# Support for live collaborative distributed coding in Node-RED: Project Paper

Student Name: S.Church

Supervisor Name: S.Bradley

Submitted as part of the degree of MEng Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

***Abstract —***
**Background/Context:** Live coding is a method of developing and deploying source code in real-time to create a performance, usually related to music. A system has been created to allow live coding of music using Node-RED; a visual block-based programming environment. This project focuses on extending this system to allow on-site collaboration in a distributed setting with the aim of allowing groups to create music performances together.

**Aims:** The aims are three-fold: to understand how existing synchronisation methods can be used to synchronise music beat generation in a distributed network of computers, where the beat rate may change over time; to explore the use of a version-control system to record and replay actions from the distributed network of computers; to analyse the use of this collaborative live coding music system in a classroom setting.

**Method:** A slave and master node have been created in Node-RED , which communicate throughout a performance to synchronise logical application clocks using berkeley's algorithm. An existing node in Node-RED has been adapted (beat - created by S.Bradley) to create slave and master variations to synchronise the beat generation and tempo changes using the logical clocks.

Git has been used to record all of the deployments and messages passed in the Node-RED system in a single repository; this repository can then be analysed and replayed programmatically onto another set of distributed computers.

Finally, a test has been conducted in a school with 10 students to analyse the effectiveness of the beat synchronisation solution and a separate test has been conducted with 5 users to analyse the recording/replaying solution.

**Results:** The results from the school experiment show how the beat synchronisation has been successfully implemented. Furthermore, it is shown how with the use of Git, a successful recording/replaying solution has been created and combined with the previously mentioned beat synchronisation solution to record collaborative music performances in Node-RED.

**Conclusions:** This project shows how computer science techniques can be used to enhance an existing system (Node-RED) to allow live collaborative distributed coding of music. It has been shown how beat synchronisation and recording/replaying have successfully been implemented; however some further modifications are needed if this system were to be used as a teaching tool in the future.

***Keywords —*** Distributed computing, Synchronisation, Live coding, Node-RED, Version control, Pedagogy

## I INTRODUCTION

### A Motivation

(Bradley 2017) explains that there is a clear lack of women working in computing, and (Microsoft 2017) has shown that this trend starts from an early age as fewer girls tend to study computer science. This is down to many factors, but the main problems are that computer science is often viewed as a socially isolating, non-collaborative subject, and girls tend to prefer more collaborative and creative work (DotDiva.org 2017).

(Bradley 2017) has created a system to help combat this which allows a user to control a music synthesis system via Node-RED (Node-RED 2017).

### B Node-RED

Node-RED is a programming tool that allows users to develop simple Internet-Of-Things (IoT) applications by using boxes (nodes) and wires to connect them (creates flows); it facilitates rapid development and allows the creation of simple applications with minimal programming knowledge. Changes to flows and nodes are only applied when the system is deployed so it acts as a small-scale insight into a normal development process. Furthermore, during run-time, messages can be passed through the system using special Inject nodes to change the behaviour without the need to re-deploy the system.

### C Live coding

Live coding of music is a fairly recent development in which performers develop a piece of music in real-time, by adding to and adapting their code as it goes along (Collins, McLean, Rohrhuber & Ward 2003). This is usually done by a single person, but this project aims to support on-site collaborative live coding in a local distributed environment of computers with the aim of allowing multiple users to run the Node-RED music generation system created by (Bradley 2017) and perform a piece of music together.

### D Beat synchronisation

Because of the collaboration while creating music, the beats produced by each computer must occur at the same time so as to create a cohesive sound. The timing of the beats at each computer is determined by the some timer based on the system clock, therefore there needs to be synchronisation between those clocks. The problem that arises is that each clock has a different rate of clock drift, defined as the rate at which the clock desynchronises (”drifts apart”) from real-time and also that the clocks may start with different values.

(Kleimola 2006) suggests that human tolerance of timing deviations in music is inversely related to tempo. Due to this, there is no definitive answer to how much difference we can tolerate between music beats coming from two different machines, but (Mki-Patola 2005) showed that the point at which the difference becomes intolerable in music performances lies somewhere in the range of 10-100ms. This highlights the need for accurately synchronising the beats between each user. Furthermore, in this system each student can change the tempo of the music, and that change must be reflected in real-time amongst all machines.

## *E Recording*

An interesting issue with live coding is how to record the performance, including all the changes made throughout, without recording the audio; the reason for not simply recording audio is that it would be useful to allow other users to replay the exact steps taken during the performance, in order to analyse, reuse or even adapt the steps to change the final music generated. Node-RED currently provides no functionality to enable the recording of the development process, which with regards to music live coding means there is no ability to record the creative process of the music performance. This project investigates how the use of a version-control system can be used to record and then programmatically replay the live coding performances generated.

The distributed nature of this system again introduces a synchronisation problem. If the clocks are not synchronised between each machine, then the recordings at each machine will have different timestamps and so replaying them would be more difficult. Furthermore, every instruction that is saved must be recorded with respect to the same clock so as to maintain the order of the instructions within the system.

## *F Synchronisation*

The above two goals of this project, namely beat synchronisation and live coding recording in Node-RED have issues relating to clock synchronisation and so this project of supporting collaborative live coding of music in Node-RED essentially reduces to a question of how existing synchronisation methods can be used to extend a Node-RED music-generation implementation to work over multiple machines in the same network.

## *G Evaluation*

This project focuses on the technical aspects of synchronising beats and recording actions within Node-RED, however as it is using the system created by (Bradley 2017) with the aim of being a teaching tool it is important to consider its use within a real-world classroom setting. For this reason, as part of the evaluation of the solution, the system was taken into a school and a combination of questionnaires and observations were used to analyse the effectiveness of the solution as part of a teaching tool.

## *H Aims and Deliverables*

**Minimum:** develop a method to synchronise both the start times and changes in beat rate among all the distributed computers that are collaborating on the same piece of music; modify or add to Node-RED in order to automatically record the deployed steps in the music generation process to a version-control repository.

**Intermediate:** improve the recording process to allow the recording of messages sent through the system during run-time (i.e recording instructions that do not require the user to deploy the Node-RED system);develop a tool to allow the replaying of the synchronised performance through analysis of the code repository.

**Advanced:** take the system into school classes to be tested, and analyse the development repository resulting from those tests to identify the ways in which the children worked collaboratively on the piece of music.

## II RELATED WORK

### *A Clock Synchronisation*

When considering clock synchronisation, there are various standard algorithms currently in use. Cristian's algorithm (Cristian 1989) is a popular solution for use in an intranet, however it relies on a time server with an accurate time source. Furthermore, it requires the system to be low-latency as the round-trip time of packets is used in the synchronisation calculations, so it is not always the best option. Berkeley's algorithm (Gusella & Zatti 1989) is also designed for use within an intranet, but can function when no machine has an accurate time source. It relies on choosing one of the processes involved to act as the master, using Cristian's algorithm to get the slaves' clock times and averaging them, and finally informing each of the slaves of how much they must adjust their clock to match. Berkeley's algorithm is more robust than Cristian's due to the fact that if the chosen master server fails, a new master server can be chosen via any number of distributed election algorithms.

For use in a distributed system over the internet, NTP (Network Time Protocol) is one of the most notable algorithms in use (Mills 1991). It has been designed to deal with variable-latency networks and has been shown to maintain clock accuracy to within a few tens of milliseconds.

It was not immediately clear which, if any, of the previously mentioned algorithms would be sufficient for this solution and so more research into specific beat synchronisation solutions was needed.

### *B Beat Synchronisation*

Some work has been done in this area with similar systems, most notably EspGrid, introduced by Ogborn (Ogborn 2012), and Gabber, introduced by C.Roberts (Roberts, Yerkes, Bazo & Kuchera-Morin 2015).

EspGrid is both a protocol and an application, designed for use within a live networked orchestra ensemble, to allow users to synchronise the tempo of the music without worrying about the underlying clock synchronisation involved.

Gabber is an extension for Gibber (Charlie Roberts 2017) which is a browser-based Javascript live coding system.

The reports on both of these systems detail how clock synchronisation can be used to achieve the desired beat synchronisation for live coding music performances, however the implementations differ in a few key areas discussed below.

#### B.1 Architecture

Gabber (Roberts et al. 2015) relies on a centralized network, meaning having a master node that drives the synchronisation of all other slave nodes' clocks. On the other hand, EspGrid

(Ogborn 2012) assumes a decentralized system and uses a peer-to-peer architecture, with each node in the distributed network running a version of the software, without the need for a master node.

A peer-to-peer architecture has the benefit that there is no single point of failure, however there may be some added complexity to deciding how to synchronise the clocks, and also an increase in the probability of an inaccurate clock affecting the synchronisation. A centralized architecture is easier to implement and although it will offer limited fault tolerance, this was not a concern for this project and so a centralised architecture was chosen for the solution.

## B.2 Clock type

There are two main options when considering exactly what to synchronise: the system clock, or some logical clock defined in the application. Neither method on its own would be enough to synchronise the beats as the specific timings of each beat would still need to be synchronised between all computers relative to the synchronised clock.

Many clock synchronisation algorithms focus on synchronising system clocks, and EspGrid (Ogborn 2012) follows this method. As an alternative solution, Gabber (Roberts et al. 2015) defines application level audio clocks and chooses to synchronise those instead of the system clocks. This approach is perhaps more usable in terms of applying it to different systems, as not all machines have access to a reliable system clock. Furthermore, they explain how a driving factor for the choice was that using application level clocks means no other software has to be run in the background simultaneously, which aids ease of use. For this project, ease of use was an important factor to consider as the aim was to eventually use the system in a school; using a logical clock in the Node-RED application allows more room for teaching the concepts of clock synchronisation.

## B.3 UDP vs TCP

For communication in the network, there is a choice to make about whether it is more appropriate to use User Datagram Protocol (UDP) or Transmission Control Protocol (TCP). Across large sections of the internet, TCP is more reliable due to its retransmission policy and the fact that it is connection-oriented. However, in a small system such as this, the choice of which protocol to use relies more on which of the following constraints is most important. If transmission time is not important but reliability is, then TCP is again the better choice, however in a real-time system where transmissions must be fast and efficient, UDP is often the better choice.

EspGrid (Ogborn 2012) uses UDP but Gabber (Roberts et al. 2015) uses TCP connections for its synchronisation, and there is not enough evidence as to which performs better. However, it is suggested that the choice of protocol can affect the results of using a given algorithm. For example, EspGrid relies on Cristian's algorithm, but the report on Gabber details how they attempted to implement Cristian's algorithm in a similar way, but found that due to the use of TCP as opposed to UDP, they obtained worse results which is something to consider when choosing a solution.

In order to test this further, both Cristian's and Berkeley's algorithms were implemented using both TCP and UDP. The synchronisation ran in rounds, where a round consists of all clocks applying an offset, and rounds occurred every 2 seconds. The test involved using two servers on one machine and the results are shown below:

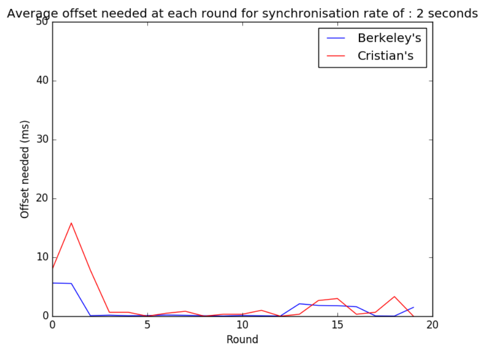


*Figure 1: The average offset needed at each round, using a synchronisation rate of 2 seconds and TCP*

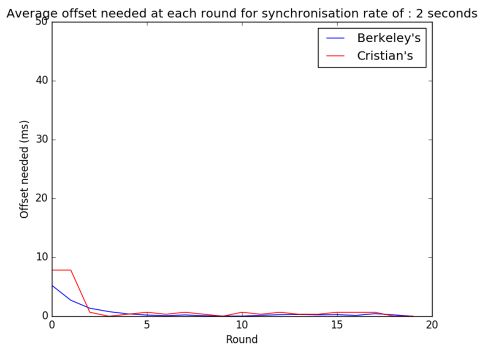


*Figure 2: The average offset needed at each round, using a synchronisation rate of 2 seconds and UDP*

These results indicate that UDP will provide more reliable synchronisation as on average, each clock had to apply a smaller offset than when using TCP, and using UDP results in less variation in offsets applied.

Furthermore, this system is designed for use in a classroom setting, and will therefore be run in a local area network; this means the reliability of the network will be quite high and so UDP will suffice.

## *C Replay history*

One main challenge of live coding is the difficulty of recording it. It is obviously possible to simply use an audio recorder to record the music but it would be better if there were some way to record and then replay the exact steps the performer took when coding it. This stems from the fact having a full recording of all the steps taken in a performance allows for the ability to reuse or even edit performances. There has been very little work done in this area, but the previously discussed report on Gabber (Roberts et al. 2015) briefly mentions how it saves all the versions of each users' code in a database, to keep track of any changes and potentially be able to replay those changes. This shows how it is feasible to accomplish, but raises the question of whether a database is the best thing to use, or if it may be better to use a full version-control management system.

One main advantage of using a database is that you can use a local database, removing the need for any access to an online system. However, the biggest problem is that you would have to design a specific database for each type of code or application you wanted to record and any changes would require more work to implement later in the process. The main benefit to using a full version-control system is that they are purposefully designed to make it easier to keep track of file histories and include simpler ways to retrieve that history.

If choosing a version control system, there is still a choice to make between decentralized version control systems (DVCS) or centralized systems (CVCS). Git (Loeliger & McCullough 2012) is an example of a DVCS, which has benefits over CVCSs such as SVN due to the fact that DVCSs can better handle frequent commits and provide the ability to work offline due to storing the whole repository history locally. The disadvantage of this is that as the repository grows, DVCSs may experience more performance issues than CVCSs which only store local snapshots of the repository at any given time (Mulu, Bird, Nagappan & Czerwonka 2014), however in this system using live-coding, where music performances are relatively short, repository size is not be an issue.

### *D Educational use in schools*

When considering evaluating the use of the system in a school, it is important to examine similar tests of other systems in order to gain an understanding of the best things to analyse and the best ways to test its use.

One of the most recent related reports concerns BlockyTalky (Shapiro, Kelly, Ahrens, Fiebrink & others 2016) and details how tests were conducted during summer camps to analyse its use as an educational tool. The students involved were in the age range 11-14 and the two summer camps were 20 hours each, taking place over multiple days. Multi-day workshops are a popular way of testing these systems as it allows more time for teaching and allows the students to become familiar with the system involved. In contrast, a report on Networked DrumSteps (Jennings & Tangney 2005) consists of a test in a school classroom, taking place during one day. This has the benefit of being able to do more tests over a shorter period of time, and also allows you to see how the system may be used in an actual school, rather than just a dedicated workshop. However, it is also important to note that the tests with Networked DrumSteps only involved two students as opposed to the other report which used more students. It is clearly best to test with as many students as possibly as everyone will interact with the system in a different way.

In terms of testing, both the tests with BlockyTalky and Networked DrumSteps used observations of the students' interactions with each other while using the systems to gain an understanding of its general use. Furthermore, they also both included follow-up interviews with the students involved to get a better understanding of how they felt using the system. Surveys were also used by BlockyTalky, with open-ended prompts such as ”The best thing about this workshop was”. These may seem subjective, however they were analysed and certain themes were extracted such as ”Music” and ”Programming” to try to gain a better understanding of the important aspects that the students picked up on, so in that regard it was useful.

It is clear that for this project a combination of observations and questionnaires/surveys was the most appropriate way to test the system in a classroom setting.

## III SOLUTION

### *A Tools*

The system that was used for live coding is Node-RED (Node-RED 2017), which is described in Section I. It runs on a Node.js server which can be deployed remotely or locally, and there is a large community that supports its ongoing development by contributing new nodes or flows. During run-time, every time a change is made to the visual data, the system must be re-deployed but messages can be injected into the system using special inject nodes without re-deploying. New nodes can be written by creating specific JavaScript and HTML files.

In the system created by (Bradley 2017), SuperCollider was being used for audio synthesis (SuperCollider 2017).

For testing purposes, a RaspberryPi was used, however the system should work on any computer running Node-RED. For the test in schools (described later), pi-tops were used to turn the RaspberryPis into easy-to-use laptops.

For version control Git was used (Git 2017): a free, open-source distributed version control system designed to handle projects with speed and efficiency. When using Git, users commit to a local repository and then push those changes to a global repository, so each user has access to a full local copy which will be used to update the global one at a later date.

In terms of programmatically interacting with Git to commit, push and then eventually get a full history, a Node.js module called NodeGit was used (NodeGit 2018).

### *B Architecture*

The system runs in a local distributed network of computers, all on the same LAN. Each user has their own computer running Node-RED - with the relevant music nodes installed - and SuperCollider running. The Node-RED instances communicate with SuperCollider via OSC messages which are sent using the UDP from a Node-RED node, and the users communicate with the Node-RED servers through a browser interface.

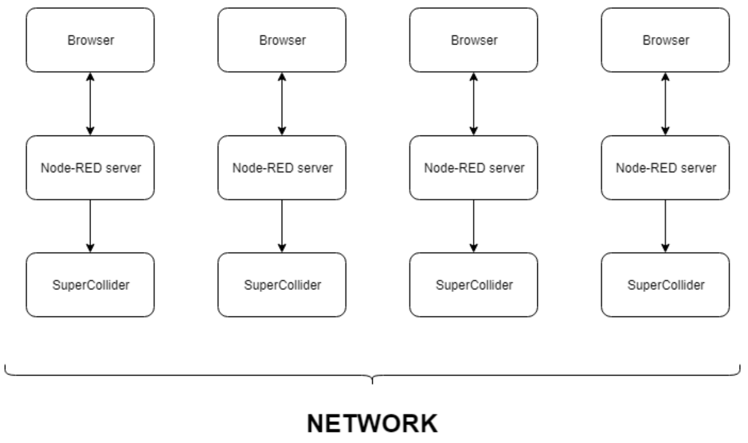


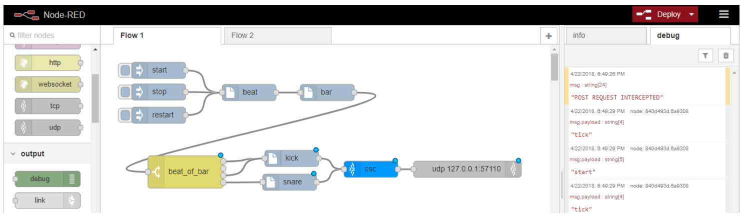
*Figure 3: Architecture of the system*

For beat synchronisation, communication is performed between Node-RED servers on each machine using UDP and one machine is chosen to be the master by the user in the Node-RED browser application.

Each computer has its own local repository copy of the performance and pushes changes to the same global repository. The main files saved are retrieved from the Node-RED servers, and these files represent the flows currently deployed by each computer, and a list of all previous instructions from each user where an instruction can be one of two forms: a deploy instruction or an inject instruction. These instructions are stored in a JSON instruction file for each user. Finally, when replaying the recording, the system uses POST requests to the running Node-RED servers to replay the exact deploy and inject instructions from each user's instruction file.

### *C Beat synchronisation solution*

After considering all the issues highlighted in Section II B, the synchronisation solution chosen was based on a centralized network architecture, using a logical clock and UDP for message passing. Having decided this, the next step involved considering how to incorporate this into the existing Node-RED system created by (Bradley 2017). The most basic set-up to create music in this system is shown in figure 4.



```
4/22/2018, 6:49:26 PM
msg : string[24]
"POST REQUEST INTERCEPTED"
4/22/2018, 6:49:29 PM   node: 840d493d.6a9308
msg.payload : string[4]
"tick"
4/22/2018, 6:49:29 PM   node: 840d493d.6a9308
msg.payload : string[5]
"start"
4/22/2018, 6:49:29 PM   node: 840d493d.6a9308
msg.payload : string[4]
"tick"
```

*Figure 4: Basic use of the music generation system*

In the original system, the beat generation was controlled by a ”beat” node, which calculates the timings of all the beats given a start time and beats-per-minute (bpm) value and then sends a message to the next node in the flow. All of the timings are calculated relative to the system clock; for this solution a new synchronised logical clock was defined that can be accessed by all nodes that need to synchronise.

## C.1 Berkeley nodes

A ”berkeley-master” and ”berkeley-slave” node have been created in Node-RED, by creating the relevant javascript and html files for each. The aim of these nodes is to synchronise the logical clocks between each user’s computer. Each node sets a global ”clock” variable that can be accessed by every node on the same Node-RED server. This ”clock” variable is a reference to an object, which stores an offset value which is added to the current system time to obtain the logical clock time. The object also has two useful functions including getCurrentTime and updateOffset which allow any node to access the current time and update it; however, only the Berkeley nodes make use of the updating capability. Both nodes simply run Berkeley’s algorithm to synchronise the logical clocks, and this algorithm is described in more detail in the next section. Multiple ”berkeley-slave” nodes can connect to a single ”berkeley-master” node using its IP address, and they communicate using the UDP. When a ”berkeley-slave” is deployed, it joins a group (connects to the master) by sending a UDP message to the chosen ”berkeley-master” node (the master is chosen by typing in an IP address in the ”berkeley-slave” node in the browser), which allows the ”berkeley-master” to keep a list of all the slaves connected to it. The clock variable these nodes set allows all nodes to synchronise their actions relative to the same logical clock.

## C.2 Berkeley’s Algorithm

Berkeley’s algorithm, (Gusella & Zatti 1989), works as follows:

- The master process, M, requests the time from all slave processes.
- M uses the same approach as in Cristian’s algorithm to obtain up-to-date estimates of the times from each slave (i.e adds half of the round-trip-time to the results)
- M then calculates an average of the times received and its own time, ignoring any clock times that are significantly far outside of the others (using an f-value relating to how many of the highest and lowest clock times to ignore)

- If the number of slaves is greater than 2, the f-value is equal to the closest multiple of 10 (rounding down) plus 1 (e.g if there are 4 slaves, the f-value is 1, if there are 14 slaves the f-value is 2 etc.)
  - This has the effect that if 2 computers are already synchronised and a 3rd one joins, the f-value will be 1, and this 3rd clock will be the most out-of-sync and so will definitely be ignored when calculating the average for this round, and so the 3rd computer's clock will be synchronised properly with the master's time before the second round where it may contribute to the average; this stops any spikes in clock changes at already synchronised clocks.

- For each slave process, M calculates the difference between the slave's time and the average calculated in the previous step

- M sends the corresponding difference value to each slave process so they can update their clock times accordingly

- Finally, M updates its own time to match the average

The important thing to note about this algorithm is how it uses differences/offsets to let the slaves know how to adjust their clocks. This eliminates the uncertainty from the round-trip time that would arise if the master sent the calculated average time. Furthermore, the averaging function can be improved and changed to potentially improve the synchronisation.

While testing the beat synchronisation using this method, the occasional audio glitch was heard, which was believed to be caused by software processing time jitter, with the end result that the clocks were being given a much larger offset than needed at random points through the performance. In order to fix this, a change was proposed to the Berkeley's algorithm and is outlined below:
A limit-value (LV) is given a starting value of 20000ms, and in a given round of synchronisation, all offsets applied are forced to be less than this LV. In a given round, if any offset for any slave is greater than the LV, then the LV for the next round will be 50% larger than the current one. If no slave offsets are larger than the LV, then the next LV will be half as large.

This change was intended to smooth out the Berkeley synchronisation and not allow any huge spikes of offsets applied. This implementation was compared to the original Berkeley algorithm using two machines, with synchronisation rounds taking place every 2 seconds, and both methods were tested in the Node-RED system both without music playing and with music playing (playing music uses UDP messages so congests the network more). Only the results for the tests with music playing are shown in figures 5 and 6.

These results show how the change to the Berkeley algorithm smooths out the synchronisation. If there is constant congestion in the network, the LV will increase and eventually the offsets will match the required level, but short spikes are ignored as these may be due to either network congestion or software jitter. However, one thing to note is that through this testing the audio glitch did not occur and so even without the limit, the synchronisation is within 1ms after initial convergence which is enough for this project. More testing is needed during a full performance to fully understand the effects (if any) of this change to the algorithm.

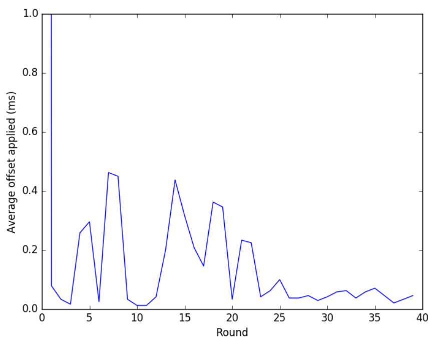


*Figure 5: Berkeley algorithm in Node-RED without using limitValue and without playing music*

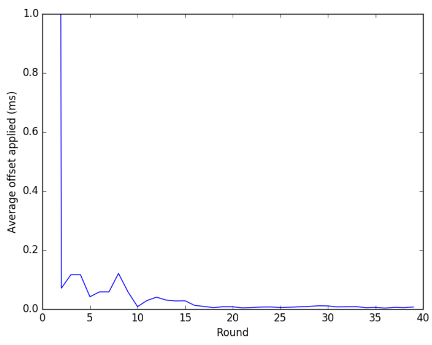


*Figure 6: Berkeley algorithm in Node-RED without using limitValue and without playing music*

When testing for the first time with 3 pi-tops, one main issue arose. The synchronisation worked well between 2 of the pi-tops, however when a third computer joined the network, it did not synchronise well. Under closer inspection, this happened because as the first two computers synchronised, they lowered the LV to around 5ms and the computer joining was out-of-sync by about 2 minutes (i.e 120000ms), so it was taking far too long for the computer to reach synchronisation. To solve this problem, three solutions were proposed:

**Solution 1:** When a berkeley-slave joins a berkeley-master node, the master should send its current logical time to the slave so the slave can just update its clock to be the same.

**Solution 2:** Create a dictionary linking IP addresses to how many rounds of synchronisation they have been involved in; add a new entry for each slave and update every round. If an IP address of a slave has been in 0 rounds of synchronisation, ignore the limitValue for this IP address when setting the offset (will make it the same time as the other clocks for the next round).

**Solution 3:** Store a list of all the slaves that have joined but not had any rounds of synchronisation. When synchronising clocks and sending offsets, if an IP address of a slave is in the list, ignore the limitValue and then remove it from the list.

Solution 1 is the simplest, however due to the use of UDP, this message may get lost and so the original problem would not be solved. Solution 2 is viable, however the master has no way of knowing if a slave has disconnected, so if a slave were to disconnect and then reconnect (i.e reset its logical clock), the value for how many rounds of synchronisation it had been involved in would not be 0 and so the original problem would reoccur. Solution 3 solves these two issues and so this is the change that was implemented in the final solution.

## C.3 Beat synchronisation nodes

Two new nodes were created: ”beat-slave” and ”beat-master”, both of which are modified versions of the original ”beat” node. Both nodes have been extended to use the logical clock described in C.1 and are capable of performing beats on their own without synchronisation. The ”beat-slave” can choose any other ”beat-master” node to synchronise with, leaving the possibility of having more than one synchronised group on the same network. For communication, each node sets up its own UDP server, capable of sending to and receiving messages from all other UDP servers in the system. Similarly to the Berkeley nodes, the slave sends a UDP message to the master on deployment so the master can keep track of all its slaves.

The ”beat-master” is essentially the same as the ”beat” node, however, whenever it performs a beat, it sends a UDP message to any attached ”beat-slave” nodes with the following information: the time the beat started relative to the global logical clock, the number of the beat (i.e the number of the beat within a given bar), the bpm value, and the latency value (this value determines how far in the future the beats are performed and is used to remove software jitter). Using this information, the ”beat-slave” nodes can calculate the timings of the beats, using the same values the master uses. Furthermore, the ”beat-slave” nodes can compare the master’s latency value to their own and update it if necessary.

In terms of synchronising the tempo changes, one considered solution was to simply send a UDP message to the ”beat-slaves” whenever the ”beat-master” changed its tempo. However, due to the possibility of lost messages that occur with the use of UDP it was decided that the best solution was to send the current tempo value along with the message described earlier whenever a beat is performed; the slaves then compare this value to their tempo value, changing it if necessary. Through testing, one issue that came up was what should happen if a slave tries to change the tempo or latency values while connected to a master. To solve this issue, a method was implemented in the ”beat-slave” node to periodically check to see if the current slave is connected to a ”beat-master”. If it is connected, any changes to bpm and latency on the slave machine are ignored and the master’s values are maintained.

The ”beat-slave” and ”beat-master” nodes were combined into one ”beat-combined” node which has one extra parameter so a user can decide whether to be a master or a slave for a performance. Throughout the paper, any references to ”beat-master” and ”beat-slave” are meant for clarification and both are actually implemented within the ”beat-combined” node.

## *D Storage implementation*

From the related work it was clear that a DVCS was an appropriate method to use for recording the performances and so for this reason, the solution uses Git to save the steps that contribute to the final Node-RED music performance.
Arguably the most important aspect of this requirement is choosing exactly what information to store and how to retrieve it as enough information must be stored to almost perfectly replay not only the music generated by each user, but also the steps involved.

As shown by figure 4, the music is generated when a UDP message is sent to 127.0.0.1:57110, which is the default IP address of the SuperCollider (music synthesiser) server running on the same machine. This indicates that the best way to accurately record the exact audio would be to simply save every UDP message sent from that final node. This would have the benefit of only needing to synchronise the times used by the UDP nodes between each machine and would achieve the most accurate recording. However, this solution was not used as it is not future-proof or usable with any other Node-RED flow: if the communication protocol or audio-synthesis method was changed a new solution would be needed, and furthermore, if you wanted to record other actions in the system, that would not be possible.

The solution chosen instead relies on two important features of Node-RED: any time a change is made to the flows in the browser, Node-RED must be re-deployed so that the servers have an up-to-date copy of the flows; in this solution, the only action that can be performed without re-deploying the system is injecting a message through the flows using an Inject node. The recording solution takes advantage of these features by only recording a copy of the flows after every re-deployment, and a copy of every inject message. Both deployment and injection instructions are saved in an instruction file for each machine, and the flows data is also saved in a separate file so that it may be retrieved separately if needed. The flows files and the instructions files for each user in the performance are saved to the same global git repository using the IP address of the machine.

In the Node-RED system, the flows are represented by JSON objects and the user's machine stores a local copy of the flows which is updated on re-deployment. The exact steps are as follows: the user re-deploys Node-RED; the Node-RED servers make a POST request using the JSON representation of the flows which then saves the flows JSON in local storage and updates the browser version of the flows. The solution takes advantage of this by intercepting the POST request - by modifying the Node-RED api source code for dealing with flows - and adding in a step to send the flows JSON to a saveInstructions.js file which saves the flows JSON in the local repository, along with a new instruction referencing the deployment, and then pushes both to the remote repository.

Once a flow is deployed, a user must use a specific node called an Inject node to send a message through the flow in order to perform some action. In the system designed by (Bradley 2017), this can include the following injection messages: ”start”, ”stop”, ”reset”, and ”bpm”, however this solution should be able to deal with any messages to be future-proof. In the original Inject nodes, when the user presses the button attached to the node in the browser, a POST request is sent which is dealt with by the node itself. A modified Inject node was created and has the same functionality as the original, with the extension that whenever the node is triggered and a POST message is received, the unique node ID is sent to a saveInstructions.js file which stores a new inject instruction in the instructions file. As before, the instructions file is then committed and pushed to the Git repository.

All instructions are saved with the IP address of the machine they came from and a timestamp which is correct relative to the logical clocks and described in more detail below. Multiple bands can save their instructions in the same global Git repository. The ”beat-master” node has an extra parameter for the ”band name” which is sent to all ”beat-slave” nodes in that group when they join the band. This name is used as a directory name within the global repository to separate the bands.

## D.1 Synchronising

The main issue with recording was how to make sure that all instructions were recorded relative to the same clock, so as to maintain the order of events. When dealing with this synchronisation issue, the solution uses the same Berkeley nodes described in section C.1. In this way, all inject nodes retrieve their time from the same source and so will be synchronised and in the correct order. One potential problem with this that did not arise in the beat synchronisation solution is that of negative offsets. If one clock is running faster than the others and requires a negative offset to synchronise it then there is the potential for instructions to appear out of order. The most obvious solution to this would be to implement a method so that negative offsets never occur. However, time constraints meant that this was not implemented. In practice, because of the limited-offset solution described in C.2, any negative offset is relatively small and so no instructions happen close enough together for this to be a problem.

### D.2 Replaying

Once a performance has been successfully recorded, there must be a method to replay it. Firstly, using NodeGit, a method has been implemented to retrieve a full history of the Git repository, saving all the files from each stage of the repository in folders named using the commit sha IDs. The user can then specify the group performance to retrieve instructions from. All of the instruction files (one per machine) in the given group are then parsed, creating a list of instructions for each user, indexed by timestamps. The instruction objects are then combined into one object, and the timestamps updated to be relative to the timestamp of the first instruction so that they start from time 0. Finally, the instructions are replayed based on timestamps and at each stage; the next instruction is retrieved and then sent to one of two functions depending on if it is a deploy or inject instruction. If it is a deploy instruction, the flows json data is extracted from the instruction and if it is an inject instruction, the node ID is extracted. In both cases, the IP address is extracted and a relevant POST request made to the Node-RED server (running on the given IP address) to replay the instruction. A method has also been implemented to allow the replaying of a subset of the users' performance so a specific contribution can be isolated, and furthermore, the performance can also be replayed on different IP addresses than the original ones and so can be replayed at any time. This is achieved by modifying the parsing instructions step: when an instruction is retrieved, the IP address is replaced by the new IP address of the computer that will be playing this user's contribution, and whenever a flow is retrieved, the JSON data is parsed and all relevant IP addresses replaced to make sure the synchronisation nodes set up UDP servers on the correct machines and send messages as intended.

Through testing it became clear that one main issue was that replaying a performance simply repeated exactly the steps taken in the original performance, and Node-RED had no way to know if this was a new performance or a replaying and so would save the instructions from the recording again. To solve this, a new header was added to every inject and deployment POST request called "Node-RED-SaveDeployment". Within the Node-RED source code itself, this value is set to true by default, but when programmatically replaying the performance, the POST requests are sent with this header set to false. Users also have the option to turn off recording altogether by setting a savePerformance boolean to false within a customSettings.js file which was added to the Node-RED user directory.

## *E Implementation issues*

One main implementation issue that became clear later in the project is that of using GitHub for hosting Git servers. This worked perfectly on the wireless, internet-connected network used throughout the development process of this solution, however when taking the system into schools for evaluation, a local network was used and so the GitHub servers could not be reached. To solve this, some time had to be spent setting up a local Git server, which meant that the recording solution was not ready for the test in schools. Fortunately, all of the recording code used one main settings file to find the location of the remote git server, and so once the new local git servers were set up it was simply a case of replacing the URL of the repository in the settings file and the recording then worked again.

Throughout the process, the solution focused on synchronising the timings of the beats, however later testing of the system using more complicated beat and bar configurations highlighted the need to also synchronise the bars because even if the beats are synchronised, if two identical bars start on different beats the music sounds non-synchronised. To fix this, a simple modification was made to the original divider node that separates beats into bars; the solution now uses the beat counter provided by the beat node, as this number will be the same for all synchronised nodes.

## IV RESULTS

### *A Results*

This project has some technical aspects that can be tested with regards to the clock synchronisation part of the solution. However, the majority of features are best tested with a user study as music and also timing intolerances in beat synchronisation are very subjective, and furthermore, the recording and replaying solution would be very difficult to test numerically. This is best tested in a user study to gain an understanding of if it performs its intended role. With this in mind, and considering that this project aims also to support the goal of (Bradley 2017) which is to use the system in schools to encourage young girls to get involved in computer science, it was clear that the best evaluation method available was to perform a user test in a classroom setting.

### *B User test*

Having looked at various other sources of creative computer science systems being taken into schools for analysis, it was intended to take this system into a school for two sessions, each lasting up to one hour, and use a post-questionnaire as well as general observations to gather data on how well the students thought the solution worked. Due to adverse conditions, including the unexpected snow storm, and a lack of equipment for testing (sound cards and pi-tops), the project was only able to be taken into a school for one of the two intended sessions. This will obviously have an impact on the results as more users would produce more reliable results and the session could potentially have been fine-tuned for the second attempt.

The session took place in a secondary school as part of an after-school code club and involved 12 students in an age range 11-14. All students and parents were given an information sheet detailing the experiment and gave prior consent to the test taking place.

### *C Post-questionnaire*

The questionnaire was designed to be taken immediately after the test and included both Likert-type questions and general feedback questions. To analyse the results it was simply a case of analysing the scores of the Likert-type questions, and for the general questions the idea was to see if any common responses or themes were included in the students' responses.

### *D Questionnaire results*

Due to circumstances mentioned earlier, only one session took place, with this session lasting 1 hour and involving 10 students (only 7 students gave results). Futhermore, the recording solution required access to the Internet, which was not available during the school test and so this part of the solution was not tested in the session. The questionnaire results for the Likert-type questions are shown below:

## D.1 Beat and tempo synchronisation

It would be ideal if there were a way to exactly measure the difference in beats produced at each machine, so that an exact measure in milliseconds could be attributed to the level of beat synchronisation reached. However, due to the fact that the beats are synchronised using a local logical clock and the actual system clocks are never synchronised, it is not viable to do this. Therefore, the results must come from the questionnaire. The question, ”The beat generation sounded synchronised”, was the most helpful in terms of determining whether, in a classroom setting, the beat synchronisation solution would work. This question encompasses three separate parts of the project: beat synchronisation; tempo synchronisation; bar synchronisation. The results are shown in the figures below:

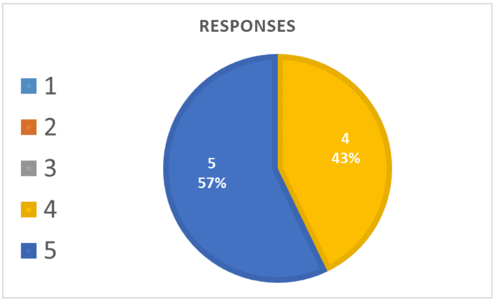


*Figure 7: Likert-type responses from the prompt: ”The beat generation sounded synchronised”*

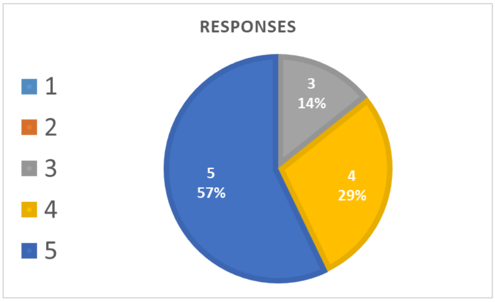


*Figure 8: Likert-type responses from the prompt: ”It was simple to create music”*

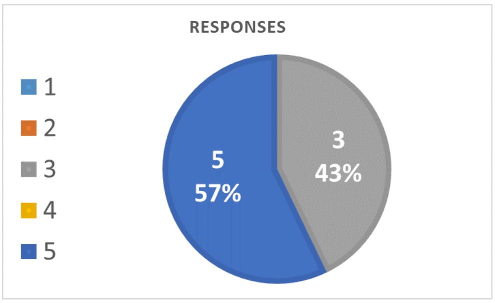


*Figure 9: Likert-type responses from the prompt: ”I learnt computer science techniques using this system”*

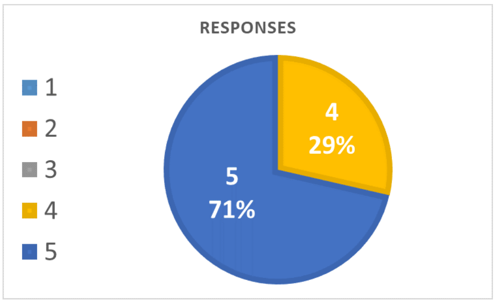


*Figure 10: Likert-type responses from the prompt: ”I would like to use this system again for creating music”*

Figures 8, 9, 10 all relate to the wider implications of this work to aid the goal of (Bradley 2017) and will not be analysed in more detail in this paper. Figure 7 shows that all of the participating students agreed or strongly agreed that the beat generation was synchronised. This result is backed up by observations made during the test which found that when the set-up was complete and every student using the correct nodes, the beats were synchronised and some interesting, complex music was created with multiple different instruments and bar structures.

## D.2 Recording/Replaying

A separate test was conducted to determine the correctness of the recording and replaying solution. The test was very similar to the school test and involved 5 users, but with one difference that it used a wireless network instead of a wired one, however having tested the system on a wired network with 3 computers it is clear this is no problem.

The question used to determine the effectiveness of the recording solution must encompass the subjective nature of music in that two performances of the same piece of music can differ slightly but still sound like the same piece of music, and so the question chosen was: ”The replaying solution sounded the same as the initial performance”. The results are shown below:

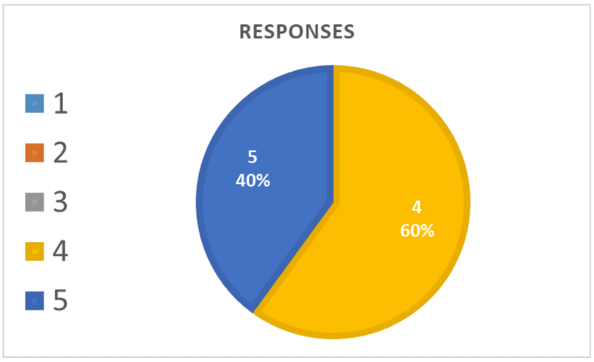


*Figure 11: Likert-type responses from the prompt: ”The replaying solution sounded the same as the initial performance”*

This figure shows how the majority of users thought the recording solution was perfect. The disparity in results, with two users responding with a 4, comes from the subjective nature of the music, and the fact that it was perhaps difficult to determine exactly how similar the solutions were. One of those two participants indicated that they could not fully remember the exact timings of the original piece, in particular due to beat synchronisation solution as there are a few beats that are not synchronised before the synchronisation fully works and it was not clear if that period of lack-of-synchronisation was exactly the same in the replaying. However, even with this slight uncertainty they still agreed that it sounded similar to the original performance and so it is clear the solution works.

## D.3 General observations

Throughout the test the students were observed and the main things to note are that the solution seemed to technically work, however certain issues arose that highlighted how more work would need to be done to introduce the system as a permanent teaching tool. The main issue revolved around setting up the equipment, with several hardware and software faults arising; this is further highlighted as an issue by one student who mentioned ”setting up” as the worst thing about the session. Furthermore, the test took place in one room, with two bands of 5 students, with each student having their own speaker; naturally this led to a problem of too much noise to fully hear the performances which detracted from the experiment. However, the students appreciated the experimental nature of the system, as shown by the fact that almost all of the responses to the quesion ”what was the best thing about the session?” involved some notion of experimenting to create music. Finally, the test brought to light some potential problems and further work: a few students struggled to understand the need for the Berkeley nodes to be placed on separate flows and it is clear this needs to be more obvious, or perhaps a more intuitive method created to remove this added complexity.

## V EVALUATION

### *A Strengths*

The results from the user test clearly showcase a few key strengths of the solution. Firstly, the students agreed that the beat synchronisation works, and through observations it was clear that the tempo changes, beat generation timings and bar start times were being synchronised to allow users to successfully create music collaboratively using Node-RED. Furthermore, the recording solution works as intended, allowing for a full collaborative performance to be saved to Git and replayed at a later date. Another strength of the recording solution is that it can theoretically be used with any Node-RED application and so with further work to improve the user-friendliness of the replaying code, it would be a valuable tool during any Node-RED development process.

### *B Weaknesses/Limitations*

As well as highlighting strengths of the solution, the user test brought light to some of the weaknesses, some of which had already been identified earlier in the testing process as potential problems. The main problem is that of sound: during the test, having 10 users in the same room all using speakers resulted in a hard-to-hear environment which made it difficult to fully collaborate on music. If beats were instantaneous (i.e, no latency), all members of the group could simply send UDP messages to the SuperCollider servers on all other members' machines, thereby allowing each user to listen to the whole performance through headphones. However, due to the problem of propagation delay through the Node-RED flow, latency must be added to beats to schedule them for the future, so software jitter can be minimised and synchronisation maintained. This causes the problem that as SuperCollider uses system clocks and the time tags associated with the beats are scheduled relative to these system clocks; the system clocks are not synchronised and so sending these messages to other SuperCollider servers would not result in the beats being played at the correct times. This is a problem that results from using a logical clock solution; this solution simply synchronises the times at which the beat() functions are called for each user, but not the times at which the actual beat sound is generated.

One other highlighted problem was the intuitiveness of the Berkeley nodes; it is not immediately clear without an explanation why they are needed or especially why they are needed on a separate flow; further work should include making this more obvious or incorporating the clock synchronisation as part of the beat synchronisation so that this is not an issue.

A further issue with the replaying concerns replaying a subset of the performance. There is the ability to only play certain users' contributions to a performance by simply just replaying their instruction files and ignoring the rest. However, due to the fact it is the actions (i.e deployments and injections) that are saved and not the specific timings of the sounds produced, if the subset of users that is being replayed does not include the master, then the slaves will not synchronise anything when their actions are replayed and so the replaying will not be accurate. This could have been solved by using the UDP saving method mentioned in section D, however as explained in that section the solution does not use that method due to its inflexibility and inability to be used for any Node-RED application.

### *C Organisation of project*

The general organisation of this project was good, with time evenly split between creating the synchronisation method and recording solution in the early stages. Furthermore, there was efficient use of the iterative approach to add new features to recording and replaying section as the project progressed. Although the recording and replaying code was organised and modularised well, the modularisation of the beat synchronisation code was less well planned meaning that refactoring had to later be done. Furthermore, the initial plans did not consider how difficult it would be to test this solution as various problems arose (such as not being able to use UDP communication on the university Wi-Fi), and the plans also did not take into account the need to build multiple machines in order to properly test the solution and the majority of testing in the early stages was performed on one machine; this naturally led to new issues being discovered late in the project timeline when multiple machines were used for testing.

## VI CONCLUSIONS

This project involved using computer science synchronisation methods, along with a version control system, to modify an existing system created by (Bradley 2017) using Node-RED, to allow the collaborative live coding of music. The results and evaluation have shown how using a traditional, popular synchronisation algorithm (berkeley's algorithm) with a minor change to synchronise logical clocks is a sufficient basis for beat synchronisation within this system. Furthermore, the results have shown how this same Berkeley-based solution can be used alongside a version control repository to accurately record the actions taken by users on multiple machines so as to record a full performance in Node-RED and then be able to replay the performance at a later date. The recording solution works well for this purpose, and due to its nature could easily be used to record any projects created in Node-RED and so could potentially form part of a version control system for any Node-RED application. The user test in a school setting has shown that with some small changes the solution could be used as a teaching tool and potentially aid the goal of (Bradley 2017) to encourage young girls to get involved in computer science by providing a simple-to-use, creative, collaborative live-coding environment.

As further work, potentially the use of another clock synchronisation algorithm could be tested, especially if these synchronisation algorithms were to be used as a teaching tool later down the line. Furthermore, work should be done to allow the use of headphones so all users can hear the full performance on their machine. Finally, as an extension to the recording solution, a more user-friendly application could be designed to allow users to visually replay, analyse and edit old performances as part of a new performance in Node-RED.

## References

Bradley (2017), Collaborative Creative Computing with the Internet of Things, *in* 'Proceedings of 12th Workshop in Primary and Secondary Computing Education', Nijmegen, The Netherlands.

Charlie Roberts (2017), 'http://charlie-roberts.com/gibber/about-gibber/'.

Collins, N., McLean, A., Rohrhuber, J. & Ward, A. (2003), 'Live coding in laptop performance', *Organised sound* **8**(3), 321.

Cristian, F. (1989), 'Probabilistic clock synchronization', *Distributed Computing* **3**(3), 146–158.

DotDiva.org (2017), 'Dot Diva | Educators & Parents | Whats the Problem with Girls and Computing? - http://dotdiva.org/educators/problem.html'.

Git (2017), 'Git'.
**URL:** *https://git-scm.com/*

Gusella, R. & Zatti, S. (1989), 'The accuracy of the clock synchronization achieved by TEMPO in Berkeley UNIX 4.3bsd', *IEEE Transactions on Software Engineering* **15**(7), 847–853.

Jennings, K. & Tangney, B. (2005), 'Virtual Collaborative Learning Environments for Music: Networked Drumsteps', *Comput. Educ.* **44**(2), 173–195.

Kleimola, J. (2006), 'Latency issues in distributed musical performance', *Telecommunications Software and Multimedia Laboratory* .

Loeliger, J. & McCullough, M. (2012), *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*, "O'Reilly Media, Inc.".

Microsoft (2017), 'Why dont European girls like science or technology? - Microsoft News Centre - https://news.microsoft.com/europe/features/dont-european-girls-like-science-technology/'.

Mills, D. L. (1991), 'Internet time synchronization: the network time protocol', *IEEE Transactions on Communications* **39**(10), 1482–1493.

Mki-Patola, T. (2005), 'Musical effects of latency', *Suomen Musiikintutkijoiden* **9**, 82–85.

Mulu, K., Bird, C., Nagappan, N. & Czerwonka, J. (2014), Transition from centralized to decentralized version control systems: a case study on reasons, barriers, and outcomes, ACM Press, pp. 334–344.

Node-RED (2017), 'Node-RED : About - https://nodered.org/about/'.

NodeGit (2018), 'NodeGit - http://www.nodegit.org/'.
**URL:** *http://www.nodegit.org/*

Ogborn, D. (2012), Espgrid: a protocol for participatory electronic ensemble performance, *in* 'Audio Engineering Society Convention 133', Audio Engineering Society.

Roberts, C., Yerkes, K., Bazo, D. & Kuchera-Morin, J. (2015), Sharing Time and Code in a Browser-Based Live Coding Environment, *in* 'Proceedings of the First International Conference on Live Coding. ICSRiM, University of Leeds, Leeds, UK', pp. 179–185.

Shapiro, R. B., Kelly, A., Ahrens, M., Fiebrink, R. & others (2016), 'BlockyTalky: A Physical and Distributed Computer Music Toolkit for Kids'.

SuperCollider (2017), 'SuperCollider'.
**URL:** *http://supercollider.github.io/*