Understanding Cloud Service in the Audience Participation Music Performance of *Crowd in C[loud]*

Antonio Deusany de Carvalho Junior
Universidade de São Paulo
Rua do Matão, 1010,
CEP 05508-090, São Paulo,
SP, Brazil
dj@ime.usp.br

Sang Won Lee
Computer Science and Engineering
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
snaglee@umich.edu

Georg Essl
Electrical Engineering & Computer Science and Music
University of Michigan
2260 Hayward Ave
Ann Arbor, MI 48109-2121
gessl@umich.edu

ABSTRACT

Cloud services allow musicians and developers to build audience participation software with minimal network configuration for audience and no need for server-side development. In this paper we discuss how a cloud service supported the audience participation music performance, *Crowd in C[loud]* [10], which enables audience participation on a large scale using the audience audience's smartphones. We present the detail of the cloud service technology and an analysis of the network transaction data regarding the performance. This helps us to understand the nature of cloud-based audience participation pieces based on the characteristics of a performance reality and provides cues about the technology's scalability.

Author Keywords

Audience participation, Cloud Service, Networked Music

ACM Classification

H.5.5 [Information Interfaces and Presentation] Sound and Music Computing, H..5.3 [Group and Organization Interfaces] Web-based interaction, J.5 [ARTS AND HUMANITIES] Performing arts (e.g., dance, music)

1. INTRODUCTION

It has been a long-standing endeavor to create musical performances in which the audience can easily participate. In particular, mobile smartphones have the highly desirable characteristic of already being in the possession of the audience members while offering networking and rich sensor capabilities. Golan Levin's Dialtones[13, 6] used ring-tones and wireless network dial-up to enable a concert-hall filled audience involvement piece with mobile phones. Since then much effort has been devoted to build mobile-based infrastructure to support mobile-device based audience participation such as echobo[12], massMobile[17] or Swarmed[7]. This paper aims to describe the technical realities of a recent audience participation piece called *Crowd in C[loud]* [10]. Studying cloud services in real performance settings will enable us to better understand them as they evolve. In part this paper can also be understood as advocating for the simplicity of this approach to audience participation and that some services that can be easily integrated into musical performances without requiring a deep technical background in cloud computing or virtual machines from the artist.

Our work draws upon two long-standing research and performance traditions: (1) distributed musical performances, a field of network music where people can collaborate in music making remotely, and (2) mobile music, that considers musical interaction through mobile devices. We present a three-way connection between a human, a musical instrument and the cloud server, in which the interface is built on a web page for collaborative music and the instrument communicates with the cloud server to enable social interaction among the audience. Realizing this network setup using the cloud service will require no network configuration for audience (other than visiting a web page) and no server-side programming (other than uploading the web pages) for the musicians. In addition, leveraging recent advances in web audio, we were able to distribute a complex networked musical instrument by sharing a shortened link.

In this paper we will focus on low level aspects of the technological reality of the piece, both in terms of architecture and in terms of the observed network characteristics in a public performance of the piece. For a more detailed description on the piece, please read [10] and watch a videotaped performance at `https://youtu.be/8nnrKJ4Ap0c`.

NIME'16, July 11-15, 2016, Griffith University, Brisbane, Australia.
.

2. COLLABORATIVE NETWORK MUSIC

Network music is a computer music tradition developed to support collaboration among computer musicians [2]. The use of local networking is a rather standard piece of networked ensemble today. Interaction through a local network requires a setup effort but adapts to the performer's collaboration scheme and offers reliability as musicians know the whole network structure while defining desired settings.

One of the technical aspects of setting up network performances is addressing the hosts that will be participating. If a remote host can be correctly targeted to local network, this can be extended more globally. The conventional solution to this problem are fixed IPs. However, the necessity of a NAT for IPv4, and before the wider adoption of IPv6 around the world, the need for a public IP on the Internet limits this approach, as it is cumbersome to acquire. Some solutions mitigate this problem by choosing a single central server with a public IP. In this case, the other users will need to create a socket with the server while this server distributes data to all users. Collaborative live coding interconnected through TCP/IP (fixed IP) is a good example of extending local network to Internet and we already have some performances like that using Republic[1], an extension to SuperCollider, for example. However, if this solution is applied to an audience participation context, the musician will need to guide their audience to follow configuration steps. This may include downloading an app, joining a specific network and typing the IP address, which may prove difficult for casual users. Also, it might be challenging for the musician to control a large-scale crowd.

The use of online services like Twitter can save some network setup, as can be seen in Dahl's TweetDreams [3]. Roberts' Gibber [15] allows for code sharing through existing services/web browsers. Allison's NEXUS [1] leverages web services which provide interfaces for user interaction, including easy and distributed setups. Cloud computing emerged as an alternative to facilitate easier network music configuration. Hindle's CloudOrch [8] and cloud orchestra [9] are early examples that use Cloud solutions to leverage user interaction through powerful network infrastructure with less effort for the network setup. The system architecture that supported the piece *Crowd in C[loud]*, the work described in this paper, also used the Cloud, but focused on Cloud Services, as we discuss in the next section.

3. CLOUD SERVICES

Cloud services can achieve our goal of enabling audience participation in musical performances using mobile devices and provide an immersive/interactive musical environment with fast and easy set up. The "always on" paradigm and realtime interaction is now a technological reality in network settings. Much of it has been enabled by cloud computing technologies to distribute data faster through a reliable and distributed network infrastructure. As part of unlocking cloud services for network music performance we have evaluated the efficiency of a cloud service in acomputer music context and obtained good results of 83ms latency between devices in North and South America through the Pusher cloud service[2] [4]. Although this work used an Android app during the evaluations, the cloud service provides an API (application programming interface) and SDK (software development kits) for web applications that can be loaded on browsers without any system configuration or app installation. In order to facilitate remote collaboration for live coding, we created an application using the Pusher cloud service. The application developed was named SuperCopair, acts as a package for Atom.io, and explores pair programming and distributed music performance using SuperCollider [5]. Users can share SuperCollider files, edit the files together online in a collaborative session, and can also run codes locally, remotely and globally regardless of the number of connected programmers, exchanging data with the Pusher cloud service. As the communication is administrated by the cloud servers, the users only need to be connected online to have a collaborative session.

Before cloud service, the developers would have to create, setup, and start a server locally (or remotely) and develop a custom software in order to manage the user interactions and the message exchanges between connected devices. Cloud services make network setup abstract to users and distribute the data using computers clusters available in distributed data center over the globe. The available APIs offered by these services contain the functions necessary for interconnection between devices, including methods to connect, disconnect, and send messages, and also a callback to listen for received messages.

[1]`https://github.com/supercollider-quarks/Republic`
[2]Pusher Cloud Service: `http://www.pusher.com/`

Upon our evaluations on a number of different cloud services, we chose PubNub cloud service[3], which had better specification, for this performance. The advantages of PubNub, compared to Pusher, include larger message size and a greater number of messages allowed. Both services have plans that preserve the size of the messages but differ in the allowable quantity of serviced messages. While Pusher limits the messages to 10KB, PubNub accepts messages of 32KB. Another disadvantage of Pusher, especially in the context of audience participation music, is that it will discard messages if the device exceeds the rate of 10 messages per second and plans limit the number of messages sent per day. PubNub limits the quantity of messages sent per month but the limits are never throttled even on free plans for our purpose, so we can use the full service capacity during one single performance, assuming no other performance within that month. We also opted to use a paid plan for a month to guarantee that we would have technological support during the performance and a dedicated key for this application only, as opposed to using a free demo key. More details regarding PubNub Cloud Service can be found at their website: `http://www.pubnub.com/`. In the next section we will describe the piece and application created for *Crowd in C[loud]*, the performance evaluated in this paper.

4. CROWD IN C[LOUD] : ONLINE DATING THROUGH MUSIC

Crowd in C[loud] is a networked music piece composed and developed for audience participation at a music concert [10]. It draws on the idea from the piece *In C* by Terry Riley, wherein musicians (with various instruments) were guided to play pre-composed melodic fragments in C Chord [14]. In *Crowd in C[loud]*, each participant uses web browsers (typically on their smartphones) that support Web Audio API and are instructed to play a short snippet (or tune) composed by herself and by other audience members for a random amount of time. The aggregated result of each individual playing a short tune creates a heterophonic texture of chance, largely in C chord. For more detailed motivation regarding the aesthetic of the piece, see [10].

Ease of use in designing a musical interface for audience participation is one of the most significant qualities in audience participation [12]. This is especially essential to motivate people to participate in the piece with clarity and musicality. *Crowd in C[loud]* incorporates two design decisions to achieve this accessibility. First, the interaction design in the instrument is loop-based where a participant needs to place musical notes on screen and the pattern of five musical notes will create a tune that is looped indefinitely based on where the notes are. This means the user does not need to make a playing gesture constantly in order to generate tones. This nature of modifying a musical loop ensures that the instrument will generate sound so that the musician need not worry about being too sparse or silent due to low participation.

Second, the piece uses the metaphor of online dating for browsing the composed tunes of others. In the beginning of the performance, once a participant finishes the composition, he or she can browse, and play, what other audience members have composed. Browsing tunes composed by others mimics an online-dating website (such as Tinder [4]) where a user creates a personal profile and then browses other member profiles that include pictures and written descriptions about themselves. The networked instrument creates a temporary social network that lasts until the end of

[3]PubNub Cloud Service: `http://www.pubnub.com`
[4]`www.gotinder.com`

the performance where each tune is a musical profile of a participant. In addition, the collection of each tune composed by individuals serves as musical phrases found in Riley's In C.

The metaphor of creating and browsing online profiles creates a set of states a participant can be in during the performance. The musical instrument for a user can be in one of five different states: NAME, EDIT, WAIT, CHECK, and MINGLE. Each state is used to design different interfaces and different modes of social interaction: 1) NAME is the initial state where a user determines the screen-name for participation. 2) EDIT is the state in which a user can compose and modify the looped tune to create a profile. 3) WAIT is a transient state that involves waiting for data response of tunes (typically composed by other audience members) 4) CHECK is the state where one can browse (and play) someone else's tune and 5) lastly, MINGLE is the state where a participant can play two tunes at the same time, which is a metaphor of conversation, off-line meeting, or being a match. The times in which a participant makes a transition from one state to another are the points when a participant's smartphones request data from the cloud service.

In *Crowd in C[loud]*, the only sounds that constitute the piece are coming from the audience seats, from speakers of the audience's smartphones. There is a performer on stage who runs a performer's interface, which serve as a scoreboard-like visualization in which stats of each profile is displayed (e.g. the number of likes). In fact the computer that the performer runs acts as a server where all the audience's tunes are stored and stats are calculated. Whenever a participant modifies a pattern or browses audience member's tune, the audience's interface request data from the performer and waits for its response, which would come via the cloud server from the performer's web page loaded at the laptop on stage. Additionally, the performer can orchestrate the crowd by live coding in Javascript. The on-the-fly code is distributed to all connected devices and is used to change the property of the instrument that audience play, as seen in [11]. For example, a performer can change chord and scale of the instrument so aggregated outcome of tunes can make chord progression over time. Therefore all devices are connected and will transmit data to the performer's laptop. In the following sections, utilization of cloud service for data transmissions will be discussed.

5. PERFORMANCE STRUCTURE FOR AUDIENCE INTERACTION

The performance structure was inspired by other networked pieces that presented a centralized network with a server for receiving and sending information [16]. However, our choice of using the cloud service set us free from the burden of developing a custom server application, which is replaced with a single web page written in HTML and Javascript.This will be a useful setup for artists who want to write a network music piece who do not have a networking background but can write an interactive web page. As its name suggests, PubNub cloud service permits easy interconnection between users through channels using publish-subscribe (or pub-sub paradigm). An application (or device) can subscribe to a channel and receive every notification that is published to the channel. For example to broadcast a message to all devices, a device can publish a message to a channel that every device is subscribed to. This provides a convenient abstraction that is robust against changes in network or end-user device and allows dynamic reconfiguration of participation. The push(or publish) notifications paradigm also makes performances more robust against technical disruptions such as disconnection and delays.

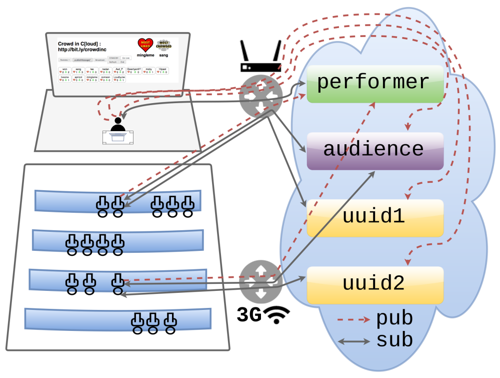

Figure 1: Performance structure. The left side depicts the performance space. On the right side, a rectangle represent a channel, solid lines show publishing and dotted lines show subscription.

Price/month	Free	$15	$49	$125	$399
Daily devices(max)	20	100	3k	10k	25k
# of msg per month	1 million	Millions(underspecified)			
Additional messages	-	Up to $5/M			

Table 1: PubNub pricing plan(April 2015)

The left side of Figure 1 presents the performance setup at the concert hall. There were two kinds of web pages running during the performance, one for the performer and another for the audience. The performer sat center-stage and his application (a web page) was project onto a screen for the participants seated in the audience. They were instructed to visit their application that has the musical instrument on. The performance hall had wireless network available from the university and had a good reception of mobile network connectivity such as 3G.

A representation of the cloud service with the channels we used is on the right side of the Figure 1. In our performance we designed three types of channels following the PubNub API for web application: a performer channel, an audience channel and individual channels equaling the participant number. The performer application is subscribed to the performer channel to receive messages sent from audience member's devices. Once a participant visits, it requests a performer's application to create an individual channel by publishing a message to the performer channel. The individual channel is then created to respond to an audience application's request, typically for the audience to retrieve a tune composed by others. When the page is loaded, every device is given a universal unique identifier (UUID) from the cloud service API, which is used to create an individual channel. Lastly, all the audience members are subscribed to the audience channel. This channel is used for crowd control: to send textual instructions, to live-code the mobile instrument, to start/end the performance and to troubleshoot the performance when it goes wrong (silence, refreshing the page).

The service plans available from PubNub at the time of the performance were divided into free and paid plans, and the price is described at Table 1. The number of messages was not an issue for the performance context (even with the free plan), because we would use the application only during rehearsals and on the performance day. However, the free plan limits the maximum number of daily devices to 20. As we could not predetermine the number of turnout for the performance, we could limit the number by the capacity of the concert hall (450), which led us to choose the third plan option in the assumption that there may be more than 100 devices.

PubNub cloud service provides an extensive API and SDKs, and the service can be easily configured with a few functions. Initially, the application needs to get an UUID and initialize the PubNub object including information regarding the account: the publish key and the subscribe key. After that, the device can publish messages to any channel as long as the channel name is known (e.g. "performer", "audience"). When subscribing to a channel, it is also necessary to define a callback function that will typically parse the received messages. The basic function to set up a publish-subscribe mechanism is presented on Listing 1. The code of two applications are available in the open-source repository at:

`https://github.com/panavrin/tindermusic`.

The structure of the code and performance is similar to other performances that follow the same paradigm of audience participation, but our solution avoids any server implementation or any intercommunication manipulation, thanks to the advantages of the cloud service. The actual physical machines are abstracted and run in the background to realize the performance. The audio generated from that application was based entirely on Web Audio and could run on any compatible device, including many mobile devices that can run a web browser that supports Web Audio. In the following section, the performance is evaluated in terms of the network message transactions.

6. PERFORMANCE RESULTS

In April, 2015, "*Crowd in C[loud]*" was premiered at the annual concert of the Mobile Phone Ensemble, University of Michigan. Our performance lasted for 6 minutes with the sole sound from audience creations, proceeded by a performer's introduction explaining how to participate. We developed applications to log transmitted messages among connected devices during the piece, using a computer located on site and a remote server that listens to the cloud service. The two logs were compared and were identical other than the timestamps. Given the remote server was physically the farthest machine (California) from the other machines at the site of the performance hall (Michigan) and the cloud data center (Northern Virginia), it is reasonable to claim that message loss was unlikely.

The messages logged were limited to the performer and the audience channels as they were predefined and the messages are anonymous. Individual channels of audience members that were created on the fly were not logged for this study. As the purpose of this study was to understand network traffic of the performance, no attempts were made to provide logging details that would have allowed identification of individual user behavior. Rather user numbers are used to indicate overall load and temporal patterns on the network. The time window in which we analyzed the messages is defined by the time that the performer went live (or pressed the "go live" button) and lasted for the actual performance (six minutes).

The number of UUIDs created by PubNub during the performance was 184, indicating that the audience interface page was visited 184 times (including page refreshing). On the other hand, we had only 76 different screen names entered during the performance, and 69 of them send/receive messages for the actual participation. The high number of UUIDs compared to the number of screen names is because we did not restore previous sessions of UUIDs when refreshing the screen and the performer sent a message that refreshes the page right before the performance. Another discrepancy in the number of screen names (76,69) may be caused by the incompatible devices or voluntary disconnection from the participation.

Figure 2 presents the duration of individual user's participation during the performance. Each horizontal line is drawn from the time a participant submits the first message to the performer channel until the system observes the last message. Most lines starts approximately one minute after the performance started, indicating that initially participants spend time in composing their musical profiles (or composing tunes). The short line indicates that the user was active for a short amount of time. This reflects the cases in which a user refreshed the page due to the delay from the cloud and created another screen name after refreshing the page. Therefore, multiple lines from a single participant may exist.

Based on Figure 2, the total number of users connected at the same time is calculated in the Figure 3. The maximum value in the graph, which indicates the maximum number of

```
// Request an UUID
var my_id = PUBNUB.uuid();
// Initialize with Publish & Subscribe Keys
var pubnub = PUBNUB.init({
 publish_key: publishKey,
 subscribe_key: subscribeKey,
 uuid: my_id,
});

// Subscribe to a channel
pubnub.subscribe({
 channel: my_id + ",audience",
 message: parseMessage, // callback for msg
 error: function (error) {
  // Handle error here
 },
 heartbeat: 15
});

// Parse received message
function parseMessage( message ) {
 if (typeof message.type !== 'undefined'){
  if ( message.type == "create-response"){
   // Do something
  }
 } else if ...
}

// Publish a message to a channel
pubnub.publish({
 channel: "performer",
 message: {"type":"create",
  "my_id":my_id,
  "nickname": strScreenName},
 error : function(m) {
  // Handle error here
 }
});
```

Listing 1: Example of JavaScript code from PubNub API presented at audience page

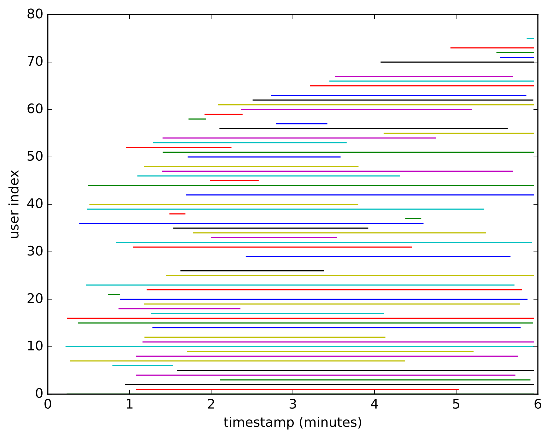

Figure 2: Participation of the audience during performance

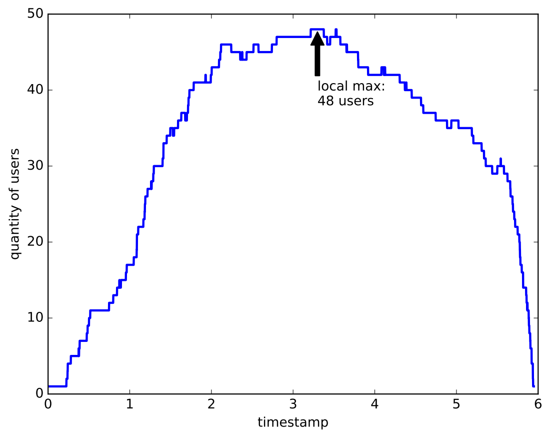

Figure 3: Number of users connected during the performance

users online at a specific moment during the performance, was 48. This reflect the practical number of participants at the performance based on our observation of the number of turnouts at the concert.

One of the significant aspects of the analysis is to develop metrics so that we can estimate maximum network traffic per second (bytes/sec) in order to evaluate the feasibility of the network setup given the scale of the performance. This will be a useful measure to secure the performance in terms of network configuration based on the estimated number of attendance and the networked interaction scheme of the musical application. Based on the log data available, all the transaction messages (that are not logged) are estimated according to the PubNub API and the message protocol that we defined. The size and number of estimated messages are close to the reality because the missing messages are in response to the request messages (that the logging application kept track of) and we can calculate the fairly accurate size of each message type based on the protocol regardless of its value.

The Figure 4 and 5 show two different metrics with which we can infer the feasibility of the network configuration. Figure 4 shows the number of transmitted messages per second, and Figure 5 presents the number of bytes in the messages sent per second during the performance. In both graphs, the peak of the graphs matches the time window that includes the peak in Figure 3. The number of messages matters because the cloud service keeps track of the number of messages and can limit the functionality by the monthly plan of choice. In the meantime, the metric in Figure 5 helps us estimate the bottleneck step in the network pipeline. While these two graphs can vary as the number of bytes per message differs based on its type (25 to 470 bytes), two graphs show similar patterns. Given that the cloud service allowed messages up to 32 kBytes, the message that the program required was at most 470 bytes($=$ 0.00047 kBytes). In addition, most network devices consider the maximum transmission unit (MTU) as 1500 bytes and divide messages into different packet, which indicates that all messages would be sent in one packet.

In total, we had 86961 bytes received by the performer, 370,630 bytes sent by the performer, and 457,591 bytes in total, during the 6 minute-performance. In terms of bandwidth, we can assure that the total data exchanged during the whole performance including entire participants did not exceed 1Mb. This information can be useful for the audience to alleviate their concern with the data usage caused by participating in the performance if they were using limited mobile data plan. Rather, loading the web interface page in the beginning constituted the largest share of data downloaded at 1.5MB. Based on our observation, the performance would not have been affected even if we chose PubNub's cheapest service plan in terms of bandwidth. Unsurprisingly, the device with the heaviest network traffic was

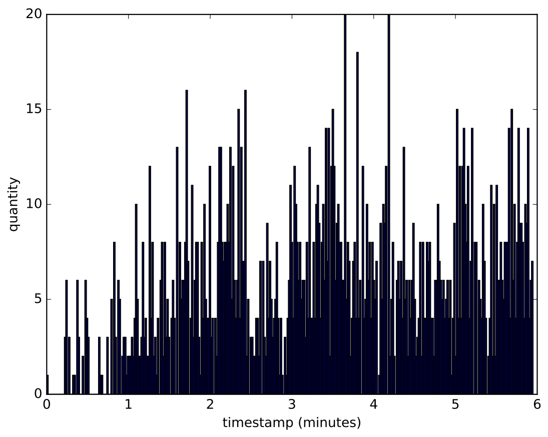

Figure 4: Histogram of the number of messages in each second during the performance (Estimated)

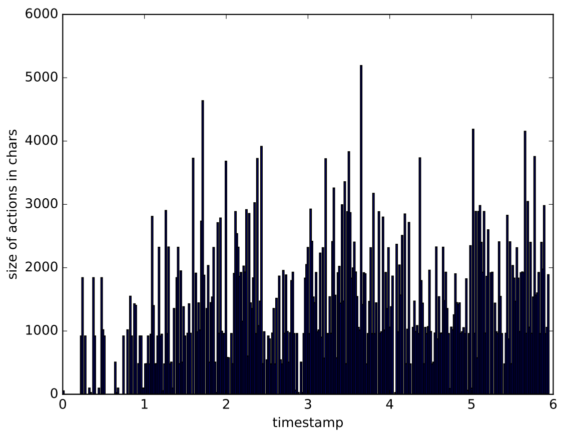

Figure 5: Size in bytes of the messages in each second during the performance (Estimated)

the performer's laptop (at under 0.5 Mb for the whole performance). However, since any delay (either computational or network) in the performer's laptop would impact all connected devices, it is recommended to use wired connection to route the data separately as well as optimize the code that is running that may contribute to the delay.

7. DISCUSSION AND CONCLUSIONS

In this paper we presented utilization of a cloud service for audience participation in musical performances. The amount of network configuration both for the developers and audience members was minimal. The network configuration using the cloud service allowed performer to mediate the orchestration of the piece and let audience members participate and collaborate with the social interaction metaphor. Our analyses on the performance log show the cloud service had more than enough capacity to service the performance and gives us some information regarding what we should consider for upcoming performances. We discovered that the bandwidth required for this performance is low given the specific interaction scheme (browsing musical profiles) among audience members. This minimize our concern of failure for heavy traffic and audience concern of data usage consumption for participation. This is particularly significant given that audience participation cannot be easily be rehearsed in realistic setting upfront.

Regarding the cloud service plans, we note that the number of messages can hardly limit the network usages of the performance, even if we would have performed everyday during the whole month. However, the number of synchronously (daily) connected devices could have been the limit and actually could have been the case if we used the free plan given our miss on restoring session of UUIDs. The dress rehearsal on the same day should have consumed a certain number of devices. In general, we can conclude that the cloud service was stable for the performance with a strong evidence of no message loss.

For the future, we plan to further investigate and improve the performance practice. First, the log application will be modified to complete all the metrics that we had to estimate in this work.Not only will this let us measure the actual network traffic but we will also be able to analyze the user experience during the performance. For example, we will be able to detect network delay for individual and dropout when a user refreshes the page due to a delay in information retrieval. Second, we are in the process of enhancing the user experience in case of contingency. Given that all the relevant code to run the musical instrument is already downloaded locally, the interactivity should have been continued while the application waits for a response from a server. Lastly, we are interested in understanding how collaboration among audience members changed their experience with the performance. Learning whether the social aspects of the instrument help the audience sustain their interest in participation is important to garner further understanding in prolonged audience engagement.

8. ACKNOWLEDGMENTS

We would like to thank the students from winter class of Mobile Music Ensembles from University of Michigan. Thanks for CAPES(Brazil) funding during the research and Emma Planet support on paper proofreading.

9. REFERENCES

[1] J. T. Allison, Y. Oh, and B. Taylor. Nexus: Collaborative performance for the masses, handling instrument interface distribution through the web. In *Proceedings of New Interfaces for Musical Expression*, 2013.

[2] Á. Barbosa. Displaced soundscapes: A survey of network systems for music and sonic art creation. *Leonardo Music Journal*, 13:53–59, 2003.

[3] L. Dahl, J. Herrera, and C. Wilkerson. Tweetdreams: Making music with the audience and the world using real-time twitter data. Citeseer.

[4] A. D. de Carvalho Junior, G. Essl, and M. G. de Queiroz. Computer music through the cloud: Evaluating a cloud service for collaborative computer music applications. In *Proceedings of the International Computer Music Conference*, Denton, Texas, 2015.

[5] A. D. de Carvalho Junior, S. W. Lee, and G. Essl. Supercopair: Collaborative live coding on supercollider through the cloud. In *International Conference on Live Coding*, 2015.

[6] A. de Souza e Silva. Art by Telephone: from static to mobile interfaces. In Lanfranco Aceti, editor, *Leonardo Electronic Almanac*, volume 20. Leonardo On-line, 2004.

[7] A. Hindle. Swarmed: Captive portals, mobile devices, and audience participation in multi-user music performance. In *Proceedings of the 13th International Conference on New Interfaces for Musical Expression*, pages 174–179, 2013.

[8] A. Hindle. Cloudorch: A portable soundcard in the cloud. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, London, United Kingdom, 2014.

[9] A. Hindle. Orchestrating your cloud-orchestra. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2015.

[10] S. W. Lee, A. D. de Carvalho Junior, and G. Essl. Crowd in c[loud]: Audience participation music with online dating metaphor using cloud service. *In Proceedings of Web Audio Conference*, 2016.

[11] S. W. Lee and G. Essl. Live coding the mobile music instrument. In *Proceedings of New Interfaces for Musical Expression (NIME)*, Daejeon, South Korea, 2013.

[12] S. W. Lee and J. Freeman. echobo: A mobile music instrument designed for audience to play. *Ann Arbor*, 1001:48109–2121.

[13] G. Levin. Dialtones - A telesymphony, September 2001, 2001.

[14] T. Riley. In c. Composition, 1964.

[15] C. Roberts and J. Kuchera-Morin. Gibber: Live coding audio in the browser. In *Proceedings of the International Computer Music Conference (ICMC)*, Ljubljana, Slovenia, 2012.

[16] G. Weinberg. Interconnected Musical Networks: Toward a Theoretical Framework. *Computer Music Journal*, 29:23–39, Jun 2005.

[17] N. Weitzner, J. Freeman, Y.-L. Chen, and S. Garrett. massmobile: towards a flexible framework for large-scale participatory collaborations in live performances. *Organised Sound*, 18(01):30–42, 2013.