any type of data through the cloud respecting plan limits. For instance, one might share codes from computer music languages like CSound[2], ChucK[3], and SuperCollider[4]. Cloud services can also share symbolic data or control signals between any mobile applications, allowing one to make use of any computer music synthesis engines available in tandem with Cloud services. On the next section we are going to present the evaluation proposed to verify the utility of push notifications on computer music in a large scale context.

```
Pusher pusher;
PrivateChannel channel;

HttpAuthorizer authorizer =
        new HttpAuthorizer(AUTH_PAGE);
PusherOptions options = new PusherOptions();
options.setAuthorizer(authorizer);
pusher = new Pusher(PUSHER_API_KEY, options);
pusher.connect();

String channelName = "private-channel";
channel = pusher.subscribePrivate(channelName);
String eventName = "client-event";
channel.bind(eventName,
        new PrivateChannelEventListener() {...});
JSONObject jsonObject = new JSONObject();
String message = MESSAGE;
jsonObject.put("message", message);
channel.trigger(eventName, jsonObject.toString());
```

Listing 1: Example of Java code from Pusher API

5. EVALUATION

Our core metric for evaluation of push-notification-based services in large scale computer music application is round-trip time (RTT). The *Pusher.com* cloud service has been chosen due to its popularity and hard limits on the free plan. For comparison, another available option would be the PubNub[5] service, which at this time offers a free plan that has a limit of one million messages per month and messages up to 2KB.

Some pilot tests performed before the full evaluation presented an intermittent loss of connection when exactly 10 messages were sent per second, which represent the nominal maximum allowed. To avoid this issue a 150ms delay between messages was used, and we established 10 cycles of 100 messages per test with a delay of 500ms between each cycle during the tests. Each test represented a performance of approximately 3 minutes with these definitions.

Six tests were defined based on the number of floats sent as arguments inside the messages. We selected to evaluate messages with 1, 50, 100, 150, 200, and 250 floats. In this case, the evaluation simulated messages in a range from 7 to 1750 floats per second. The messages had five divisions: a unique device id; message number with cycle number on the range of thousands; the total number of messages; a random integer as a key for message identification; and a block of floats.

[2] CSound: `http://www.csounds.com/`
[3] ChucK: `http://chuck.cs.princeton.edu/`
[4] SuperCollider: `http://supercollider.github.io/`
[5] PubNub: `http://www.pubnub.com/`

An example of a message used during the tests is presented below:

```
23.0.1.A87422113 1001 1000 76 [0.28506452]
```

We created an Android application based on the loop-back concept, so one device would send messages, the "*sender*", while the other device, the "*loopback*", would need to answer the message as soon as possible, simulating a loop-back circuit. The first device registered all messages sent and received on a report file with millisecond time-stamp precision. The method used to register the events on the report was *SystemClock.elapsedRealtime()*. We also run AsyncTask invoking *executeOnExecutor()* with *THREAD_POOL_EXECUTOR* in order to have parallel execution on Android. An activity diagram is presented on Figure 1. The *loopback* device also registered sent messages on report with the same precision.

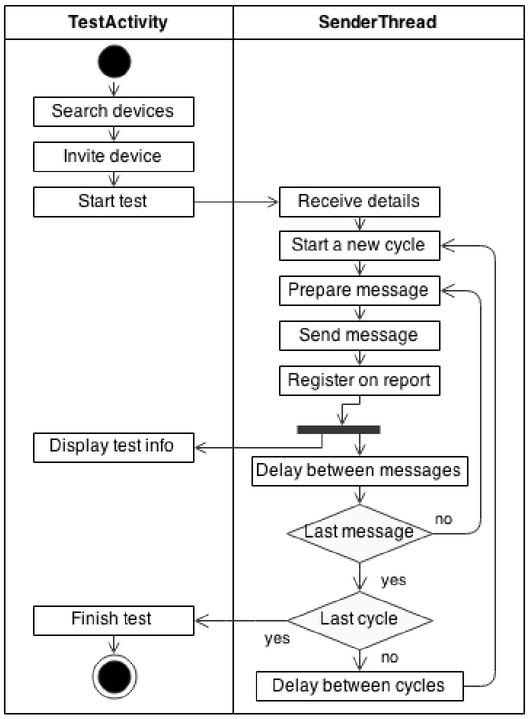


Figure 1: Activity diagram of the test from the sender point of view.

Two mobile devices were used for this evaluation: LG D685 G Pro Dual Lite (D685) and Sony Xperia Z3 Compact (Z3) [6]. The tests were performed between three different universities on the American continent. The mobile device D685 was selected to be the *loopback* device at the Federal University of Paraíba (João Pessoa, PB, Northeastern Brazil). On the other hand, the Z3 was defined as *sender* from two different universities. The first evaluation was performed from the University of São Paulo (São Paulo, SP, Southeastern Brazil), and the second from the University of Michigan (Ann Arbor, MI, USA). We decided to use the Z3 as the sender due to its quad core processor that could reduce the problems with system latency.

The routes between the universities and the cluster used by the cloud service are presented on Figure 2. It is important to notice that every message needs to be sent to the cluster before being pushed to its final destination. This means that every message exchanged between São Paulo and João Pessoa passes twice through São Paulo before going back to São Paulo.

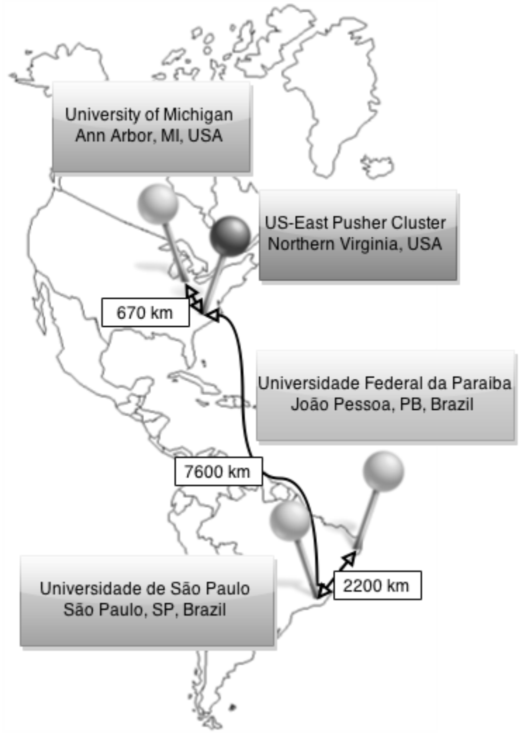


Figure 2: Routes between the universities with linear distance in kilometers

6. RESULTS

The geographic configuration imposes physical restrictions on the lower bounds of networked performance. We made use of the broadband Internet connection between the universities, whose actual medium of interconnection is optical fiber, and the speed of light in this medium is about 200,000 km/s. If we consider the distances on the map at Figure 2, we have the light RTT as 174ms on the first evaluation, between São Paulo and João Pessoa (SAO-JPA) [7], and 104ms on the second evaluation, between Ann Arbor and João Pessoa (ARB-JPA) [8].

The RTT for each message sent is presented in the charts of Figure 3. Some values overshoot the chart limit to maintain the scale of all charts. The results that reach the ground

[6] Comparison between the devices: `http://www.gsmarena.com/compare.php3?idPhone1=5736&idPhone2=6538`

[7] The route between São Paulo and João Pessoa is 34,800km.

[8] The route between Ann Arbor and João Pessoa is 20,940km.

represent a loss of message or connection during evaluation. Table 1 and Table 2 show a summary with the main results extracted from these measurements.

Floats	1	50	100	150	200	250
Lost msgs	14	26	25	3	21	38
Msg size	41	614	1190	1782	2355	2950
Minimum	342	332	332	329	332	352
Maximum	2430	3916	4371	1595	3014	1700
Average	515	578	563	486	536	543
Std. dev.	224	366	394	181	305	168

Table 1: SPA-JPA tests: Results from RTT evaluation using cloud services between São Paulo and João Pessoa. RTT is in milliseconds and average message size is presented in bytes consisting of 1 byte per character.

Floats	1	50	100	150	200	250
Lost msgs	3	0	0	17	5	0
Msg size	43	613	1189	1784	2378	2935
Minimum	166	172	172	182	199	190
Maximum	1953	1052	898	3100	1869	951
Average	243	230	273	316	348	329
Std. dev.	138	83	103	317	143	101

Table 2: ARB-JPA tests: Results from RTT evaluation using cloud services between Ann Arbor and João Pessoa. RTT is in milliseconds and average message size is presented in bytes consisting of 1 byte per character.

We can see in the charts that the RTT is better on ARB-JPA evaluation. Another point from ARB-JPA evaluation is that the more floats we have on a message, the higher RTT we find. The SAO-JPA evaluation presented some instability regarding the average RTT and there was no discernible pattern linking the number of floats to the average RTT.

Although we have had some high RTT values during the tests, the concentration of high RTT values appear to be more frequent in clustered sequential messages than when messages are isolated (meaning musical algorithms based on sporadic communication would suffer less). Furthermore, it was relatively common for the service to lose at least one message after a high RTT value.

The results summarized in the Table 1 present a minimum RTT of 329ms with 150 floats and average RTT values between 486-578ms for all message sizes. On the other hand, we have a minimum RTT of 166ms with 1 float on Table 2, and average RTT values between 230-348ms. We sent 1000 messages on each test and lost no more than 4% of the messages in the worst case, and we lost more messages in the SAO-JPA evaluation than we lost in ARB-JPA. Moreover, ARB-JPA evaluation had tests with zero message loss.

7. DISCUSSION

A common scenario for network music performance expects minimum latency when synchronization is at stake. Participants' locations are an important factor in this case. When participants are in the same place, technologies for local networks might offer the best solutions, whereas if we want participants from different places or continents, the Internet would be the obvious choice. Despite the fact that those are easy choices, the merging of solutions for handling simultaneously both scenarios may not be easy to achieve. Moreover, depending on the technology used, some participants will have difficulties to engage in the performance due to technical restrictions. In our work we decided to evaluate a cloud computing solution for handling both situations at once.

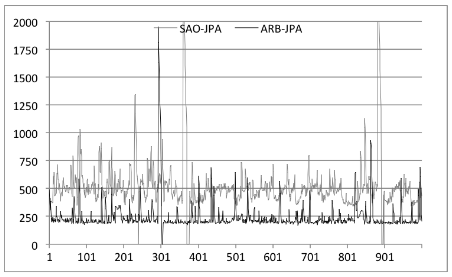
We evaluated the communication between mobile devices from different locations using the Pusher cloud service and WiFi connection. We expected some differences due to the length of the route connecting the cities, the time to process the messages in all endpoints, and delays caused by any network congestion, since those personal devices were not using a dedicated connection. Notwithstanding, we got good results that are going to be discussed on this section.

First of all, the location of the clusters used by the cloud servers might be known or better selected beforehand. During our tests, we found out that we were taking redundant routes as a consequence of the specific cluster for the free plan used on Pusher. The paid plan allow the definition of the cluster, and we recommend a previous verification of the possibilities before signing up to any cloud service.

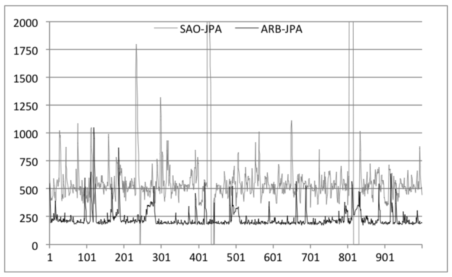
The advantages of the cloud service offered by Pusher are many: we can exchange messages through a robust cloud computing structure with just a few lines of code; we do not need to care about any other cloud computing configuration (e.g. virtual machines, web server); they have libraries for many languages used on desktop computers and mobile devices, and the same application can interact using both platforms at the same time; and we can also send any kind of data using any string conversion or encoding, for instance, any Base64 encoding schemes.

Although the free plan fitted our evaluations, it has some disadvantages for music performances. The users are restricted to one cluster localization and they need to upgrade to the paid plans, in case a performance requires lots of users and messages. We have reached daily limits with our evaluations when we tried to repeat some tests, and we had to wait 24h to restart everything. Another disadvantage is that apparently we can only use one cluster location per application, but it is possible to transit between clusters under some special conditions (e.g. on presence channels we can't have clients connected during transitioning).

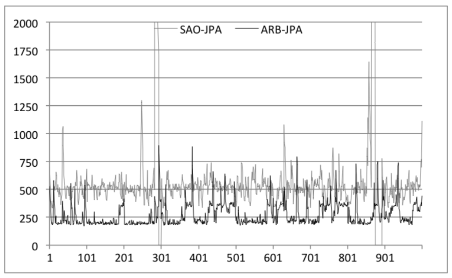
Despite of that, computer music performances can experience new possibilities for collaborative works using this cloud service and taking everything into account. Audience participation is also a good target for users of cloud services due to its easy integration with WebAudio applications that can run on browsers of both mobile devices and computers. In this



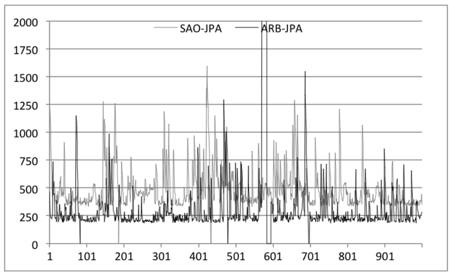
(a) One float



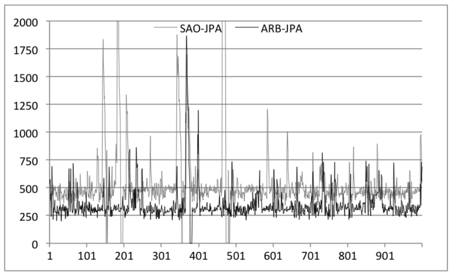
(b) 50 floats



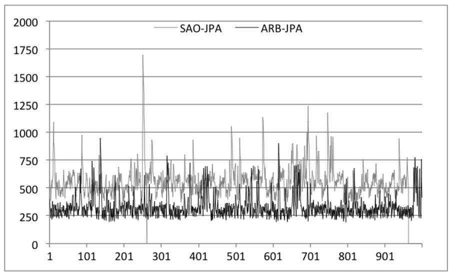
(c) 100 floats



(d) 150 floats



(e) 200 floats



(f) 250 floats

Figure 3: RTT for each message.

case, the audience will only need an Internet connection to access the application through a compatible browser.

The use of symbolic data reduce the costs of data transmission for participants, and might be better accepted than an audio application that can result in an expensive bill in the end of the month considering the prices of current 4G plans (particularly in Brazil at this point). Moreover, the audio synthesis is becoming less problematic while most of actual mobile devices are multi-core enabled and support any programming language with libraries to mitigate problems in audio processing.

Considering half of RTT as latency, we got a minimum of 83ms and averages between 115 and 289ms in our tests, even with large messages, and we expect that better results can be achieved in places near the clusters. The advantages of using cloud computing in musical performances are also related to reliability in message distribution. We lost just a few messages, less than 4% on each test, and we had many tests without losing any messages.

We also took advantage of HTTP Keep-Alive feature to reduce the overhead of TCP connections. We can infer that cloud computing is a reliable alternative when compared to UDP, which is one of the most used solutions for collaborative musical performances but cannot be easily adapted to multiple mobile devices hidden under many widespread local networks.

Even though these results may suggest that the free plan of this particular cloud service is not suited for a highly synchronized performance across the continent, or a performance highly sensitive to data losses, in other contexts latency and losses may be absorbed in musical settings with user interaction without being perceived as flaws or problems. Some compositions or performances could rely on sounds with long attacks or with very smooth variations so as to minimize (or even overcome) latency issues during music performance. Another interesting approach to handle latency is to work with two time-lines: one for preparing and composing and another for sharing and performing, so that participants can spend time creating short music sequences on their devices before sending, without worrying about getting it wrong somehow.

8. CONCLUSION

The delay between messages is an important setting on the Pusher cloud service. Message loss may have occurred in our evaluations due to overhead on the network buffer. In the event that at least two packets are grouped by a network buffer at any point before being sent to Pusher, we will probably have more than 10 messages per second even if we decide to use 150ms delay between packets. Pusher will disconnect the user at this moment, and will try to reconnect the socket as soon as possible. Although some messages might be lost in this situation, the application developer can bind the disconnection event and stop sending messages until the device gets connected again.

Paid plans offer different clusters that can be selected depending on routes between performers or application users. Although we have lots of clusters available around the world on Pusher cloud service, it may be necessary to try another cloud service if the localization of these clusters are not suitable. At the time of writing this paper, Pusher still does not have clusters in Brazil, and we would suggest other cloud services like PubNub in the case that all participants are from this country.

We have showed that the computer music community may use cloud services as a new way of intercommunication in musical applications, a way that facilitates implementation and improves scalabity of previous musical ideas that were conceived for small groups of participants. The delay and reliability of the service has proven to be suitable for many applications and opportunities, and we may expect improvements in a near future. Push-notifications help users to send and receive data from any part of the world and control the final sound from their own mobile devices, using other users' inputs to influence their music creation in a collaborative way. We expect that many musical works aimed at local networks may now be extended to cloud-based services.

Acknowledgments

This research would not be possible without the support of CAPES [9], RNP [10], Computer Music Research Group [11] and NuSom Research Centre on Sonology [12] at USP, the Digital Video Applications Lab (LAViD) [13] at UFPB, the mobile music group at the University of Michigan, and Internet2 [14].

9. REFERENCES

[1] C. Bartlette, D. Headlam, M. Bocko, and G. Velikic, "Effect of Network Latency on Interactive Musical Performance," *Music Perception: An Interdisciplinary Journal*, vol. 24, no. 1, pp. 49–62, 2006.

[2] C. Chafe and M. Gurevich, "Network Time Delay and Ensemble Accuracy: Effects of Latency, Asymmetry," in *Audio Engineering Society Convention 117*, Oct 2004. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=12865

[3] A. Harker, A. Atmadjaja, J. Bagust, and A. Field, "Worldscape laptop orchestra: Creating live, interactive digital music for an ensemble of fifty performers," in *International Computer Music Conference*, 2008.

[4] J.-P. Cceres and C. Chafe, "JackTrip: Under the Hood of an Engine for Network Audio," *Journal of New Music Research*, vol. 39, pp. 183–187, 2010.

[5] J. W. Kim and G. Essl, "Concepts and Practical Considerations of Platform-Independent Design of Mobile Music Environments," in *International Computer Music Conference*, 2011, pp. 726–729.

[6] A. J. Bianchi and M. Queiroz, "On the performance of real-time DSP on Android devices," in *Sound and Music Computing Conference*, 2012, pp. 113–120.

[7] G. Weinberg, "The aesthetics, history, and future challenges of interconnected music networks," in *International Computer Music Conference*, 2002, pp. 349–356.

[8] G. Levin, "Dialtones - A telesymphony, September 2001," Brucknerhaus Auditorium, Linz, Austria, 2001. [Online]. Available: http://www.flong.com/projects/telesymphony

[9] G. Geiger, "PDa: Real time signal processing and sound generation on handheld devices," in *International Computer Music Conference*, 2003.

[10] M. Rohs, G. Essl, and M. Roth, "CaMus: live music performance using camera phones and visual grid tracking," in *New Interfaces for Musical Expression*, 2006, pp. 31–36.

[11] G. Essl and M. Rohs, "ShaMus: A sensor-based integrated mobile phone instrument," in *International Computer Music Conference*, 2007, pp. 200–203.

[12] A. D. de Carvalho Junior, "Sensors2PD: Mobile sensors and WiFi information as input for Pure Data," in *Joint Conference: 40th International Computer Music Conference and 11th Sound and Music Computing Conference*, 2014.

[13] G. Wang, G. Essl, and H. Penttinen, "Do mobile phones dream of electric orchestras," in *International Computer Music Conference*, 2008.

[14] I. I. Bukvic, "A Behind-the-Scenes Peek at World's First Linux-Based Laptop Orchestra The Design of L2Ork Infrastructure and Lessons Learned," in *Linux Audio Conference*, 2012.

[15] H. Choi and J. Berger, "Waax: Web audio api extension," in *New Interfaces for Musical Expression*, 2013, pp. 499–502.

[16] C. Roberts, G. Wakefield, and M. Wright, "The Web Browser as Synthesizer and Interface," in *New Interfaces for Musical Expression*, 2013, pp. 313–318.

[17] F. L. Schiavoni, M. Queiroz, and M. Wanderley, "Alternatives in network transport protocols for audio streaming applications," in *International Computer Music Conference*, 2013, pp. 193–200.

[18] N. Weitzner, J. Freeman, S. Garrett, and Y.-L. Chen, "massMobile - an Audience Participation Framework," in *New Interfaces for Musical Expression*, 2012.

[19] J. Allison, Y. Oh, and B. Taylor, "NEXUS: Collaborative Performance for the Masses, Handling Instrument Interface Distribution through the Web," in *New Interfaces for Musical Expression*, 2013, pp. 1–6.

[9] CAPES: `www.capes.gov.br`
[10] RNP: `www.rnp.br`
[11] Computer Music Research Group: `compmus.ime.usp.br`
[12] NuSom Research Centre on Sonology: `www.eca.usp.br/nusom`
[13] LAViD: `www.lavid.ufpb.br`
[14] Internet2: `www.internet2.edu`

[20] A. Hindle, “SWARMED: Captive Portals, Mobile Devices, and Audience Participation in Multi-User Music Performance,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2013, pp. 174–179.

[21] ——, “CloudOrch: A Portable SoundCard in the Cloud,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, B. Caramiaux, K. Tahiroglu, R. Fiebrink, and A. Tanaka, Eds., London, United Kingdom, Jun. 2014, pp. 277–280.